PRÉFACE

Le modèle des réseaux associatifs a d'abord été défini par A. MÉRIGOT à l'*Institut d'Électronique Fondamentale* en 1992, dans un contexte de multi-processeurs dédiés au traitement d'images de bas et de moyen niveau. Ce modèle exploite le parallélisme de données et permet de mettre en œuvre des traitements sur des zones irrégulières, qui sont définies à l'aide de graphes dynamiquement modifiables. Il s'est révélé riche dans ses applications algorithmiques et l'étude de la Maille Associative a montré qu'il pouvait être mis en œuvre de façon très efficace dans une architecture électronique incorporant des idées novatrices (reconfigurabilité, asynchronisme).

Mais les primitives et le mode de programmation du modèle ont tôt fait de lui révéler des qualités d'«expressivité» qui ont incité à l'appliquer à d'autres domaines que le traitement des images. Dans cette optique, A. MÉRIGOT m'a proposé d'étudier les possibilités d'une généralisation du modèle dans le cadre de mon stage de DEA [Ducourthial, 1995]. Cette étude a été organisée selon quatre axes : les comparaisons avec des modèles plus connus, l'étude de la topologie sous-jacente (la maille), les routages dans la maille et des algorithmes dans d'autres domaines que l'analyse d'images. Divers aspects de ce travail (plongements, routage) m'ont amené à travailler avec D. SOTTEAU au *Laboratoire de Recherche en Informatique*. Ce stage a donné lieu au sujet de ma thèse, commencée en octobre 1996, sous la direction de D. SOTTEAU et A. MÉRIGOT, dans le cadre d'une collaboration entre leurs laboratoires respectifs.

Évolution de la problématique. Le sujet initial mentionne quatre voies de recherche : l'étude du modèle des réseaux associatifs en tant que modèle de calcul théorique, l'évaluation de ses aptitudes au calcul parallèle, la recherche des topologies adaptées à son fonctionnement et sa mise en œuvre.

Mon premier travail a donc été l'étude du modèle, et de son implémentation dans la Maille Associative. De cette analyse est venue l'idée d'une redéfinition partielle du modèle ; elle allait changer l'optique du travail. L'introduction d'une nouvelle primitive de programmation (`direct-association`) a intégré l'asynchronisme de la Maille au cœur même du modèle. Cette primitive réalise un calcul global sur un réseau à l'aide de calculs locaux concurrents et non coordonnés, jusqu'à stabilisation complète des nœuds et des liens du réseau. Elle n'est définie que si le calcul global se termine. Il s'agit d'une primitive de bas niveau, qui peut être mise en œuvre sans difficulté. Restait à savoir si elle était pratique à utiliser en programmation.

La seconde phase de mon travail fut donc l'étude des opérateurs utilisables avec la nouvelle primitive. Elle a abouti à la définition des r-opérateurs. Ces opérateurs assurent la convergence de calculs globaux asynchrones sur tout type de topologie. Ils ont révélé d'intéressantes propriétés tant théoriques que pratiques. Ainsi ils donnent au modèle des propriétés de tolérance aux fautes tant sur les liens que sur les nœuds (duplication et désordonnancement des messages, auto-stabilisation du réseau en cas de pannes). De plus, ils ont permis d'exploiter l'asynchronisme de la nouvelle primitive et ainsi d'améliorer sensiblement divers algorithmes dans le modèle.

Ce faisant, ils ont contribué au développement d'une philosophie de programmation asynchrone. En effet, les performances algorithmiques des r-opérateurs, et la maîtrise de l'asynchronisme par le modèle, ont incité à développer des algorithmes dans l'optique asynchrone, plutôt que de traduire des algorithmes synchrones dans un modèle asynchrone performant, comme cela était proposé avec la première définition du modèle. J'ai ainsi étudié les algorithmes d'arbre en largeur et en profondeur, de la transformée en distance, de la segmentation de Voronoï, de la morphologie mathématique...

Restait donc la mise en œuvre du modèle redéfini, qui seule permettait d'expérimenter et de valider la philosophie de programmation asynchrone. Partant de la bibliothèque séquentielle de simulation de la Maille Associative (`SAM`) écrite par A. MÉRIGOT, j'ai envisagé une bibliothèque de programmation parallèle des réseaux associatifs, basée sur l'environnement *multithread* PM^2 [Namyst, 1997] : `ANET`. Sa réalisation a donné lieu au stage de Magistère de N. SICARD [Sicard, 1998b].

La bibliothèque `ANET` est une hiérarchie de classes C++ et de méthodes permettant de définir un réseau associatif de topologie quelconque et d'y appliquer les primitives du modèle. Elle parallélise la `direct-association` en respectant l'irrégularité des objets et en répartissant les calculs sur plusieurs *threads*, lesquels peuvent s'exécuter à distance sur d'autres machines (fonctionnalités de PM^2). Ainsi, `ANET` manie des objets irréguliers, autorise la gestion de la charge irrégulière grâce à la souplesse des *threads*, et met en œuvre la primitive `direct-association` qui réalise des calculs locaux asynchrones sur des portions irrégulières des objets, jusqu'à stabilisation. Cette bibliothèque a pour l'instant été testée sur réseau Ethernet et Myrinet de stations de travail (PC / Linux).

Résultats. Au final, si les réseaux associatifs restent un modèle adapté au traitement des images, ils offrent un paradigme de programmation original qui conduit à des programmes simples pour des algorithmes complexes : une seule `direct-association` permet d'effectuer un parcours en largeur, ou en profondeur, de calculer une distance dans une image, ou d'obtenir les polygones de Voronoï, ceci sans pour autant négliger le temps d'exécution. Le ***modèle des réseaux associatifs*** peut ainsi être défini comme un ***modèle de programmation à parallélisme de données, pour algorithmes et données irréguliers, à primitives de calcul asynchrones***.

Les objectifs du sujet initial sont donc atteints : le modèle a été étudié au niveau théorique ; son évaluation en programmation parallèle a été menée ; il a été montré que la topologie de la maille permet d'utiliser toute autre topologie[a], et la bibliothèque `ANET` propose une implémentation du modèle.

Document. Le texte est organisé en trois parties, reflétant chacune l'une des phases ($\approx$ 9 mois / phase) de mon travail. La première partie est consacrée à la définition du modèle. La seconde concerne son étude et la dernière traite de ses applications et de sa mise en œuvre.

Pour mener à bien cette étude, des résultats et des méthodes de divers domaines ont été utilisés, mixés et confrontés. J'avais, au début de ma thèse, exprimé le souhait d'un sujet large, ce que facilitait la collaboration entre le LRI et l'IEF. J'ai été très largement satisfait puisque des domaines aussi divers que le calcul parallèle, les architectures de machines, le traitement d'images, la théorie des graphes, l'algèbre, l'algorithmique parallèle et distribuée, les systèmes distribués et l'auto-stabilisation sont abordés dans ce document.

Leur utilisation est cependant justifiée — et nécessitée — par les besoins de l'étude des réseaux associatifs, fil conducteur de ce travail. Il n'est pas anormal que l'étude d'un modèle se voulant généraliste fasse appel à des domaines très divers, et la cohérence de l'ensemble apparaît clairement. L'abord de ces divers domaines a été, sinon un parti pris, du moins l'une des difficultés de ce travail. Pour faciliter la lecture de ce document, de nombreuses définitions sont placées dans la marge pour que le lecteur à qui elles sont familières puisse les éviter. Un index permet de les retrouver rapidement.

Remerciements. J'ai bénéficié durant ce travail d'une allocation de recherche de la formation doctorale en *Architecture Parallèle*, dirigée par le Professeur D. Étiemble. Je l'en remercie. Mes remerciement vont aussi aux membres du Laboratoire de Recherche en Informatique et de l'Institut d'Électronique Fondamentale, aux équipes *Architecture*, *Graphes et Communications*, *Parallélisme*, et *Architecture et Conception des Circuits Intégrés et des Systèmes*.

Ma thèse résulte d'une collaboration entre deux laboratoires, et deux équipes, aux préoccupations et à la pratique de la recherche assez éloignées. Je tiens à remercier D. Sotteau et A. Mérigot pour

(a) Ce travail, commencé durant mon stage de DEA, a été poursuivi durant mon service militaire, pour être achevé au début de ma thèse.

l'intelligence des relations que nous avons entretenues, leurs conseils avisés et leur grande ouverture d'esprit. A. Mérigot encadre mes recherches depuis le stage de DEA. Ma vision des modèles informatiques associant abstraction et mise en œuvre lui doivent beaucoup. Je voudrais le remercier pour la confiance qu'il m'a témoignée. Dominique Sotteau a suivi de près mon travail, et notamment ses développements théoriques. Elle a beaucoup contribué à en améliorer la rigueur. Ma rédaction scientifique — et notamment anglaise — a largement bénéficié de ses conseils.

Je voudrais aussi remercier les nombreuses personnes qui m'ont conseillé, aidé, ou avec qui j'ai directement travaillé, et en particulier : E. Angel, C. Arcile, J.-P. Allouche, D. Barth, P. Berthomé, P. Bosch, J. Beauquier, B. Charron-Bost, G. Constantinescu, C. Delorme, S. Delaët, Y. Denneulin, D. Dulac, M. Gondran, M.-C. Heydemann, M. Raynal, O. Richard, J.-F. Méhaut, R. Morin, R. Namyst, N. Sicard, S. Tixeuil et V. Zissimopoulos.

Les Professeurs J. Beauquier (LRI, Orsay), J.-M. Geib (LIFL, Lille) et J.-M. Jolion (INSA, Lyon) ont accepté de participer au jury. Je les en remercie vivement. Le Professeur L. Bougé (LIP, ENS Lyon) et Michel Gondran (DER EDF / GDF) ont accepté de rapporter sur ma thèse ; je leur adresse mes sincères remerciements pour l'intérêt qu'ils ont porté à mon travail. Ils ont contribué à l'amélioration de ce document.

La réalisation d'un tel travail en temps compté nécessite un soutien, une organisation et des motivations sur le long terme. Merci Elodie pour ton aide. J'ai rédigé pendant qu'Alice dort, qu'Alice rit ou qu'Alice pleure. Avec Alice sur les genoux, sur le clavier, ou devant l'écran (`xfishtank` !). J'ai certainement délaissé beaucoup d'amis durant cette période ; je tiens à m'excuser auprès d'eux. Je ne voulais pas (trop) délaisser ma famille.

Sommaire

INTRODUCTION

Notre étude ayant trait à la définition et l'étude d'un nouveau modèle de calcul, nous commençons dans un première partie par présenter, de façon très générale, le rôle et les problématiques attachées au modèle informatique, ainsi que les principaux termes du domaine.

Dans une seconde partie, nous introduisons les thèmes de recherche actuels, et les objectifs de notre travail.

Du modèle en informatique

Suivant E. Dion, on pourra remarquer que les progrès de l'humanité sont tout autant jalonnés de découvertes quant à la domestication de l'énergie que d'avancées dans le traitement de l'information [Dion, 1997]. On notera en ce qui concerne la maîtrise et l'utilisation de l'énergie — et sans vouloir être complet — la découverte du feu (-450000 ans avant J.-C.), de la roue puis de la voile, du charbon, des moulins à vent puis des roues à aubes (12$^{\text{ème}}$s.), de la machine à vapeur (D. Papin en 1679), de l'électricité (A. Volta en 1800), du moteur à explosion (A. Dion, 1889), de l'énergie atomique...

Pour la maîtrise et l'utilisation de l'information, on relèvera l'apparition de la parole, la naissance de l'écriture (Summer, -4500 av. J.-C.), l'invention d'un alphabet à nombre réduit de symboles (Gueret en Syrie, -1500 av. J.-C.), les premiers journaux manuscrits («Acta Diurna», Rome 59 av. J.-C.), l'imprimerie (Gutenberg, 1440), la calculatrice mécanique (B. Pascal), la feuille perforée (métier à tisser, B. Bouchon, 1725) puis le métier à tisser automatique et programmable (J.-M. Jacquard, 1800), l'algèbre de Boole, le téléphone (A.G. Bell, 1876), la machine de Babbage (1910), la machine de Turing (1936), l'ENIAC, premier ordinateur (J.P. Eckert, J.W. Mauchly, 1944), la machine de Von Neumann (1945), le transistor (J. Bardeen, W. Brattain et W. Shockley, 1948), le Fortran, premier langage de programmation (J. Backus, 1956), le processeur (1972)...

Ainsi, à la montée en puissance de l'information au XX$^{\text{ème}}$ siècle (apparition du transistor et développement de l'ordinateur, des réseaux, des supports numériques...), on peut associer la révolution industrielle du XVIII-XIX$^{\text{ème}}$ siècle (apparition du coke et développement de la machine à vapeur, du chemin de fer puis de l'automobile...).

Il y a donc un parallèle saisissant entre ces deux entités, que sont l'énergie et l'information, et qui commence par la difficulté qu'il y a de les définir.

De même que la physique traite de l'énergie et de ses échanges, l'informatique peut se définir comme la science étudiant le traitement de l'information. Tout comme celui de l'énergie, le traitement de l'information se caractérise par des échanges et des mouvements (communications), qui résultent de la transformation de l'information fournie en entrée en vue de la production d'un résultat en sortie.

Cette notion de mouvement distingue l'informatique des mathématiques. À l'«objet mathématique» intrinsèquement statique, on pourrait substituer la notion d'«objet informatique». Celui-ci modéliserait des *calculs* sur des données, et des *communications* entre ces données et les éléments chargés de les traiter.

Modèle informatique. Comme toute science, l'informatique se définit des modèles, qui sont autant de sujets d'études mais qui facilitent les classements, tant pour le chercheur que pour l'utilisateur final. Nous restreindrons notre présentation aux modèles émanant des nombreuses façons de coder l'information et de la diversité des moyens de la traiter.

En règle générale, le modèle a pour but de résumer les caractéristiques pertinentes et essentielles de tel ou tel phénomène dans le but de faciliter l'analyse et la prédiction. Le modèle en informatique est principalement utilisé pour comparer, ou pour le moins, préciser ce qui peut l'être : le modèle sert alors de fédérateur et détermine des classes d'objets semblables qu'il est légitime de comparer. Cela permet, par exemple, de comparer la complexité d'un nouvel algorithme à celle de ceux précédemment connus, définis pour un type de machines donné, et programmés avec une technique ou méthodologie donnée, c'est-à-dire suivant un modèle donné.

Le second avantage essentiel du modèle est d'augmenter la portée d'un travail. Un programme écrit pour une machine particulière ne pourra pas, *a priori*, être réutilisé sur une autre machine sauf s'il n'est pas écrit «trop près de la machine» (c'est-à-dire en utilisant des caractéristiques propres à la machine). Le programme aura intérêt à être décrit dans un langage «compris» par plusieurs machines, c'est-à-dire selon un modèle générique de programmation. On parle alors de programmes *portables*, c'est-à-dire qui peuvent être portés d'une machine à une autre sans trop de modifications. Le modèle pourra manquer de précision sur tel ou tel point, mais le programmeur aura gagné du temps, d'une part parce que son programme sera, d'emblée, utilisable sur différentes machines, et d'autre part parce que le modèle lui aura épargné la prise en compte de paramètres pointus trop spécifiques à la machine elle-même.

Parallélisme. La notion de parallélisme — que l'on peut définir comme étant l'art de réaliser plusieurs traitements à la fois — a grandement contribué à la prolifération des modèles informatiques.

L'emploi du parallélisme est d'abord justifié par des besoins de performances pour satisfaire des délais. Les premières contraintes de délais étaient liées à des problèmes de fiabilité : on cherchait à effectuer le maximum de traitements dans le laps de temps séparant deux pannes. Actuellement, les contraintes de délai sont dues à la nature *temps-réel* de problèmes tels que la prévision météorologique, l'interprétation de photographies satellites de régions critiques, la réaction à un signal dans un système auto-asservi, *etc.* Une autre contrainte de délai qui mérite d'être signalée est l'homme. Les données physiologiques impliquent en effet des délais, pour respecter notamment la cadence des 24 images par seconde qui donnent l'impression de mouvement, ou celle de la «réponse instantanée» qui ne doit pas, en réalité, dépasser la seconde.

La taille des problèmes est généralement la seconde justification du parallélisme. Que ce soit pour traiter des phénomènes physiques toujours plus complexes, ou pour obtenir toujours plus de précisions (nombre de décimales) sur un phénomène précis, la physique — pour ne citer qu'elle — transforme tout progrès dans la vitesse des traitements des machines en accroissement de la quantité de calcul à réaliser.

Citons également la nature intrinsèquement parallèle de certains problèmes (éléments finis ou traitement d'images par exemple) qui gagneront en clarté à être programmés selon un modèle parallèle offrant les constructions adéquates. Pour finir, signalons le besoin de puissance en tant que phénomène économique, voire sociologique : une machine rapide sera plus vite rentabilisée parce qu'elle réalisera plus de travail, et la rapidité d'une réponse à un quelconque service est un plus indéniable dans une société où la vitesse est érigée en qualité.

La recherche de la puissance a d'abord été confiée à l'électronique des processeurs et des mémoires, en constant progrès depuis les années soixante. Cependant on voit poindre les limites du gain offert par le processeur : la taille de la grille des transistors se rapproche petit à petit de la limite théorique, et fondre de gros circuits posera toujours des problèmes d'ingénierie complexes. Aussi s'est-on tourné vers le parallélisme, qui seul pourrait, à terme, satisfaire les besoins de puissance cités plus haut.

Les débuts du parallélisme remontent aux recherches sur la bombe atomique, dont les calculs étaient confiés à des salles d'opérateurs humains, armés de machines à calculer mécaniques [Sabot, 1988]. Les opérateurs effectuaient des morceaux élémentaires de calcul, et se passaient des fiches avec les résultats. L'ensemble était assez souple car l'opérateur pouvait corriger de lui-même les petites fautes dans le programme parallèle ; mais les fautes importantes se traduisaient par un brouhaha d'opérateurs confus. L'une des premières machines parallèles, le Cmmp pour *Computer with multiple mini-processors*, fut construite à l'université Carnegie-Mellon (USA) dans les années 1970 [de Rumeur, 1994]. Depuis cette époque, de nombreuses machines ont été étudiées.

On distingue généralement trois formes de parallélisme. Le *parallélisme de flux* correspond au flux des données sur les unités de calcul, qui sont ainsi utilisées au maximum, et simultanément. Le *parallélisme de contrôle* consiste à découper les traitements à réaliser en entités, appelées *tâches*, et à gérer au mieux le placement de ces tâches sur les ressources de la machine, de telle sorte que les ressources soient utilisées simultanément et le mieux possible. Le *parallélisme de données*, quant à lui, est l'art d'effectuer simultanément le même traitement sur des données différentes : si la machine séquentielle travaille sur des scalaires, la machine à parallélisme de données travaillera, elle, sur des ensembles de scalaires, organisés, par exemple, en tableaux.

Modèle et parallélisme. Si l'on peut trouver différentes façons de programmer une machine séquentielle, l'unicité du processeur limite cependant les variantes, et le modèle «optimal» est vite trouvé. Le parallélisme, au contraire, a très largement multiplié les possibilités : nombre de processeurs, agencement, réseau de communication, mémoire partagée ou répartie, contrôle centralisé ou distribué, etc. La quête du modèle optimal en parallélisme reste encore du domaine de la recherche. En fait, les analyses concordent sur le fait qu'un modèle unique est illusoire, tant la diversité des problèmes et des machines pour les résoudre est grande.

Pour [Maggs et al., 1995], le modèle doit être un juste équilibre entre simplicité et précision, abstraction et pratique, description et prescription. Nous retiendrons pour notre travail que le modèle spécifie la façon dont les traitements sont organisés pour abstraire, et offrir au programmeur des schémas de traitement et des primitives indépendants de la machine.

Le modèle doit donc être de haut niveau pour simplifier la tâche du programmeur, et de bas niveau pour être suffisamment pertinent et précis sur telle ou telle machine. Il y a donc une ambivalence inhérente à la notion de modèle informatique, sensé être tout à la fois de haut et de bas niveau. Cette ambivalence ne facilite d'ailleurs pas la tâche d'évaluation du modèle. En effet, s'il est facile de constater qu'un modèle est efficace (rapide) — ce qui est, en général, plus ou moins lié au fait qu'il n'est pas trop éloigné de la machine —, il reste difficile de jauger de sa facilité d'utilisation pour la conception d'algorithmes, car cette dernière mesure inclut un facteur humain.

Différents types de modèles. Il existe beaucoup de taxinomie de modèles, ce qui traduit certainement le fait qu'aucune d'entre elles n'est réellement satisfaisante [Duncan, 1990] [Sansonnet, 1994] [Hambrusch, 1996] [Demeure et Méhaut, 1997]. Suivant l'article de synthèse [Maggs et al., 1995], nous pouvons définir trois classes de modèles.

On trouve tout d'abord les *modèles de programmation* qui permettent d'écrire un programme sans le destiner à une machine précise. Le programmeur utilise pour cela un langage de programmation qui est une *instanciation* du modèle de programmation. Les modèles *data-parallèles* [Marquet, 1993] et *à passage de messages* [Calvin et Colombet, 1994] sont les plus connus. Les premiers sont dédiés à l'exploitation du parallélisme de données ; le langage Fortran 90 en est une instanciation parmi d'autres. Le second — qui est parfois assimilé à un modèle d'exécution car il reste de bas niveau — exploite plutôt le parallélisme de contrôle. Le langage Occam en est une instanciation, ainsi qu'Ada. Mais il est plutôt utilisé à l'aide de bibliothèques de communications telles que PVM, MPI, ou bien LANDA [Guyaux et al., 1997].

Historiquement, les langages de programmation ont été développés en privilégiant l'*expressivité*. Le programmeur avait à sa disposition des constructions élégantes, et parfois prouvables, pour traduire ses intentions algorithmiques. Cependant les langages de haut niveau doivent se traduire en instructions dépendant de la machine. Ce processus de traduction est facilité par les *modèles d'exécution* (b)

(b) Bien que le modèle d'exécution soit aussi appelé *modèle de calcul* par certains auteurs, nous réservons ce dernier

[Demeure et Méhaut, 1997]. Un modèle d'exécution doit définir la façon dont les traitements sont mis en œuvre. En particulier, il donne des informations sur le déroulement des calculs, sur la gestion des communications, sur le contrôle des différentes ressources, etc.

Finalement, les différentes machines sont regroupées selon des *modèles architecturaux* ou modèles de machines [Duncan, 1990]. Comme nous l'avons déjà signalé, la diversité des modèles de machine vient de la multiplicité des unités de calcul : les modèles reflètent l'organisation des ressources que sont les processeurs, les mémoires, les réseaux, etc. On citera en particulier, en utilisant un classement par autonomie du processeur [Charot, 1993], les machines SIMD (même instruction pour tous les processeurs), telles que les tableaux de processeurs, les machines MIMD (les processeurs sont indépendants les uns des autres) [Duncan, 1990] et les machines «virtuelles» composées de stations de travail reliées par un réseau [Cappello et Richard, 1998].

Modèles pour le parallélisme

Exploiter le parallélisme. Les différents modèles qui ont été développés en *informatique parallèle*, et les hiérarchies de modèles que nous avons évoquées, n'ont d'autre but que de capter le maximum de parallélisme pour un problème algorithmique donné, en vue d'une exécution sur une machine donnée.

Le temps d'exécution parallèle théorique d'un algorithme, calculé en divisant le temps d'exécution obtenu sur un seul processeur par le nombre de processeurs utilisés dans la version parallèle, est souvent bien au-dessous de ce qui est constaté dans la pratique. Les pertes sont dues à l'application en elle-même, qui impose des contraintes de «séquentialité» entre des phases qui sont, elles, parallélisables. Mais une fois le parallélisme de l'application exhibé, il reste à l'exploiter. Or, et les modèles utilisés, et les machines employées, prennent, comme autant de taxes, leur part sur ce parallélisme maximal.

C'est en fait la recherche de l'exploitation maximale du parallélisme qui justifie l'étude de nouvelles machines, ainsi que de nouveaux modèles ; ces derniers doivent rendre accessibles au programmeur les fonctionnalités et le parallélisme de la machine, sans trop lui complexifier la tâche.

Parallélisme de flux. Le parallélisme de flux n'est généralement pas proposé au programmeur. Il relève principalement du matériel (processeurs pipelinés) ou de dispositifs particuliers, tels que les chaînes de traitement d'images, qui sont alimentées par un *flux* d'images en provenance de capteurs.

Citons aussi certaines machines spécialisées, dites *systoliques*, par allusion aux systoles du cœur, pour lesquelles un modèle de programmation adéquat permet d'exploiter le flux des données en entrée de l'ordinateur [Charot, 1993] [Sansonnet, 1994].

Parallélisme de contrôle. Le parallélisme de contrôle est couramment utilisé dans la programmation parallèle. Il se traduit par des programmes découpés en plusieurs *processus*, qui doivent (éventuellement) communiquer entre eux. Cette communication est principalement assurée par envoi de messages.

La programmation par passage de messages présente cependant des inconvénients, qui la destinent au spécialiste. La principale difficulté consiste à coordonner les morceaux d'un programme unique, qui évoluent indépendamment les uns des autres, et qui n'ont donc aucune raison d'aller à la même vitesse. La coordination des morceaux nécessite un délai de transmission des messages, pendant lequel la situation des processus peut évoluer. Il faut alors prendre garde que les processus ne s'attendent pas tous mutuellement, conduisant alors à un inter-blocage. Pour éviter les problématiques des systèmes distribués [Raynal et Hélary, 1988], et rechercher une meilleure performance, on met en place des *barrières de synchronisation*, à l'aide de sémaphores. Elles sont cependant très coûteuses en temps [Axelrod, 1986] ; la situation est améliorée par des dispositifs matériels adéquats [Hoare et al., 1996].

---

terme pour désigner tout modèle décrivant des calculs sur des données, indépendamment de la classification hiérarchique présentée ici.

Les solutions préconisées actuellement consistent à entrelacer des phases de calculs sur chacun des processus et des phases de communications globales entre les processus, assurant ainsi leur synchronisation [Hambrusch, 1996]. Il reste alors à équilibrer la longueur des phases de calcul par rapport aux délais de communication pour que l'ensemble, bien retardé par les synchronisations, reste performant. Il faut aussi jouer avec le nombre de processus, car un trop grand morcellement de l'application entraîne inéluctablement un surcoût de communication. On parle de *grain* de l'application.

Les modèles s'évertuent alors à mesurer avec précision le coût du réseau (latence et débit) par rapport au coût des calculs [Hambrusch, 1996]. En fait, chaque application à programmer devient un problème de recherche, la preuve en est le nombre d'articles ne traitant de la parallélisation que d'un algorithme.

Parallélisme de données. Selon [Dekeyser et Marquet, 1996], on peut appeler langage à parallélisme de données tout langage dans lequel une instruction implique plusieurs traitements identiques sur les données d'un ensemble. Le modèle à parallélisme de données offre donc au programmeur des primitives définies sur des ensembles (vecteurs et matrices le plus couramment), qui se traduisent par la duplication du traitement autant de fois qu'il y a de données dans la structure spécifiée. On utilise l'anglicisme «data-parallèle» pour qualifier ces primitives. Nous nous concentrons maintenant sur ce modèle.

Le modèle de programmation à parallélisme de données

Origines. Avant que la réflexion sur les modèles ne fût suffisamment abstraite, tous les modèles informatiques étaient plus ou moins liés à une machine [Dekeyser et Marquet, 1996] [Bougé, 1996]. Le modèle à parallélisme de données ne déroge pas à cette règle, même si le modèle théorique du tableau de cellules identiques et travaillant de concert, imaginé par J. von Neumann dans les années soixante [Kutrib et al., 1997], peut être considéré comme le premier modèle à parallélisme de données.

L'origine du travail sur tableaux vient du calcul scientifique et des architectures vectorielles, telles que le Cray 1 et le Cyber 205, qui furent optimisées pour traiter rapidement ces structures et les calculs associés [Dekeyser et Marquet, 1996]. Ce paradigme de programmation devint ensuite le modèle de prédilection des architectures en tableau, depuis l'Illiac IV (mis en service en 1975) jusqu'à la MASPAR (1990). L'heure de gloire de ces machines fut sans doute l'époque des Connections Machines CM 2 (1988) puis CM 200 (1990) [Thinking Machine Corporation, 1990], avec des études très intéressantes sur les modèles à parallélismes de données [Hillis, 1985] [Sabot, 1988] [Blelloch, 1990b] [Lakshmivarahan et Dhall, 1994].

Mais l'augmentation de la puissance des processeurs standards a rendu caduque l'idée d'assembler de grandes quantités de processeurs simples, et l'abandon des Connections Machines a sonné le glas de ces architectures qui se voulaient généralistes. Des machines de type MIMD leur ont succédé, et les machines SIMD sont maintenant cantonnées à des applications bien précises, telles que le traitement d'images.

Évolutions. Le modèle à parallélisme de données ne fut pas abandonné pour autant. Tout d'abord les machines à mémoire partagée ont continué de s'en accommoder sans difficulté. Ensuite, plusieurs équipes ont étudié la simulation de cette mémoire partagée à partir des mémoires distribuées, avec des environnements logiciels [Demeure et Méhaut, 1997] [Cappello et Richard, 1998] ou matériels [Priol, 1997] [Houzet et Fatni, 1998]. Enfin, et surtout, un nouvel axe de recherche s'est rapidement développé : celui de la compilation et de l'exécution de programmes à parallélisme de données sur des machines à mémoire distribuée. Il a donné lieu à beaucoup de travaux, dont les plus connus sont ceux consacrés au Fortran [Merlin, 1994] [Marquet, 1997]. Les dernières évolutions du Fortran sont symptomatiques des réflexions dans ce domaine.

Tout d'abord, les programmes écrits en Fortran séquentiel (dernière norme en 1977) manipulent essentiellement des tableaux. Pour paralléliser ce type de programme, deux pistes sont possibles. La première consiste à paralléliser automatiquement les programmes séquentiels en transformant les boucles sur les indices des tableaux en instructions data-parallèles. De nombreux travaux ont été réalisés dans cette voie, mais les difficultés rencontrées, notamment avec l'irrégularité des données, ne peuvent que la destiner à la traduction de programmes déjà écrits, et non à une philosophie de programmation parallèle, qui aurait les avantages de la facilité : écriture séquentielle puis parallélisation automatique.

La seconde voie consiste à fournir au programmeur des primitives data-parallèles, pour qu'il indique lui-même au compilateur les passages contenant du parallélisme dans son programme. On quitte le *parallélisme implicite* pour le *parallélisme explicite*.

Ce faisant, on est confronté au problème de la *machine virtuelle* : les données manipulées par le programmeur sont généralement de plus grande taille que le nombre de processeurs. Les langages offrent donc au programmeur une machine virtuelle idyllique, correspondant à la taille de ses données. Mais il faut ensuite plaquer cette machine virtuelle sur le réseau des processeurs. On trouve alors presque autant de solutions que de langages data-parallèles, qui se répartissent en deux grandes classes : ceux dont la machine virtuelle est d'*instanciation* et ceux pour lesquels elle est d'*alignement*. Les premiers n'admettent que des objets ayant la topologie de la machine virtuelle (langages C$\star$, DPCE, Hyper C), tandis que les seconds autorisent des objets de formes diverses et pour lesquels il faudra préciser la taille et le placement dans la topologie de la machine virtuelle (Fortran 90, HPF [Marquet, 1997], Idole [Kokoszko, 1996]).

Avantages. On pourrait alors se demander pourquoi le modèle de programmation à parallélisme de données persiste encore, malgré ses déboires (abandon des machines SIMD généralistes, difficultés à compiler sur des machines MIMD). Outre des facteurs historiques (existence de codes scientifiques basés sur des matrices), on peut avancer plusieurs explications.

Le premier facteur déterminant est sa simplicité. Le nombre de constructions est faible [Bougé et Levaire, 1992]. De plus, on passe aisément d'un raisonnement sur des scalaires à un raisonnement sur des tableaux. Suivant la «généricité» du C++ — principe dont le raisonnement humain, friand d'analogies, s'accommode assez bien —, un algorithme défini pour des scalaires peut être appliqué sans trop de difficulté à des matrices. Et l'usage de ces «objets» plus complexes masque, comme autant de traitements internes, la répétition du travail sur chacune des composantes de la matrice. En un sens, le modèle parallèle simplifie l'écriture de l'algorithme. On dit qu'il offre plus d'expressivité.

Un autre facteur est celui de la preuve de programme. Encore à l'état de recherche, il se pourrait bien qu'il devienne déterminant. La simplicité des constructions data-parallèles autorise des schémas de preuves semblables à ceux qu'il est possible de faire sur des programmes séquentiels [Bougé et Levaire, 1992] [Bougé et al., 1997]. Ceci est à opposer au modèle à passage de messages, dont la complexité conduit à multiplier le nombre d'erreurs, qui ne sont pas toujours simples à détecter, ni à corriger, par la suite.

Enfin, et suivant [Bougé, 1996], on pourrait conclure sur le fait que le parallélisme de données apporte au modèle à parallélisme de contrôle, tel que CSP [Hoare, 1978], ce que le `while` du Pascal a apporté aux langages abusant des `goto`. L'auteur argumente en remarquant que les programmes Occam sont des suites parallèles de phases séquentielles (`PAR` de `SEQ`), tandis que les programmes à parallélisme de données sont des listes séquentielles de phases parallèles (`SEQ` de `PAR`). Le parallélisme de données apporte une structuration au parallélisme, qui faisait défaut au modèle à passage de messages, ce qui contribue à limiter le nombre de fautes de programmation.

Les recherches en parallélisme de données s'orientent actuellement vers la gestion de l'irrégularité des traitements et des données. Nous proposons aussi d'introduire des primitives asynchrones dans ce modèle. Nous présentons ensuite notre travail.

Irrégularité et parallélisme de données

Idéalement, le parallélisme de données n'aimerait s'occuper que de données très régulières, toutes de la même taille, et qui se répartiraient facilement sur les processeurs de la machine. Mais si le modèle ne veut pas être cantonné à des applications trop simples, et aborder des problèmes aussi irréguliers que le traitement d'images, l'optimisation combinatoire ou l'algorithmique de graphes, il doit savoir gérer l'irrégularité.

Le terme *irrégularité*, en l'absence d'autres qualificatifs, n'a pas toujours la même signification dans la littérature. On distingue essentiellement l'irrégularité de la forme des données, qui nécessite des dispositifs adaptés pour son codage, l'irrégularité dans l'évolution des données au cours de l'exécution de l'algorithme, et l'irrégularité des traitements à appliquer aux données.

Codage de données irrégulières. Selon [Dekeyser et Marquet, 1996], les langages data-parallèles ont suivi une évolution semblable à celle suivie par les langages séquentiels. Au départ les langages séquentiels se contentaient de structures régulières, laissant au programmeur la charge de la gestion de la mémoire lorsqu'il désirait des données irrégulières. Puis les pointeurs ont autorisé la gestion dynamique de la mémoire, ce qui a facilité le travail sur des données irrégulières. Ainsi, après s'être contenté de données exclusivement régulières, les langages data-parallèles ont proposé des solutions pour coder des données irrégulières. Par exemple, en traitement d'images, il s'agira de coder l'irrégularité d'une région au contour quelconque, tel qu'un avion gris sur fond de ciel clair.

Les langages data-parallèles à machine virtuelle d'instanciation ont proposé des *collections* sur des machines virtuelles régulières (C$\star$), ou des machines virtuelles irrégulières (Parallaxis, Hyper C). Les langages à machine virtuelle d'alignement ont proposé des objets parallèles dynamiques (en anglais *DPO*) [Dekeyser et al., 1994] ou des tableaux dynamiques, comme dans Fortran 90, qui sont placés dynamiquement sur les processeurs par des directives dans HPF.

Enfin, d'autres solutions plus générales ont été proposées, telles que les langages HPF 2 et Vienna Fortran [Marquet, 1997], les extensions pour matrices creuses [Ujaldon et al., 1997], l'utilisation d'arbres [Brégier, 1998], les `shape` dans Idole [Dekeyser et al., 1996], les objets fragmentés dans ACADA [Verbièse et al., 1998], etc.

Évolution irrégulière des données. L'évolution irrégulière des données apparaît dans l'exécution d'un algorithme lorsqu'une partie des données est plus «active» que les autres, et engendre plus de nouvelles structures que les autres. Le ou les processeurs gérant ces données «actives» vont donc avoir un surcroît de travail et la performance finale sera loin de l'estimation théorique de l'accélération de l'application parallélisée. Cette irrégularité est dite *dynamique* lorsqu'elle n'est pas connue avant d'exécuter l'algorithme.

Par exemple, dans une segmentation d'images, les processeurs ayant obtenu la région du ciel auront moins de travail que ceux qui ont obtenu les régions du paysage, car ces dernières devront être scindées récursivement en de plus nombreuses et plus petites sous-régions, jusqu'à satisfaire les critères d'homogénéité définis dans l'algorithme. À moins d'analyser l'image, ce qui revient à exécuter l'algorithme, il n'est pas possible de prévoir une découpe répartissant mieux les futures créations des sous-régions.

On pourrait alors définir un programme aux évolutions irrégulières comme étant un programme qui génère du parallélisme au cours de l'exécution, en fonction des données manipulées [Roucairol, 1996]. En fait, comme cela a été noté dans [Gautier et al., 1995], à une irrégularité de l'évolution des données correspond une irrégularité des traitements, puisque les processeurs héritant des régions «actives» (paysage) auront une charge supérieure aux autres.

Traitements irréguliers. Les programmes à parallélisme de données qui ont des traitements irréguliers sont ceux dont les données ne requièrent pas toutes le même traitement : certaines données demanderont peu de travail, tandis que d'autres en nécessiteront beaucoup plus. Cette irrégularité entraîne des charges non-homogènes sur les processeurs de la machine. On pourrait la mesurer en fonction de l'influence des données traitées sur le temps d'exécution de l'algorithme. Par exemple, l'exploration parallèle d'un arbre dans une stratégie glouton d'optimisation combinatoire (*Branch and Bound*) donne des traitements inéquitables d'une branche à l'autre de l'arbre [Denneulin et Mautor, 1997]. Comme nous l'avons dit plus haut, l'irrégularité de l'évolution des données est un cas particulier de l'irrégularité des traitements.

L'irrégularité d'un algorithme est liée à la complexité de son *ordonnancement* [Roucairol, 1996]. On représente les traitements nécessaires à l'algorithme par un graphe orienté acyclique, appelé *graphe de précédence* ; l'arc (a, b) signifie que la tâche a doit être réalisée avant la tâche b ; l'ordonnancement consiste à placer ces tâches sur les processeurs [Galilée et Roch, 1997]. Le graphe de précédence peut être complètement prévisible, semi-prévisible ou imprévisible. Lorsqu'il est prévisible, l'algorithme est régulier car les traitements ne dépendent pas des données.

Lorsque le graphe de précédence est semi-prévisible, on peut structurer l'algorithme en phases et organiser des redistributions des données, ou, de façon duale, des migrations des tâches d'un processeur à un autre. La gestion de l'irrégularité peut donc être inclue dans l'algorithme lui-même : celui-ci équilibrera la charge avant de lancer une nouvelle phase.

Enfin, lorsque le graphe est imprévisible (cas du *Branch and Bound*), il peut être nécessaire d'utiliser un ordonnanceur dynamique, tel que ceux proposés dans Cilk, Athapascan-1 ou Jade [Galilée et Roch, 1997]. Il faut généralement utiliser des heuristiques pour savoir quels sont les moments les plus opportuns pour effectuer un réordonnancement [Roucairol, 1996].

Asynchronisme et parallélisme de données

Asynchronisme. Avant de parler d'asynchronisme, il nous faut tout d'abord préciser les différents sens de ce terme, utilisé dans de multiples domaines, à l'origine par opposition au mot *synchrone* (qui, étymologiquement, signifie «avec le temps»).

En électronique, un circuit asynchrone utilise un protocole sans horloge pour prévenir qu'une donnée envoyée est disponible [Mohammadi, 1996]. Nous parlerons d'*asynchronisme électronique*. En systèmes distribués, un *système asynchrone* est composé d'entités dont les exécutions ne sont pas organisées en phases globales (*round*) [Tel, 1991]. Ces phases nécessitent en effet la synchronisation des entités pour assurer la simultanéité de leurs exécutions. Dans un tel système asynchrone, on distingue les communications synchrones des *communications asynchrones*. Une communication est asynchrone lorsque l'émetteur n'est pas bloqué dans l'attente de la lecture de son message par le destinataire [Tel, 1991].

Lorsqu'aucune couche n'est construite au dessus d'un système asynchrone pour masquer l'asynchronisme, et offrir à l'utilisateur un comportement totalement synchrone, l'asynchronisme du système n'est plus un artefact de mise en œuvre, mais transparaît dans son utilisation. Suivant [Privat et al., 1995], nous parlerons dans ce cas d'*asynchronisme fonctionnel*. Ces systèmes sont plus simples à concevoir, et plus efficaces, mais plus difficiles à utiliser.

Enfin, un *programme asynchrone* est un programme parallèle qui n'utilise pas de synchronisation [Üresin et Dubois, 1990]. Lorsque le programme a été conçu de façon synchrone, pour être «implémenté» (mis en œuvre) de façon asynchrone, nous parlerons de *mise en œuvre asynchrone* ;

il y a «traduction» du programme synchrone en une suite d'opérations asynchrones. Lorsque le programme est conçu directement de façon asynchrone, nous parlerons de *conception asynchrone*.

Dé-synchronisation des traitements data-parallèles. Le modèle à parallélisme de données sous-entend, par des instructions définies sur des ensembles de données, que le même traitement est réalisé sur chacune des données de l'ensemble. Les instructions data-parallèles se traduisent par des traitements élémentaires sur les données ; elles définissent des phases de calcul durant lesquelles ces traitements locaux ont lieu.

Généralement, le travail à effectuer sur chaque donnée est simple, et pour appliquer plusieurs traitements sur les mêmes données, on programme plusieurs instructions data-parallèles. De ce fait, il n'y a qu'une instruction élémentaire par donnée au cours d'une phase définie par les instructions data-parallèles. Ces traitements locaux sont donc synchrones.

Le modèle data-parallèle incite donc à concevoir des algorithmes synchrones, qui se traduiront par des programmes parallèles synchrones, qu'il sera difficile de mettre en œuvre sur un système présentant un asynchronisme fonctionnel.

Si maintenant les traitements appliqués à chaque donnée d'un ensemble sont plus conséquents, et consistent en une suite de sous-traitements très élémentaires, la synchronisation des sous-traitements pourrait être remise en cause d'une donnée à une autre. En fait, grossir le traitement élémentaire subi par chaque donnée n'a d'intérêt que pour la remise en cause de la synchronisation de ces traitements : les traitements élémentaires restent globalement synchronisés par les instructions data-parallèles, mais les sous-traitements élémentaires peuvent être, eux, désynchronisés d'une donnée à une autre. En d'autres termes, on grossit le calcul élémentaire pour pouvoir considérer ses sous-parties et les désynchroniser.

Avantages recherchés. Sur le plan de la programmation, la dé-synchronisation des traitements locaux permet une mise en œuvre asynchrone des programmes data-parallèles (à condition que la traduction ne change pas leur comportement). Les études concernant les itérations asynchrones ont montré combien les synchronisations des programmes parallèles bridaient leurs performances. Les programmes qui permettent de lever cette contrainte obtiennent une forte *accélération* (rapport du temps de calcul séquentiel sur le temps de calcul parallèle) [Üresin et Dubois, 1989] [El Baz et al., 1996].

De plus, la dé-synchronisation autorise aussi la conception asynchrone de programmes, ce qui permet de tirer parti de l'asynchronisme des traitements locaux dans l'algorithmique.

Enfin, la programmation dans un environnement présentant un asynchronisme fonctionnel n'est pas simple ; elle hérite des difficultés que nous avons évoquées pour la programmation par passage de messages. Comme nous l'avons mentionné, le modèle data-parallèle offre une structure qui manque à la programmation par passage de messages. Mais sa mise en œuvre sur un tel système est délicate, et parfois inefficace. Un modèle data-parallèle présentant des primitives de calcul «à gros grain», et dont les sous-traitements sont asynchrones, est une solution intermédiaire entre le tout asynchrone non structuré et le tout synchrone très structuré. On peut espérer qu'un tel compromis conserve les avantages des deux extrêmes sans leurs inconvénients, c'est-à-dire qu'il soit assez structuré et assez efficace pour intéresser le programmeur.

Le modèle des Réseaux Associatifs

Origines. Le modèle des réseaux associatifs est un modèle de calcul né d'une réflexion sur l'évolution des machines de traitement d'images et des modèles d'exécution associés [Mérigot, 1992] [Mérigot, 1997a]. Dans un souci de performance, sa définition a été faite en étroite relation avec une étude architecturale[(c)]. Aussi, le modèle des réseaux associatifs, tel que défini en 1992, est assez

(c) Notre étude est, bien sûr, influencée par cette vision des modèles de programmation. Bien que nous définirons un modèle abstrait, l'implémentation n'en sera jamais très éloignée. Si cette absence de séparation radicale entre modèle d'exécution et de programmation peut apparaître gênante, il nous semble cependant qu'elle est nécessaire pour obtenir, au proche du prototype architectural réalisé en 1996 et appelé *Maille Associative* [Dulac, 1996].

Le modèle est défini à partir d'une topologie de base quelconque, telle que, par exemple, la *maille* — une grille orientée symétrique à deux dimensions, torique, et avec les connexions diagonales — qui est la topologie de la Maille Associative. Il exploite le parallélisme de données. Le codage de l'irrégularité est assuré par la définition de sous-graphes du graphe de base, obtenus en masquant des arcs. Ces sous-graphes sont codés efficacement au niveau de chaque nœud ; ils sont manipulés par des variables parallèles.

Le modèle repose sur l'`association`, une primitive de calcul data-parallèle originale. Elle s'applique sur un ensemble de données réparties sur les processeurs (une donnée par processeur), et structuré à l'aide d'un sous-graphe quelconque. Le résultat de l'`association` sur un processeur donné u est égal à l'application d'un opérateur sur l'ensemble des données des processeurs v, tels qu'il y a un chemin de u à v dans le sous-graphe. La Maille Associative met en œuvre cette primitive de façon asynchrone.

Objectifs. Le modèle des réseaux associatifs est un modèle de calcul de haut niveau, offrant un parallélisme de données et une gestion des objets irréguliers. Sa primitive travaille sur des ensembles, structurés par des graphes. Le modèle a été essentiellement utilisé en traitement d'images, pour les phases de bas et de moyen niveau (pré-traitement, isolement d'objets), et toujours en relation avec la Maille Associative. Le but de notre travail est d'étudier les propriétés intrinsèques du modèle théorique, ses applications en calcul parallèle, et sa mise en œuvre.

Parti pris. Cette étude s'inscrit dans la recherche d'un modèle de programmation offrant au programmeur une abstraction, des schémas de calcul et des primitives facilitant la programmation, sans gâcher les performances. Le modèle à passage de messages ne peut, à notre avis, répondre à cette double problématique d'efficacité et de facilité ; il est, selon l'expression de [Calvin et Colombet, 1994], l'«assembleur du parallélisme», utilisé faute de mieux en parallélisation automatique ou parallélisme de données.

Les réseaux associatifs s'apparentent à un modèle à parallélisme de données utilisant une machine virtuelle d'instanciation (le graphe de base). Nous verrons que cela n'est pas contradictoire avec la gestion des objets irréguliers quelconques. L'irrégularité des traitements, quant à elle, peut être gérée par l'introduction d'un parallélisme de contrôle dynamique, même sur réseau de stations. Nous préconisons cependant de cacher ce parallélisme de contrôle à l'utilisateur. Nous proposons une parallélisation des primitives data-parallèles à l'aide de processus légers PM^2 [Namyst, 1997], qui se répartissent les données en respectant leur irrégularité. Cette solution souple autorisera un ajustement dynamique (en cours de programme) de la distribution en fonction de la charge des machines, des performances du réseau, ou des nécessités de l'algorithme.

Mais la parallélisation des primitives data-parallèles souffre du grain faible de ce type de calcul, qui pénalise les exécutions distantes, sur réseau de stations au moins (acheminement des données et retour du résultat). Ceci est d'autant plus vrai que la dynamique des données est forte : les objets irréguliers changeant de forme rapidement, leur distribution sur les machines du réseau doit constamment être revue. Le traitement d'images, et notamment la segmentation de Voronoï, en est un exemple typique. Nous proposons de grossir le grain du calcul data-parallèle, en introduisant de nouveaux opérateurs. Ceux-ci permettent de remplacer des itérations data-parallèles synchrones par un seul calcul data-parallèle asynchrone. L'asynchronisme, d'ordinaire hors propos en parallélisme de données, peut en fait jouer un rôle salvateur, à la fois dans l'écriture des algorithmes, dans leur complexité et dans leur parallélisation. Plutôt que de provenir d'une traduction d'un programme synchrone dans un modèle d'exécution asynchrone, l'asynchronisme peut ainsi être introduit délibérément, dès la programmation, sous réserve de quelques garde-fous, prévenant des problèmes rencontrés en passage de messages.

---

final, des performances intéressantes.

Connexions. L'étude des calculs asynchrones nous a amenés à utiliser — ou adapter — des méthodes définies dans d'autres contextes. Ainsi, la recherche des sous-graphes utilisables dans la Maille Associative nous a conduit à étudier les plongements dans le graphe maille. Nous montrons que tout graphe peut être plongé dans la maille, sous des hypothèses en accord avec le modèle.

De même, la recherche d'opérateurs assurant la terminaison des calculs asynchrones a été un problème central de notre travail. Nous avons envisagé cette étude sous un angle macroscopique, à l'aide de matrices, mais aussi sous un angle microscopique, en étudiant le comportement d'un nœud, à la manière des systèmes distribués. Nous montrons que la primitive de calcul du modèle peut être appliquée dans un contexte distribué. La modélisation matricielle est généralement limitée aux systèmes synchrones. En utilisant les itérations asynchrones — des méthodes provenant de l'optimisation des calculs matriciels parallèles —, nous indiquons comment un calcul matriciel peut modéliser un système au comportement asynchrone.

Les opérateurs habituellement utilisés dans les calculs asynchrones sont associatifs, commutatifs et idempotents (nous les appelons s-opérateurs, par commodité). Ces opérateurs exploitent peu l'information codée par les relations inter-données, modélisées par un sous-graphe dans le modèle. En effet, seule l'information relative à la connexité est exploitée. Nous proposons des propriétés plus larges que celles définissant les s-opérateurs, en déformant l'algèbre habituelle par une application r. Nous obtenons ainsi les r-opérateurs, aux propriétés algébriques intéressantes, et aux applications algorithmiques plus importantes (exploitation de la distance ou des chemins plutôt que de la connexité). En outre, ces opérateurs ont d'intéressantes propriétés de résistance aux pannes, que nous étudions sous l'angle des systèmes distribués : nous montrons que la primitive de calcul asynchrone est auto-stabilisante.

Cependant, mis sous une forme matricielle, les calculs utilisant des r-opérateurs ne sont pas sans rappeler l'algèbre de chemins — une des applications de l'algèbre max-plus —, bien que le calcul soit, ici, asynchrone et défini au niveau du nœud. Nous adaptons des algorithmes matriciels existants, et nous définissons des algorithmes de réseau originaux, qui peuvent, en retour, être adaptés à l'algèbre de chemins matricielle. Nous proposons aussi une version matricielle de la preuve de l'auto-stabilisation, grâce à l'algèbre max-plus et aux itérations asynchrones.

Ainsi, l'étude du modèle quelque peu hybride des réseaux associatifs nous a conduits à sortir du cadre du calcul parallèle ; notre travail souligne des connexions entre l'algèbre max-plus matricielle et le calcul asynchrone sur réseau, grâce aux itérations asynchrones et aux r-opérateurs.

Plan du document

Définition du modèle. La première partie est consacrée aux problèmes de définition du modèle. Le chapitre 1 présente le cadre de l'étude. Nous résumons le projet des réseaux associatifs, en rappelant son contexte historique et de recherche. Nous recadrons ensuite notre travail dans ce projet.

Le chapitre 2 illustre l'utilisation du modèle en traitement d'images, avec l'exemple de la segmentation de Voronoï. Nous retrouverons cet algorithme-test tout au long de notre étude.

Dans le chapitre 3, nous justifions la refonte du modèle en étudiant les différences entre sa définition et son implémentation dans la Maille Associative. Nous présentons alors une nouvelle définition, essentiellement basée sur une nouvelle primitive, la `direct-association`, qui intègre l'asynchronisme de la Maille dans la définition même du modèle. Nous étudions ensuite la construction de l'ancienne primitive (`association`), à l'aide de la `direct-association`. Nous clôturons ce chapitre — et la première partie — en présentant la nouvelle problématique qui se dégage de la refonte du modèle. Il s'agit essentiellement d'étudier l'intérêt de la `direct-association`, alors que cette primitive n'est définie que sous certaines conditions sur les opérateurs utilisés.

Étude du modèle. La seconde partie est consacrée à l'étude du modèle. Nous commençons par présenter au chapitre 4 diverses comparaisons entre notre «objet informatique» et d'autres modélisations voisines. L'algèbre max-plus permet une modélisation macroscopique des aspects calculatoires du réseau, sous réserve de synchronisation des nœuds. Les itérations asynchrones lèvent la contrainte de synchronisation. Nous notons cependant quelques limites à la méthode matricielle : elle est inadaptée à la conception de certains algorithmes, et l'asynchronisme n'intervient qu'après la conception, comme une optimisation de l'exécution. Les systèmes distribués étudient les problèmes de résistance aux pannes et de terminaison. Bien que la vocation première du modèle soit le calcul parallèle, cette comparaison indique une autre application du modèle.

Nous développons au chapitre 5 l'algèbre déformée des r-opérateurs. Ces opérateurs assurent la terminaison des calculs asynchrones dans un réseau ; ils élargissent les applications de la `direct-association`.

Au chapitre 6, nous étudions le contrôle du réseau : propriété des liens, détection de terminaison et résistance aux pannes (auto-stabilisation) de la `direct-association` utilisant des r-opérateurs. L'étude du graphe maille est résumée en annexe C. Nous montrons que cette topologie donne au modèle un graphe de base intéressant, autorisant tout type d'objets irréguliers.

Application et mise en œuvre du modèle. La troisième partie de ce document est consacrée aux applications et à la mise en œuvre du modèle. Nous développons au chapitre 7 divers algorithmes dans les domaines des graphes, de la géométrie et du traitement d'images. Nous montrons comment une seule `direct-association` permet de calculer une distance, les polygones de Voronoï ou un arbre couvrant en profondeur. Nous expliquons aussi comment ces algorithmes peuvent être adaptés en algèbre de chemins et en systèmes distribués, et nous proposons une méthodologie de programmation asynchrone.

La mise en œuvre du modèle est étudiée au chapitre 8. Nous présentons la bibliothèque de programmation parallèle ANET (pour *Associative NET*), et les tests de celle-ci sur réseau TCP et Myrinet de stations de travail. ANET permet d'écrire des programmes C++ parallèles, admettant des structures de données irrégulières et les primitives des réseaux associatifs. ANET est basée sur PM^2, et peut s'exécuter sur un grand nombre de plates-formes.

Nous présentons le bilan de notre travail, et des idées avancées, en conclusion, où nous indiquons des pistes de recherche future.

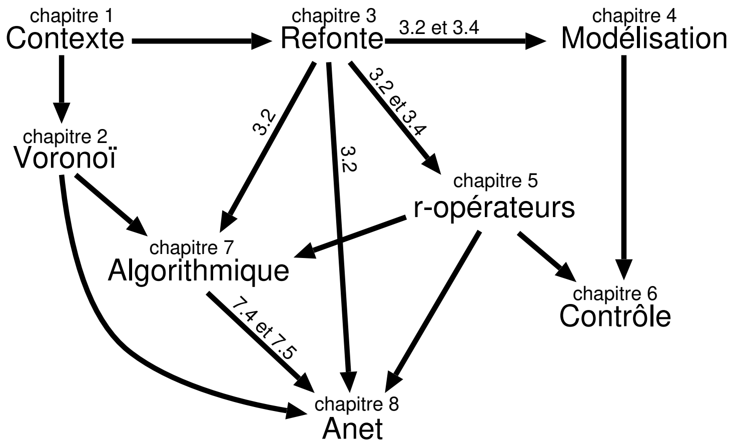


FIG. 1 – *Principales dépendances directes entre chapitres.*

Contributions et rappels. Ce document propose plusieurs rappels, dans divers domaines. Les sections concernées sont balisées avec une astérisque dans le titre. Nous précisons maintenant les sections qui incluent des rappels et celles qui ont donné lieu à une communication.

Le chapitre **1** propose plusieurs rappels : traitement d'images, machines spécialisées et modèles de calcul qui mixent calculs et communications. Les travaux menés dans le cadre du projet des réseaux associatifs y sont aussi récapitulés.

Le chapitre **2** commence par une présentation de la segmentation d'images. L'adaptation de la segmentation de Voronoï aux réseaux associatifs y est ensuite expliqué. Ce travail a été publié dans

[Ducourthial et al., 1998].

Le chapitre **3** présente le travail de redéfinition du modèle ; il reprend en partie les rapports [Ducourthial, 1997] et [Ducourthial, 1998b].

Le chapitre **4** propose une courte synthèse de l'algèbre max-plus, des itérations asynchrones, et des systèmes distribués. Dans ce chapitre, notre contribution porte surtout sur le rapprochement de ces disciplines, et leur utilisation dans l'étude des réseaux associatifs, aux § 4.1.8, 4.2.4 et § 4.3.3. Cette dernière section reprend une partie de la publication [Ducourthial et Tixeuil, 1998a].

Le chapitre **5** présente l'étude de nouveaux opérateurs. Ce travail a fait l'objet des rapports [Ducourthial, 1997] [Ducourthial, 1998b], et de la publication [Ducourthial, 1998c].

Le chapitre **6** présente l'étude des propriétés du réseau. Il est basé sur les rapports [Ducourthial, 1998b] [Ducourthial et Tixeuil, 1998b], et sur les publications [Ducourthial, 1998a] [Ducourthial et Tixeuil, 1998a] [Ducourthial, 1998d].

Dans le chapitre **7**, on trouve des algorithmes originaux et des adaptations d'algorithmes existants pour les réseaux associatifs. Certains algorithmes se trouvent dans les rapports [Ducourthial, 1998b] [Ducourthial et Tixeuil, 1998b] et [Ducourthial et Mérigot, 1998].

Le travail de conception et de tests de la bibliothèque ANET est rapporté dans le chapitre **8**. Ce chapitre contient une courte présentation des environnements *multithreads*, et de leur utilisation pour le parallélisme de données irrégulier. Ce travail a été récapitulé dans le rapport [Ducourthial et Sicard, 1998].

L'annexe **A** présente des fragments de code et des résultats du programme de la segmentation de Voronoï, présentée aux chapitres 2 et 7. L'annexe **B** donne les courbes commentées au chapitre 8. Enfin, l'annexe **C** résume l'étude du graphe maille, rédigée dans le rapport [Ducourthial et Mérigot, 1996].

Première partie

Définition du Modèle

Chapitre 1

Contexte de l'Étude

À l'origine, le modèle des réseaux associatifs a été défini pour des applications de traitement d'images, en vue de réaliser une architecture spécialisée. Son principe est inspiré d'une réflexion portant à la fois sur ce type de machines, et sur les modèles de calcul parallèle.

Dans ce chapitre, nous introduisons le contexte à l'origine du projet de recherche des réseaux associatifs. Nous présentons ensuite le modèle tel que défini en 1992, ainsi que son implémentation dans la Maille Associative. Enfin, nous proposons un bilan des apports de ce projet de recherche. Nous terminons en replaçant notre travail dans ce projet.

1.1 Origines du projet des réseaux associatifs

Dans cette section, nous présentons l'analyse d'images, les machines associées, et les modèles de calcul, dont s'inspirent les réseaux associatifs.

1.1.1 Analyse d'images*

Nous basons notre présentation sur [Pavlidis, 1982] [Jolion, 1998].

Historique. Le traitement d'images remonte aux années 1950, où l'on s'est intéressé à l'analyse des images prises dans les chambres à bulles. De là sont nées la restauration et l'amélioration des images, puis la compression en vue du stockage.

Dans les années 1970, on a cherché à extraire automatiquement des informations dans les images. Les premiers travaux sur le seuillage, la segmentation, l'extraction de contours et la morphologie mathématique datent de cette décennie. Quelques tentatives ont été faites pour interpréter automatiquement les images. Dans les années 1980, le mouvement a été pris en compte dans l'analyse des images (séquences), notamment pour des applications de robotique. Les recherches se poursuivent actuellement notamment dans le codage et la transmission des images sur les réseaux.

Image. Une *image physique* est obtenue à partir d'une scène réelle à un instant t par une mesure instantanée, qui est généralement la luminance dans une longueur d'onde donnée. Cette mesure n'est pas stockée de façon continue, mais sous forme d'un échantillonage discret à deux dimensions : on obtient des *pixels*. Les pixels[1] sont les éléments unitaires de l'échantillonage. Lorsque c'est la luminance qui a été mesurée, le pixel représente un *niveau de gris*. Une image physique est le tableau à deux dimensions des pixels.

1. Le terme *pixel* vient de l'anglais *picture element*.

Le pixel représente donc une quantité d'information très restreinte ; il n'a de sens qu'accompagné de son contexte géographique dans l'image. Une image représentant une icône (taille du timbre poste) a généralement 32×32 pixels. Les images couramment traitées sont de taille 256×256 pixels. Les images satellites sont de taille 1024×1024 pixels, voire plus.

Généralement, les niveaux de gris sont quantifiés sur 256 niveaux — c'est-à-dire en 8 bits ; le blanc est codé par 255 et le noir par 0 —, ce qui excède les capacités de l'œil humain. Par contre, les capteurs courants offrent au mieux des images de taille 512×512, ce qui peut se révéler insuffisant pour rendre les détails de la scène observée (bruit dû à la faiblesse de l'échantillonage).

Analyse d'images. L'analyse d'images consiste à passer d'une représentation physique d'une image à une information symbolique (objets), en vue d'une interprétation sémantique ultérieure : avion donc danger, arbre donc bord de la route, etc. L'information symbolique n'est pas toujours liée aux objets de l'image ; il peut s'agir d'informations sur la texture, permettant de reconnaître les zones cultivées des zones boisées par exemple.

La qualité de l'image physique initiale intervient beaucoup dans le résultat de l'analyse. Or les dispositifs physiques qui ont permis d'obtenir l'image (capteurs, caméra, objectifs, etc.), les conditions de l'acquisition (vibrations, etc.), et la faiblesse de l'échantillonage (les petits détails apparaissent comme des points, effet *poivre et sel*) ont introduit des parasites, que l'on appelle *bruit*, et qui sont des pixels qui détonnent par rapport à leur entourage. Ces pixels vont gêner la reconnaissance des objets, à commencer par leurs contours.

La figure 1 représente les différentes étapes d'une chaîne de traitement d'une image dont l'objectif est la conduite d'un véhicule. D'autres chaînes peuvent être rencontrées suivant l'objectif recherché (compression puis envoi de l'image, par exemple). Pour s'affranchir du bruit, on pratique tout d'abord un pré-traitement de l'image, en appliquant des opérations dites de *bas niveau* : moyenne pondérée des pixels voisins, filtres, égalisations d'histogrammes (**phase 1**). Cette phase nécessite beaucoup de communications locales (entre pixels proches). L'isolement des objets de l'image est réalisé par des opérations de *niveau intermédiaire* (**phase 2**). On utilise des *segmentations* d'images, dont le but est de définir des zones de l'image qui répondent à des *critères d'homogénéité* (différence entre le maximum des niveaux de gris et le minimum, niveau moyen de gris, etc.), de taille (diamètre de la zone, nombre de pixels, etc.) ou de forme (zones convexes par exemple). La recherche de ces zones est un travail coûteux, surtout quand plusieurs critères de recherche sont combinés (*cf.* § 2.1.1 page 19). La troisième phase consiste à comparer les objets à une base de données, afin de leur donner un sens sémantique (**phase 3**). Puis, si le dispositif fait partie d'un système auto-asservi (voiture pilotée automatiquement par exemple), il faut interpréter la scène et fusionner les informations obtenues avec celles en provenance d'autres capteurs (autres longueurs d'ondes, radar télémétrique, son, etc.), et prendre une décision (**phase 4**). La décision est ensuite traduite en ordre de commande.

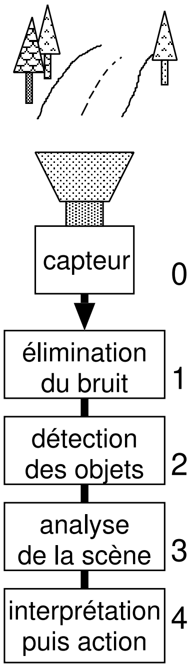


FIG. 1 – Exemple de chaîne de traitement d'images.

Problématique. Le projet des réseaux associatifs concerne les deux premières phases (bas et moyen niveau) du traitement d'images. Ces phases se distinguent par les caractéristiques suivantes :

- une masse de données à traiter importante,
- une relation géographique forte entre les pixels de l'image (voisinage),
- une organisation structurelle des objets dans l'image complexe, irrégulière et imprévisible, parce que dépendant de l'image elle-même (proximité des arbres dans la scène de la figure 1 par exemple),
- des traitements qui sont tour à tour
  - simples, locaux et répétitifs (traitements sur les voisinages des pixels, surtout au cours de la première phase)
  - complexes et globaux, c'est-à-dire mettant en jeux des ensembles de données distantes les unes des autres (travail sur les objets de l'image, tels que les arbres de la figure 1), surtout au cours de la seconde phase.

La quantité de calcul a orienté les recherches vers une solution parallèle. Les débouchés dans le domaine du traitement d'images ont contribué à l'étude d'architectures parallèles spécifiques. L'organisation des pixels dans une image a conduit aux architectures en tableaux de processeurs.

1.1.2 Machines pour le traitement d'images*

Nous présentons un panorama des architectures utilisées pour le traitement d'images. Il permet de mieux comprendre l'intérêt du projet des réseaux associatifs. Les données rassemblées ici sont tirées de [Li et Stout, 1991], [Charot, 1993], [de Rumeur, 1994], [Sansonnet, 1994] et [Dulac, 1996] ; de plus amples informations pourront être trouvées dans ces références.

Différents types. Plusieurs pistes ont été étudiées pour les machines de traitement d'images, les perpétuelles améliorations de la technologie donnant sans cesse de nouvelles idées aux architectes.

Les études les plus nombreuses ont porté sur les machines SIMD[2]. Cependant diverses machines MIMD[3] et systoliques[4] ont été réalisées pour le traitement d'images, comme l'indiquent les figures 2 et 3. Notons qu'un certain nombre de réalisations ont aussi été faites à base de DSP[5].

2. SIMD : *Single Instruction Multiple Data*, terme de la classification de Flynn. Machines ayant un seul flot de contrôle et plusieurs flots de données.

3. MIMD : *Multiple Instruction Multiple Data*. Terme de la classification de Flynn. Machines ayant plusieurs flots de contrôle et plusieurs flots de données.

4. Machine systolique : machine exploitant le parallélisme de flux. Aussi appelées machines *pipeline*.

5. DSP : *Digital Signal Processing*, processeur spécialisé dans le traitement du signal.

Topologie	Nom	Caractéristiques
Multi-étage	PASM	Université de Purdue (USA), 1981. Mode multi-SIMD et mode MIMD. Ensemble de processeurs, de mémoire et de micro-contrôleurs assurant les différents modes.
Hypercube	Ncube/ten	Ncube. VLSI spécifique, compact. Jusqu'à 1024 nœuds dans une seule armoire (un seul fond de panier). Flottants. Mémoire réduite.
	iPSC	Intel. D'après la licence du *Cosmic Cube* du CalTech (USA), mais *hardware* différent. Jusqu'à 128 nœuds. Flottants. Forte latence du réseau.
Grille	VICTOR	IBM. À base de transputers. Flottants
	iWARP	Université Carnegie-Mellon / Intel, 1988. iWARP est une version intégré du processeur de WARP (*cf.* figure 3), instructions VLIW. Commercialisation par Intel : d'une grille 2×16 processeurs sur 8 cartes Sun (640 MFlops, million d'opérations flottantes par seconde) jusqu'à une grille de 1024 processeurs (4 armoires).

FIG. 2 – *Machines de traitement d'images de type MIMD.*

Topologie	Nom	Caractéristiques
Linéaire	Cytocomputer	Université du Michigan, USA, 1981. Balayage ligne de l'image. Bit-série. Fonctions logiques de voisinage 3×3.
	PIPE	Université du Wisconsin (USA), 1985. Opérations arithmétiques, logiques et de voisinage (spatial et temporel). Chemin de données 8 bits.
	WARP	Université Carnegie-Mellon (USA), 1986. Commercialisée par General Electric, 1987. Réseau linéaire de 10 processeurs. 100 MFlops.

FIG. 3 – *Machines de traitement d'images de type systolique.*

Les architectures SIMD linéaires sont, dans certains cas, assez proches de la philosophie systolique, car l'image est balayée ligne par ligne, et la machine traite les lignes les unes après les autres. Dans d'autres cas, la ligne de processeurs, plus simple à construire que le tableau, émule une grille. La figure 4 page suivante présente quelques architectures SIMD linéaires.

Évolution des architectures en tableau SIMD. Les premières machines pour le traitement d'images sont de type tableaux de processeurs SIMD. Leur évolution reflète de façon intéressante la problématique du domaine.

Les débuts de ces architectures remontent à l'Illiac IV (NASA), même si cette machine n'était pas destinée au traitement d'images. Il s'agit d'une grille à deux dimensions de 64 processeurs synchrones, à communications vers les quatre plus proches voisins. Dans les années soixante-dix, on trouve notamment la famille des CLIP, et dans les années quatre-vingt la machine MPP (NASA / Goodyear, 1982). La figure 5 page 7 présente les principales réalisations.

La topologie en grille est très naturelle pour le traitement d'images. Elle est adaptée aux traitements de bas-niveau, qui nécessitent beaucoup de calculs répétitifs sur les voisinages des pixels.

Nom	Caractéristiques
CLIP 7-A	University College of London, 1987. 256 processeurs de 16 bits à mémoire externe, avec 512 circuits CLIP 7. Émulation d'une grille 256×256.
AIS-5000	Applied Intelligent Systems, 1987. 1024 processeurs bit-série à mémoire de 32 kb. Opérations booléennes, arithmétiques et de voisinage.
SLAP	Université Carnegie-Mellon (USA), 1988. 16 bits. Prototype de 512 processeurs sur 3 cartes de type Sun.
BLITZEN	Université de Caroline du Nord (USA), 1989. Évolution de MPP (*cf.* figure § 5 page ci-contre) : mémoire, adressage, exécution conditionnelle... 8×16 processeurs bit-série.
Princeton	Centre de recherche SARNOFF, 1989. Jusqu'à 2048 processeurs à mémoire de 16 k mots de 16 bits. Communication par diffusion ou par chemin dédié, 64 processeurs atteignables dans le cycle.
Sympati 2	CEA, Saclay / IRIT, Toulouse, 1990. Anneau de 32 à 128 processeurs 16 bits à mémoire de 32 ko. 32 processeurs par carte PC. Distribution des données hélicoïdale pour accès au voisinage 3×3 en un cycle. Commercialisation en 1993 (CENTRALP, OPENVISION).
GFLOPS	CEA, Saclay / IRIT, Toulouse, 1991. *General Flexible Linearly Organized Parallel Structure.* Contrôle SIMD et MIMD, selon configuration.
RVS-1	NEC, 1993. Émule une grille 512×512 avec 64 circuits IMAP de 8 processeurs 8 bits et 144 kb de mémoire chacun. 12,8 Gips (giga-instruction par seconde).
RVS-2	NEC, 1995. Les nouveaux circuits IMAP contiennent 64 processeurs et 2 Mb de mémoire statique. 30 Gips.

FIG. 4 – *Machines SIMD linéaires pour le traitement d'images.*

Cependant les traitements de niveaux intermédiaires nécessitent de manipuler des objets de l'image, et requièrent des communications entre pixels éloignés. Dans ce cas, la grille $n \times n$, dont le diamètre[6] est $2n - 2$ en 4-connexité[7] [(a)] et $n - 1$ en 8-connexité[8], devient un handicap.

6. Diamètre : maximum des distances entre deux sommets.

7. Grille en 4-connexité : grille à deux dimensions avec les connexions verticales et horizontales (Nord, Est, Ouest, Sud, *NEWS* en anglais).

8. Grille en 8-connexité : grille ayant les connexions horizontales, verticales et diagonales (Nord-Est, Sud-Est, Sud-Ouest et Nord-Ouest).

9. En anglais, *global or*.

Les architectes ont ajouté des fonctionnalités aux grilles, telles que des facilités de communications dans AAP, des «autoroutes» dans DAP, ou un *crossbar* dans la MASPAR (*cf.* figure 5 page ci-contre), et des possibilités de calculs globaux, comme par exemple le «ou global»[9] câblé, les additionneurs sur le côté de la grille dans CLIP 4, et le mécanisme à histogramme dans GRID. Mais ces ajouts relèvent plutôt de l'astuce que du modèle de calcul, et les recherches se sont orientées selon deux pistes. La première concerne les réseaux à faible diamètre tels que les hypercubes et la seconde concerne les pyramides (figure 6 page 8).

Topologie en pyramide (figure 6 page 8)**.** La pyramide permet d'appliquer des opérations globales assez rapidement, avec la philosophie «diviser pour régner». Elle se justifie surtout pour les traitements multi-échelles, *i.e.*, appliqués sur l'image avec différentes résolutions. En outre, les communications à longue distance sont en $O(\log(n))$ pour une base carrée de côté n. Cependant, les pyramides sont complexes à réaliser, et leur contrôle est délicat, notamment pour les pyramide multi-SIMD (chaque étage est une grille SIMD). Finalement, la complexité de ces architectures, leur coût et leur important volume matériel, comparés aux contraintes des applications de traitement d'images et à l'étroitesse du marché concerné, a limité leur développement à un contexte universitaire.

Topologie en hypercube (figure 6 page 8)**.** Les topologies en hypercube ont surtout été mises en œuvre au travers des Connection Machines. Ces topologies ont l'avantage à la fois des pyramides et des grilles, mais elles sont difficiles à réaliser. Par exemple, pour atteindre 256×256 processeurs (une image), la CM 200 nécessitait un volume de 2 m^3. En outre, elles ne peuvent pas suivre les progrès de l'intégration des circuits intégrés. En effet, s'il est possible d'intégrer plus de processeurs dans le circuit qu'auparavant, les possibilités d'interconnexion des éléments restent, elles, limitées (*cf.* § 8.1.1). Or tout accroissement du nombre de sommets de l'hypercube implique une augmentation

(a) Les termes «4-connexe» et «8-connexe» viennent de la terminologie du traitement d'images et des architectures spécialisées. En théorie des graphes, on utilise les termes de *grille simple* (sans diagonale) et *grille étendue* (avec diagonales). Notons que ces grilles sont, respectivement, 4-connexes et 8-connexes lorsqu'elles sont toriques : 4 arêtes (respectivement 8) doivent être enlevées pour déconnecter le graphe.

Nom	Caractéristiques
CLIP 1	University College of London, 1974. Grille de 100 processeurs.
CLIP 2	University College of London. Grille hexagonale.
DAP	International Computers Ltd (UK), 1976. 64×64 processeurs 1 bit. Mémoire de 4 kb par processeur. Communications longue distance avec les *Highways*, des bus sur les lignes ou les colonnes
GRID	GEC Research Laboratory, 1980. Grille 4-connexité de 64×64 processeurs. Mémoire à deux port. Communications aux voisins diagonaux en un cycle. Bus selon les lignes et les colonnes (*X*- et *Y-buses*). Dispositif pour histogramme.
CLIP 4	University College of London, 1980. 96×96 processeurs booléens bit-série. 8-connexité, mémoire de 32 mots de 8 bits. Localement asynchrone.
MPP	NASA / Goodyear, 1982. 128×128 processeurs bit-série. Mémoire (externe) de 1 kb par processeur. Supporte les pannes grâce à 4 colonnes de processeurs supplémentaires. Entrées-sorties optimisées. Ou global cablé.
AAP	Nippon Telegraph Telephon, 1983. *Adaptative Array Processor.* Grille à 2 dimensions en 8-connexité. Reconfigurable. Processeurs transmetteurs et récepteurs. Possibilité de calculs globaux avec transfert asynchrone.
GAPP	NCR, 1987. *Geometric Arithmetic Parallel Processor.* Circuit de 6×12 processeurs, bit-série. Opérations logiques, addition et soustraction. Mémoire de 128 bits. Ou global cablé.
MASIVE	École Nationale Supérieure de Strasbourg. Une des nombreuses machines à base de circuits GAPP. Contrôle optimisé.
MASPAR	1990. Grille de 128×128. Mémoire de 16 kb. Arithmétique entière et flottante (en 64 bits). Réseau *X-net* : planaire, connexions au niveau des diagonales, émule une grille 8-connexe. Routage global avec seulement 3 nœuds intermédiaires, par des crossbars.

FIG. 5 – *Machines SIMD en grille pour le traitement d'images.*

du degré au niveau de chaque nœud. Finalement, le coût et l'encombrement de ces machines les a cantonnées dans des centres de calcul.

Architectures reconfigurables (figure 8 page suivante)**.** Ces considérations ont conduit aux architectures reconfigurables. Il s'agit d'architectures simples, spécialisées dans la simulation d'une autre architecture, plus adaptée au traitement d'images. Par exemple, une ligne de processeurs peut simuler une grille, et un tableau une pyramide (*cf.* figure 8 page suivante). Ainsi, dans la machine PAPIA 2, en forme de grille, la pyramide du projet PAPIA 1 est simulée en autant de phases que d'étages (figure 7) : lors de la première phase, tous les processeurs de la grille émulent la base de la pyramide, lors de la seconde phase, un processeur sur quatre émule le second étage, etc.

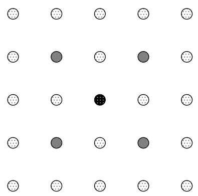


FIG. 7 – Simulation d'une pyramide dans une grille (plus un processeur est foncé, plus l'étage qu'il simule est haut dans la pyramide).

1.1.3 Modèles de calcul*

Rapide tour d'horizon. Plutôt que de présenter un panorama des nombreux modèles de calcul parallèles (*cf.* page ix), nous proposons un rapide classement, basé sur le critère de l'intégration des calculs et des communications, qui intéresse plus particulièrement notre travail. Nous renvoyons par exemple à [Maggs et al., 1995] pour un classement thématique plus complet.

Un *programme* est une suite de calculs sur des données. Il est *parallèle* lorsque plusieurs calculs ont lieu simultanément. Le programme parallèle se décompose en *calculs locaux* qui utilisent des données locales au nœud de calcul, et en *calculs globaux* qui utilisent des données réparties sur plusieurs nœuds de calcul. Un calcul global nécessite donc une *communication* entre plusieurs processeurs (réalisée par envoi de message, mémoire partagée, signaux, etc.). Lorsque la densité de communication est forte par rapport aux calculs, on parle de parallélisme à *grain fin*. A l'inverse, lorsqu'il y a peu de communications par rapport aux calculs, on parle de parallélisme à *gros grain*. En général, l'intégration des calculs aux communications se retrouve plutôt dans les modèles à grain fin.

Le modèle PRAM[10] à accès exclusif en lecture et écriture (EREW[11]) «sérialise» les communications vers le même point mémoire. Le modèle CREW[12] lève la contrainte des lectures exclusives,

10. *Parallel RAM* (*Parallel Random Access Memory*) : mémoire commune à plusieurs processeurs.

11. *Exclusive Read, Exclusive Write.*

12. *Concurrent Read, Exclusive Write.*

13. *Concurrent Read, Concurrent Write.*

Topologie	**Nom**	**Caractéristiques**
Hypercube	CM 1	MIT (USA), 1983. Commercialisation par Thinking Machine Corporation. Réseau hypercube et réseau en grille à deux dimension, en 4-connexité. Bit-série.
	CM 2	Thinking Machine Corporation, 1988. Jusqu'à 65536 PEs (processeurs élémentaires). Processeur bit-série. Mémoire de 256 kb par processeur. Un nœud de l'hypercube contient 16 processeurs complètement connectés. Coprocesseur arithmétique pour 32 processeurs. Entrées / sorties optimisées.
	CM 200	Thinking Machine Corporation. Optimisation de la virtualisation des processeurs.
Pyramide	HCLM	Université de Seattle, 1987. À base de processeurs PCLIP (1983) : *Pyramidal Cellular LogIc Processor*. Grilles en 8-connexité. Pyramide quaternaire. Filtres travaillant avec les valeurs 0, 1 et D (*Don't care*).
	GAM	Pyramide quaternaire de 5 étages, à base de processeurs de MPP (*cf.* figure 5 page précédente). Arbre d'additionneur sur le premier niveau. Ou global câblé au niveau de chaque étage. Contrôleur pour les connexions inter-étages.
	GAM 2	1988. Version améliorée avec 6 étages et 3 contrôleurs d'étages.
	PAPIA 1	Université de Pavie, Rome, Milan et Palerme (Italie), 1985. Pyramide quaternaire, étages en grille 2D 4-connexes.
	SPHINX	IEF Orsay / ETCA Arcueil 1991. Étages SIMD en forme de grille à deux dimensions, mais chaque processeur est le sommet d'un arbre binaire (pyramide binaire). Contrôle multi-SIMD. Bit-série. Mémoire de 256 bits par processeur. Réalisation de 16 processeurs.

FIG. 6 – *Machines SIMD d'autres topologies pour le traitement d'images.*

Topologie	**Nom**	**Caractéristiques**
Grille	PAPIA 2	Grille 512×512 reconfigurable, permettant d'émuler une pyramide. Circuits de 16×8 processeurs.
Tore	Tore polymorphe	Étude, 1989. Réseau physique PNET : tore à deux dimensions en 4-connexité. Réseau INET programmable sur chaque nœud (architecture reconfigurable).
	YUPPIE	IBM, 1991. Mise en œuvre du tore polymorphe. 4×4 processeurs.
	PPA	1993. *Polymorphic Processor Array*. Mise en œuvre du tore polymorphe.

FIG. 8 – *Machines SIMD reconfigurables pour le traitement d'images.*

tandis que le modèle CRCW[13] autorise aussi les écritures simultanées. Les conflits d'accès en écriture au même emplacement mémoire et au même instant sont gérés en utilisant le hasard, des priorités ou une combinaison des données arrivées au même endroit. On trouve ainsi les *sends* avec collision, qui construisent une seule donnée avec un opérateur associatif et commutatif, et permettent de n'avoir qu'une seule donnée à stocker en mémoire.

Le modèle PRAM présente l'avantage indéniable de masquer l'architecture de la machine sous-jacente, et de permettre un développement algorithmique indépendant de la machine. Toutefois, cela se fait au prix d'hypothèses très fortes sur les mécanismes de communications, et un réseau de communications donné ne sera pas exploité au maximum de ses performances. Notre objectif est l'étude d'un mécanisme de calcul permettant, sur un réseau donné, de réaliser conjointement calculs et communications. Aussi ne détaillerons-nous pas plus avant le modèle PRAM.

A l'opposé de ce modèle très général, nous trouvons des modèles de programmation dont les primitives reposent sur des échanges de données suivant les liens de communication d'un réseau donné. À de rares exceptions près — correspondant à des algorithmes particulièrement réguliers et adaptés à la topologie —, il est difficile de vraiment tirer parti d'un tel modèle. Aussi, des primitives de plus haut niveau, utilisant implicitement les caractéristiques du réseau de communications, ont été proposées. Elles permettent de réaliser rapidement des calculs globaux mettant en jeu un ensemble de données réparties. Ces primitives de calcul exploitent toutefois la régularité du réseau et les ensembles de processeurs concernés doivent avoir des caractéristiques bien définies (ensemble complet des processeurs, processeurs répartis suivant des «axes» privilégiés de la topologie, *etc.*). Nous trouvons ainsi principalement les β-réductions, qui permettent d'obtenir une seule valeur à partir d'un ensemble, en utilisant un opérateur associatif et commutatif. De même, les *scans* (ou calculs préfixes) mixent calculs et communications en une seule opération, en répartissant les résultats d'un calcul selon une chaîne de processeurs. Nous détaillons maintenant ces modèles. Leur philosophie est proche de celle des réseaux associatifs : calculs et communications ne sont plus discernables et on parle de *réseau calculant*.

Modélisation du problème du préfixe. Appelons $\mathbb{S}$ l'ensemble des données, et considérons l'opérateur unaire $f\ :\mathbb{S}\to\mathbb{S}$. Dans un contexte SIMD à n processeurs, un calcul local modifie les n données simultanément :

$$\text{calcul local par } f : \quad \begin{array}{ccc} \mathbb{S}^n & \to & \mathbb{S}^n \\ X=\{x_1,\dots,x_n\} & \mapsto & f(X)=\{f(x_1),\dots,f(x_n)\} \end{array} \tag{1.1}$$

Les opérations globales sont généralement construites à l'aide d'opérateurs binaires (addition par exemple). Considérons la loi de composition interne $\diamond$. La modélisation du *problème du préfixe* du calcul global est la suivante[(b)] :

$$\text{calcul global par } \diamond : \quad \begin{array}{ccc} A^n & \to & A^n \\ X=\{x_1,\dots,x_n\} & \mapsto & \{x_1, x_1\diamond x_2,\dots,x_1\diamond\cdots\diamond x_n\} \end{array} \tag{1.2}$$

En d'autres termes, il s'agit d'obtenir toutes les «sommes» intermédiaires jusqu'à la somme de tous les termes. Le problème du préfixe est une modélisation des calculs parallèles parmi d'autres. On trouve dans [Ferreira et Fraigniaud, 1997] une adaptation d'un calcul préfixe au modèle théorique PRAM, aux modèles à grain fin, et aux modèles à gros grain.

Historique du problème du préfixe. Les origines du problème du préfixe tiennent à la fois de la programmation et de la construction de circuits électroniques. Étant donné n entrées, il s'agit de résoudre le problème du préfixe à l'aide de portes logiques à deux entrées, assemblées en un seul circuit (*cf.* figure 9). La *taille du circuit* correspond au nombre de portes et mesure le coût en matériel. La *profondeur du circuit* correspond au nombre d'étages dans le circuit et modélise le temps nécessaire au calcul. Ces deux paramètres ne sont pas indépendants et l'optimisation de l'un dégrade la valeur de l'autre.

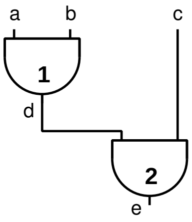


FIG. 9 – Circuit préfixe à trois entrées (taille 2, profondeur 2).

Dans l'article [Ladner et Fischer, 1980], qui semble fondateur dans le domaine, les auteurs formalisent le problème et obtiennent une construction récursive pour obtenir un circuit de profondeur exactement $\lceil \log_2(n) \rceil$ avec une taille bornée par $4n$, avec n entrées. Dans [Snir, 1986], l'auteur montre que la somme de la profondeur et de la taille du circuit est supérieure ou égale à $2n-2$; il donne aussi un certain nombre de circuits optimaux.

Les propriétés du circuit optimal dépendent en fait de la structure algébrique sous-jacente. La loi de composition $\diamond$ est, par convention, associative à gauche[14]. Pour autoriser la diversité des circuits, et donc les optimisations du calcul, il faut pouvoir effectuer des calculs intermédiaires en ordre quelconque. On suppose donc $\diamond$ associative[15] et commutative[16]. On obtient alors un monoïde abélien[(c)] $(\mathbb{S},\diamond)$. Plusieurs travaux ont montré et étudié les relations entre les dimensions du circuit et la structure de monoïde sous-jacente. En fait, ces paramètres apparaissent comme un moyen supplémentaire de classement des monoïdes. Par exemple, dans [Bilardi et Preparata, 1990], les auteurs caractérisent les monoïdes en fonction de leur aptitude à autoriser un circuit de profondeur constante (donc temps

14. Associativité à gauche : convention d'écriture pour les opérateurs binaires qui suppose que les expressions sont parenthésées par la gauche : $(a\diamond b)\diamond c \equiv a\diamond b\diamond c$.

15. Associativité de $\diamond$: $\forall x,y,z \in \mathbb{S}$, $(x\diamond y)\diamond z = x\diamond(y\diamond z)$.

16. Commutativité : $\forall x,y\in\mathbb{S}, x\diamond y = y\diamond x$.

(b) Dans [Blelloch, 1990a], le problème du préfixe est appelé *problème de toutes les sommes préfixes*. Dans [Leighton, 1992], il est appelé *Prefix computation* et dans [Ferreira et Fraigniaud, 1997], le problème est appelé *sommes partielles préfixées*. Dans [Lakshmivarahan et Dhall, 1994], les auteurs parlent de *prefix problem*.

(c) Voir la présentation des structures algébriques de la section 4.1 page 51, et en particulier la note **(b)** page 54 sur la terminologie.

de calcul constant) et de taille linéaire en n, le nombre d'entrées, qui peut être quelconque. Cette propriété est liée à l'existence d'un *circuit monoïdal* associé au monoïde dans le graphe de Cayley (figure 10).

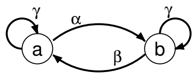


FIG. 10 – Cycle monoïdal $(a, b), (b, a)$.

Pour illustrer la relation entre la structure algébrique $(\mathbb{S}, \diamond)$ et le temps de calcul moyen du problème du préfixe sur $\mathbb{S}$, on peut considérer les ensembles $L_\diamond = \{a \in \mathbb{S}, \exists c \in \mathbb{S}, \forall b \in \mathbb{S}, a \diamond b = c\}$ et $R_\diamond = \{b \in \mathbb{S}, \exists c \in \mathbb{S}, \forall a \in \mathbb{S}, a \diamond b = c\}$ [Jakoby et al., 1995]. Ces sous-ensembles de $\mathbb{S}$ contiennent les éléments qui donnent toujours une constante lorsqu'ils sont additionnés à gauche — ou à droite — à un élément du monoïde $\mathbb{S}$. La date d_2 de fin de calcul de la porte 2 du circuit de la figure 9 page précédente est, en général, liée à la date de fin de calcul d_1 de la première porte. Mais lorsque c est un élément de $R_\diamond$, alors la deuxième porte n'attend plus la première. L'accélération moyenne du circuit préfixe sur le monoïde $\mathbb{S}$ est liée à la taille des ensembles $L_\diamond$ et $R_\diamond$ par rapport à celle de $\mathbb{S}$. On parle de monoïde ayant une *accélération à gauche* ou à *droite*.

17. Accélération : rapport du temps de calcul séquentiel sur le temps de calcul parallèle.

Dans [Jakoby et al., 1995], les auteurs étudient les accélérations[17] moyennes des monoïdes en reliant cette notion à celle d'accessibilité des automates dont le graphe est le Cayley associé au monoïde : partant d'un sommet du graphe (état de l'automate, élément du monoïde) peut-on atteindre un puits, c'est-à-dire une zone de l'automate de laquelle on ne sort plus ? Cette intéressante étude permet de conclure sur l'intérêt d'un circuit préfixe pour le calcul des fonctions logiques binaires or et and (le puits est atteint en une étape), pour le seuillage binaire de p bits parmi[18] n (le puits est atteint en p étapes), etc. En guise d'application pratique, on peut construire un circuit réalisant l'addition de mots binaires de n bits qui présente une accélération moyenne exponentielle, tandis qu'il n'en existe pas pour la fonction parité[19].

18. Seuillage binaire : cette fonction vaut 1 s'il y a au moins p 1 dans le mot binaire de n bits.

19. Parité binaire : cette fonction vaut 1 s'il y a un nombre pair de 1 dans le mot binaire, et 0 sinon.

La modélisation préfixe a aussi des applications dans les récurrences linéaires, le filtrage de signaux numériques, le tri, l'interpolation polynomiale, etc. Le livre [Lakshmivarahan et Dhall, 1994] présente une synthèse du sujet. Mais ce modèle s'est surtout popularisé en tant que modèle de programmation, à la suite des travaux de Hillis et Blelloch notamment [Hillis, 1985] [Blelloch, 1990b]. Ces travaux ont été poursuivis pour implémenter efficacement le modèle sur diverses architectures [Cohn, 1990] [Chung, 1995]. Nous détaillons dans les paragraphes suivants les aspects modèle de programmation du calcul préfixe.

Modèle du CM-Lisp. Hillis a proposé dans le modèle du CM-Lisp une formalisation des calculs parallèles proche du problème du préfixe [Hillis, 1985]. Les opérations locales sont notées à l'aide des α-extensions. Une α-extension appliquée à une expression scalaire donne une expression «vectorielle», c'est-à-dire s'appliquant à des vecteurs[(d)]. Ainsi, l'incrémentation de toutes les données du vecteur $\texttt{X} = \langle 1, 3, 5, 4, 2\rangle$ est notée $\alpha + \ 1 \ \cdot\texttt{X}$ et donnera le vecteur $\langle 2, 4, 6, 5, 3\rangle$ (le point $\cdot$ devant `X` signifie que le vecteur `X` ne doit pas être «étendu», comme l'est le scalaire 1, et l'addition scalaire $+$).

Ainsi, l'opérateur α transforme un scalaire en vecteur : $\alpha 1 = \langle 1, \ldots, 1\rangle$. L'opération inverse, de réduction d'un vecteur en un scalaire, est appelée β-réduction. Par exemple, $\beta +\ \texttt{X}$ donne le scalaire $1 + 3 + 5 + 4 + 2 = 15$. $\beta \diamond\ \texttt{X}$ donne donc la dernière somme du problème du préfixe instancié par $\diamond$. Mais on ne peut pas obtenir les sommes intermédiaires. Le modèle propose aussi une β-réduction binaire dans laquelle on précise aussi un vecteur d'indices. Mais cette primitive ne permet pas non plus d'obtenir toutes les sommes intermédiaires. Or le problème du préfixe est très riche d'applications [Lakshmivarahan et Dhall, 1994] [Blelloch, 1990b] [Blelloch, 1990a]. Nous avons proposé des constructions récursives à base d'α-extensions et de β-réductions pour résoudre ce problème dans [Ducourthial, 1995], mais ces résultats ne sont pas intéressants en pratique.

Le modèle du CM-Lisp n'a été qu'incomplètement mis en œuvre au travers du langage $\star$Lisp sur la Connection Machine (*cf.* figure 6 page 8).

Scan vector model. Blelloch a proposé en 1988 une amélioration, réalisée avec peu de modification, du modèle proposé par Hillis et utilisé sur la Connection Machine [Blelloch, 1990b] [Blelloch,

(d) Hillis appelle *xecteur* la structure proche du vecteur dans laquelle les données sont ordonnées par un indice.

1990a]. Son travail s'inscrit dans la suite de celui de Sabot sur le Parallation Lisp [Sabot, 1988] ; le compilateur du *scan vector model* est écrit dans ce langage, qui est un langage parallèle à collection.

La structure de données est le vecteur. Les calculs locaux sont réalisés par des opérations *elementwise* et notés p. Par exemple l'addition de deux vecteurs `X` et `Y` sera notée `X` $p+$ `Y`.

Les calculs globaux sont essentiellement réalisés par l'opérateur *scan* qui donne exactement toutes les sommes du problème du préfixe. Appliqué au vecteur $\langle 1,3,5,4,2\rangle$ précédent, le `max-scan` donne :

$$\texttt{max-scan}\langle 1,3,5,4,2\rangle = \langle 1,3,5,5,5\rangle$$

On définit aussi le *scan-segmenté* par l'application d'un *scan* sur chaque *segment* («sous-vecteur») d'un vecteur. Par exemple, si les segments sont représentés par un trait souligné, on a :

$$\texttt{max-scan}\langle \underline{1,3,5},\underline{4,2}\rangle = \langle \underline{1,3,5},\underline{4,4}\rangle$$

Ce modèle a pu être mis en œuvre sur diverses machines parallèles, à l'aide d'un arbre de réduction (*cf.* figures 11 et 12), y compris lorsqu'il y a plusieurs données par processeur [Blelloch, 1990a] [Leighton, 1992] (l'opérateur doit être associatif). De nombreuses applications ont été définies pour le *scan vector model*, telles que la ligne de vue, le tri *radix-sort*, le tri *quicksort*, etc. Nous donnons ci-dessous une implémentation parallèle du tri à bulle [Knuth, 1969] à l'aide de *scan*, que nous avions développée initialement pour le modèle des réseaux associatifs [Ducourthial, 1995].

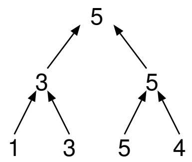


FIG. 11 – Arbre de réduction du `max-scan` (montée).

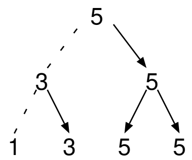


FIG. 12 – Arbre de réduction du `max-scan` (descente).

$$\begin{array}{lcl}
\texttt{max-scan}\langle 1,3,5,4,2\rangle & = & \langle 1,3,5,5,5\rangle \\
\text{segments de } \langle 1,3,5,4,2\rangle & = & \langle \underline{1},\underline{3},\underline{5,4,2}\rangle \\
\text{permutation circulaire dans les segments} & = & \langle \underline{1},\underline{3},\underline{4,2,5}\rangle \\
\texttt{max-scan}\langle 1,3,4,2,5\rangle & = & \langle 1,3,4,4,5\rangle \\
\text{segments de } \langle 1,3,4,2,5\rangle & = & \langle \underline{1},\underline{3},\underline{4,2},\underline{5}\rangle \\
\text{permutation circulaire dans les segments} & = & \langle \underline{1},\underline{3},\underline{2,4},\underline{5}\rangle \\
\texttt{max-scan}\langle 1,3,2,4,5\rangle & = & \langle 1,3,3,4,5\rangle \\
\text{segments de } \langle 1,3,2,4,5\rangle & = & \langle \underline{1},\underline{3,2},\underline{4},\underline{5}\rangle \\
\text{permutation circulaire dans les segments} & = & \langle \underline{1},\underline{2},\underline{3},\underline{4},\underline{5}\rangle
\end{array}$$

1.2 Le projet des réseaux associatifs

Au § 1.1.1 page 3, nous avons exposé la problématique du traitement d'images. Au § 1.1.2 page 5, nous avons résumé l'évolution des machines parallèles dédiées au traitement d'images. Par ailleurs, au § 1.1.3 page 7, nous avons expliqué le problème du préfixe et l'évolution des modèles de calculs associés : CM-Lisp et *scan*.

Nous présentons maintenant le projet des réseaux associatifs. Il propose une solution au problème de la programmation parallèle de haut niveau (abstraite), tout en permettant la réalisation d'une architecture performante mettant en œuvre le modèle, et satisfaisant aux contraintes du traitement d'images parallèle. Les apports du projet par rapport aux machines et modèles présentés précédemment sont résumés dans la section suivante, page 16.

1.2.1 Définition du modèle*

Nous présentons ici la définition du modèle des réseaux associatifs[(e)], donnée dans [Mérigot, 1992] [Mérigot, 1997a], à quelques nuances près.

Réseau. Soit $G_b(V_b, E_b)$ un graphe orienté symétrique, appelé *graphe de base*. Ce graphe modélise le réseau : les sommets de V_b sont les *nœuds de calcul*, qui pourront être des processeurs dans une machine, ou des éléments logiciels dans une bibliothèque de calcul. Un arc (i, j) de E_b modélise le lien de communication unidirectionnel du nœud de calcul i vers le nœud j. On note $\delta^-_{G_b}$ le maximum des degrés entrants[20] des nœuds de G_b : $\delta^-_{G_b} = \mathsf{max}_{v \in V} \delta^-_v$ (δ^-_v note le degré entrant de v).

20. Degré entrant de v : nombre d'arcs de E ayant pour extrémité finale le sommet v ; on le note δ^-_v. Les notations $d(v)$ et Δ sont utilisées, par exemple dans [de Rumeur, 1994]. Mais elles prêtent à confusion avec la distance et les opérateurs dans ce document.

Une *variable parallèle*, ou *pvar*, est un $|V|$-uplet réparti à raison d'une donnée par nœud sur le réseau. Si p est une pvar, $p[v]$ est la valeur de la pvar p sur le nœud v.

Un *sous-réseau couvrant*[(f)] $G(V, E) \subset G_b(V_b, E_b)$ est obtenu en «masquant» des connexions entrantes au niveau des nœuds dans le graphe de base ($V = V_b$ et $E \subset E_b$). Cela permet de considérer divers schémas G de communication dans le réseau. Le graphe de base G_b est déterminant dans la définition du schéma de communication global, mais aussi dans celui des sous-réseaux $G \subset G_b$ qu'il est possible d'utiliser (on utilise le terme *plonger*, *cf.* annexe C page 195).

Un sous-réseau couvrant G est manipulé par une pvar net_G de mots binaires de $\delta^-_{G_b}$ bits : au niveau de chaque nœud v, la donnée $net_G[v]$ de $\delta^-_{G_b}$ bits permet de savoir si le $i^{\text{ème}}$ arc entrant de v est masqué ou non (*cf.* figure 13). Plus précisément, pour $1 \leq i \leq \delta^-_v$, si $net_G[v][i] = 1$, l'arc n'est pas masqué ; si $net_G[v][i] = 0$, il l'est ($p[v][i]$ donne la valeur du $i^{\text{ème}}$ bit de la valeur de la pvar p en v). On appelle *masque* la valeur locale à un nœud de cette pvar spéciale. La pvar elle-même est appelée[(g)] *pvar de masques*.

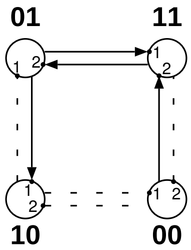


FIG. 13 – Exemple de masque (en gras), et des arcs correspondants.

Opérations. Un *calcul élémentaire* est un calcul local qui consiste à modifier une pvar avec un opérateur unaire sur tous les nœuds du réseau. Par exemple l'instruction `p += 1` ajoute 1 à la pvar entière p.

Une `association` est un calcul global qui modifie la valeur $p[v]$ d'une pvar p sur chaque nœud v, en combinant, avec un opérateur associatif et commutatif $\diamond$, toutes les composantes de la pvar situées sur les antécédents[21] de v, ainsi que sur v lui-même, dans un sous-réseau donné $G \subset G_b$ (on note $\overline{\Gamma^-_G}(v)$ cet ensemble de sommets). Plus formellement, on note ce calcul global par $\diamond$-`association`(net_G, p) et on le définit par :

21. Antécédents de v dans G : ensemble noté $\Gamma^-_G(v)$ des sommets u de V tel qu'il existe un *chemin* (suite d'arcs consécutifs) de u à v. On pose $\overline{\Gamma^-_G}(v) = \Gamma^-_G(v) \cup \{v\}$.

$$\begin{array}{llcll} \diamond - \texttt{association}(net_G, p) \; : & \{\text{pvar}\} & \rightarrow & \{\text{pvar}\} & \\ & p & \mapsto & q & \text{avec} \quad q[v] = \diamond_{u \in \overline{\Gamma^-_G}(v)} p[u] \end{array} \tag{1.3}$$

En utilisant les antécédents directs[22] de v dans G (v inclus, *i.e.*, $\overline{\Gamma^{-1}_G}(v)$) au lieu de tous les antécédents ($\overline{\Gamma^-_G}(v)$), on définit la variante de l'association appelée `step-association`. Lorsqu'en plus la valeur $p[v]$ n'est pas utilisée dans le calcul de la nouvelle valeur de la pvar sur v (*i.e.*, on utilise $\Gamma^{-1}_G(b)$), la variante est appelée `step-exclusive-association`. Bien qu'envisageables, les autres combinaisons ne sont pas définies dans le modèle pour des raisons d'implémentations.

22. Antécédent direct de v : sommets u de V tels qu'il existe un arc (u, v) dans E. L'ensemble des antécédents directs de v est noté $\Gamma^{-1}_G(v)$. On pose $\Gamma^{-1}_G(v) \cup \{v\} = \overline{\Gamma^{-1}_G}(v)$.

Lorsque le sous-réseau G n'a pas de circuit — et en pratique il a toujours été question d'arborescence dans ce cas là —, l'association est dite préfixe et on la note `prefix-association`. Il s'agit d'un qualificatif uniquement, car la définition de cette primitive est semblable à celle de l'`association`. Le terme «préfixe» vient, évidemment, du problème du préfixe (§ 1.1.3 page 7). Le résultat de l'`association` est très différent quand le sous-réseau est, par exemple, un circuit[23] au lieu d'être un simple chemin. En effet, dans le premier cas, une seule valeur est obtenue sur tous les

23. Circuit : chemin dont les extrémités initiale et finale sont confondues.

(e) Le terme «associatif» vient de la primitive `association`, qui «associe» les valeurs du réseau au cours du calcul.

(f) Sous-graphe couvrant est la traduction de *spanning subgraph*, qui est appelé *graphe partiel* en français [de Rumeur, 1994]. Pour rester cohérent avec la bibliothèque de calcul `ANET`, dans laquelle un sous-réseau couvrant est appelé `subnet`, nous utilisons les termes *sous-réseau couvrant* et *sous-graphe couvrant* (ou simplement *sous-réseau* et *sous-graphe*).

(g) Le modèle des réseaux associatifs étant généralement utilisé avec un réseau de base en forme de maille, les sous-réseaux et les pvars qui les codent sont souvent appelés *mgraphes* (m pour maille).

nœuds car les sommets du circuit sont tous antécédents les uns des autres. Ce n'est pas le cas dans le chemin (non rebouclé), où autant de valeurs différentes que de sommets peuvent être obtenues. Le qualificatif «préfixe» a donc un sens sémantique. En outre, aussi bien dans la Maille Associative (*cf.* § 1.2.2) que dans les bibliothèques de programmation des réseaux associatifs (*cf.* chapitre 8), l'algorithme de calcul de l'`association` est différent lorsqu'il n'y a pas de circuit. En particulier, il peut être optimisé dans les bibliothèques lorsqu'on est assuré de l'absence de circuit dans le sous-réseau. Le qualificatif «préfixe» a donc aussi été ajouté pour des raisons de mise en œuvre. Nous donnons un troisième sens au qualificatif «préfixe» au § 3.1.2 page 30.

Exemple. La figure 14 illustre l'application d'une max-`association` dans un sous-réseau d'un réseau de base en forme de grille 2×4. Les arcs en pointillés représentent les connexions masquées. Le sous-réseau considéré dans cet exemple comporte une composante fortement connexe[24] de plus d'un sommet. Le même résultat est, par définition, obtenu sur tous les sommets appartenant à la même composante fortement connexe.

La figure 15 illustre l'application d'une max-`association` sur un sous-réseau sans circuit (composantes fortement connexes de taille 1). On parle alors de max-`prefix-association`.

24. Composante fortement connexe (CFC) : ensemble maximal de sommets, tel qu'il existe un chemin de tout sommet vers tout autre.

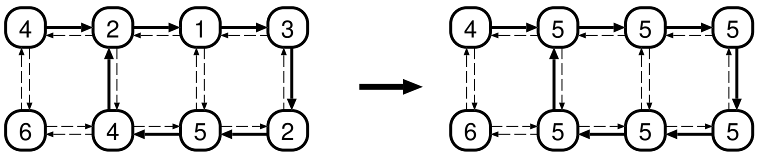


FIG. 14 – *Exemple d'une* max-`association`.

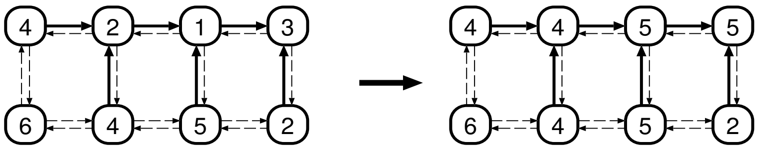


FIG. 15 – *Exemple d'une* max-`prefix-association`.

1.2.2 La Maille Associative*

Description. La Maille Associative[(h)] est une architecture électronique qui a été étudiée à l'Institut d'Électronique Fondamentale pour mettre en œuvre le modèle tel que défini précédemment [Dulac et al., 1993] [Dulac, 1996]. Il s'agit d'une machine multi-processeurs de type SIMD, dont la topologie est une *maille*, c'est-à-dire une grille orientée symétrique[25] et torique[26] à deux dimensions ayant les connexions verticales, horizontales et diagonales (*cf.* figure 17 page suivante et annexe C). La figure 16 page suivante détaille l'architecture d'un processeur élémentaire de la Maille Associative. Un circuit composé d'une maille 4×4 a été réalisé à des fins de validation expérimentale du mécanisme de calcul, et d'évaluation des temps de calcul. La technologie utilisée est du CMOS 1 μm. Un PE contient environ 5000 transistors ; il est traversé en une à deux nano-secondes, selon l'opérateur utilisé, lors d'une `association` sur des données de un bit. Une `association` nécessite un temps de l'ordre de la micro-seconde. La bibliothèque de programmation SAM[27] permet d'évaluer le temps d'exécution des algorithmes destinés à la Maille [Mérigot, 1997b].

25. Graphe symétrique $G(V, E)$ **:** $\forall(u, v) \in E$, $\exists(v, u) \in E$. Uniquement pour les graphes orientés.

26. Connexion torique dans une grille : l'arc (u, v) de direction (α, β), avec $\alpha = \pm 1$ et $\beta = \pm 1$, part du sommet u de coordonnée (x_u, y_u) dans la grille et arrive au sommet v de coordonnée $x_v = x_u + \alpha \bmod n$ et $y_v = y_u + \beta \bmod n$.

27. *Simulation of the Associative Mesh.*

Mise en œuvre d'un calcul global. Pour réaliser la max-`association`(net_G, p) de la figure 14, on procède comme suit[(i)].

(h) On appelle aussi *Maille* la machine Maille Associative. Pour ne pas confondre avec le graphe maille, le nom de la machine a une majuscule.

(i) Dans la Maille Associative, seuls le or et le xor sont câblés ; les autres fonctions logiques et l'addition sont obtenues au niveau logiciel, à l'aide d'un algorithme construit avec la or-`association` et la xor-`association`.

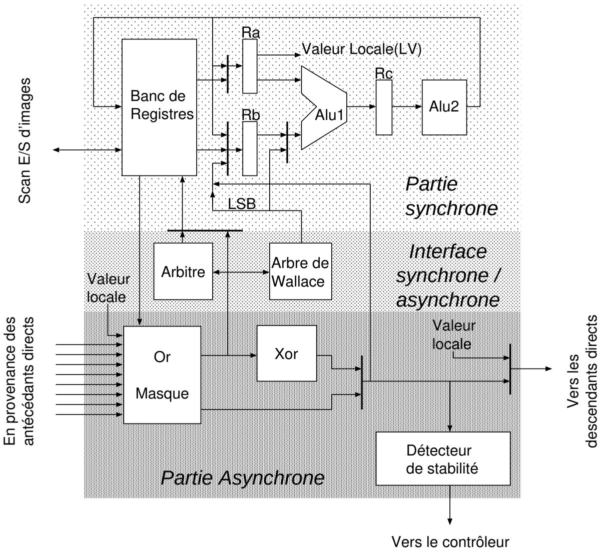


FIG. 16 – *Architecture du processeur élémentaire de la Maille Associative.*

1. le registre du masque est initialisé avec net_G sur chaque nœud v du réseau ;
2. la valeur locale de chaque nœud v du réseau est initialisée avec $p[v]$;
4. les nœuds calculent indéfiniment le maximum des valeurs des registres entrants et de la valeur locale ; le résultat est stocké dans le registre de la valeur locale et envoyé aux descendants directs ;
5. le détecteur de stabilité de chaque nœud v envoie un 0 au contrôleur quand le résultat produit est invariant pendant un délai t_{prop} correspondant au temps de traversée d'un nœud et d'un lien (*cf.* figure 18) ;
6. le contrôleur calcule un «ou global» sur les bits envoyés par les détecteurs de stabilité ; lorsque le résultat vaut 0, alors le calcul est stable partout, et le contrôleur peut donner l'ordre aux nœuds d'arrêter les calculs : la `max-association`(net_G, p) est terminée.

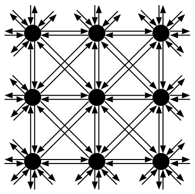


FIG. 17 – Maille 3×3 (les connexions toriques ne sont pas dessinées complètement).

Ainsi, le calcul global est réalisé par les *exécutions concurrentes et non coordonnées* des calculs locaux de chaque processeur. Dans la Maille Associative, les `associations` sont utilisées avec les opérateurs min, max, plus, and, or. Les `prefix-associations` sont utilisées sur des arborescences, avec tous les opérateurs, y compris l'addition plus. La primitive `spanning-tree` permet d'obtenir des arborescences couvrantes pour calculer, par exemple, la somme des valeurs des nœuds dans les composantes fortement connexes. Il s'agit d'une primitive câblée, qui organise la propagation asynchrone d'un jeton. Un «arbitre» (arbre de bascules) permet de ne retenir, au niveau de chaque nœud, qu'un seul antécédent parmi ceux qui ont envoyé un jeton. Enfin, la primitive `link-with-equals` permet de ne conserver au sous-réseau que les arcs reliant deux sommets ayant la même valeur [(j)].

Asynchronisme électronique. Le terme asynchronisme étant utilisé sous différents sens dans ce document, nous lui ajoutons l'épithète «électronique» et nous détaillons maintenant cette notion [(k)]. D'autres informations sur les systèmes électroniques asynchrones peuvent être trouvées dans [Dulac et al., 1995] et [Mohammadi et al., 1994].

Un système logique comprend des opérateurs combinatoires et des éléments de mémorisation. Un opérateur est constitué de portes, caractérisées par leur fonctionnalité univoque et leur délai de traversée, qui est, lui, variable en fonction des fluctuations technologiques et de la température. Puisqu'un opérateur est formé d'un ensemble de portes, le comportement temporel de ses sorties

(j) Des primitives semblables sont disponibles dans les bibliothèques SAM et ANET (*cf.* chapitre 8). Mais elles peuvent aussi être obtenues à partir des autres primitives du modèle (*cf.* chapitre 7).

(k) Cet exposé est tiré de la présentation faite dans [Ducourthial, 1995], elle-même construite à partir du séminaire *Circuits et architectures asynchrones : état de l'art et perspectives* présenté par A. Mérigot à l'IEF en 1995.

ne peut en général être prévu de façon précise. Seul l'état des sorties *au bout d'un temps fini* est prévisible.

Les systèmes synchrones utilisent une horloge commune pour que tout opérateur sache à quel moment il peut utiliser les données présentes sur ses entrées. La période de cette horloge est fixée par le temps nécessaire à la traversée du plus long des opérateurs combinatoires du système, lorsque ce dernier est utilisé dans les pires conditions. Il y a donc un nivellement par le bas : le plus «mauvais» des opérateurs et les pires conditions fixent la règle d'utilisation générale.

Dans les systèmes asynchrones, il n'y a pas de référentiel global de temps. C'est la disponibilité des données qui commande localement leur mémorisation. Il n'y a donc plus de nivellement par le bas. Il existe différentes solutions pour prévenir un opérateur que les données présentes sur ses entrées sont valides ; citons les codes autosynchronisants (le «double rail») ou les protocoles dits de «poignées de main» [Mohammadi, 1996].

Gestion de l'asynchronisme dans la Maille. Les diverses solutions pour prévenir de la disponibilité d'une donnée évoquées ci-dessus ont été jugées trop contraignantes pour la Maille Associative : elles nécessiteraient des opérateurs logiques supplémentaires pour le contrôle et les entrées / sorties, et surtout des voies de communication inter-processeurs plus nombreuses, ce qui prendrait trop de place dans le circuit intégré [(I)]. C'est donc un mécanisme plus simple qui est utilisé : on vérifie la stabilité d'un certain signal sur l'ensemble de la machine. Un détecteur local repère la stabilité du signal pendant un temps t supérieur au temps t_{prop} de propagation du signal à travers un processeur. Cette information est ensuite centralisée vers le contrôleur de la machine (figure 18). La stabilisation du signal peut, certes, arriver avant le temps t, et, dans ce cas, on attend plus longtemps inutilement. Mais cette sécurité ne rentre en jeu qu'à la fin de l'`association`, et non à chaque traversée d'un opérateur, comme dans les systèmes synchrones.

On dit que le calcul est *terminé* lorsque le contrôleur a centralisé des informations partout concordantes sur la stabilisation locale des valeurs produites par les nœuds. On dit aussi, de façon équivalente, que le réseau s'est *stabilisé*.

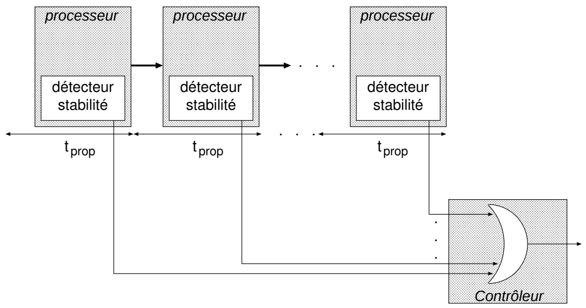


FIG. 18 – *Détection de la terminaison dans la Maille Associative. Les détecteurs envoient 1 tant qu'ils détectent une instabilité locale. Lorsque le contrôleur obtient 0 en sortie de la porte* or *pendant un certain délai, il peut être sûr que le calcul est globalement stable.*

L'asynchronisme dans la Maille Associative permet des temps de calcul particulièrement rapides, et cela sans surcoût notoire en logique par rapport aux machines classiques SIMD en maille. Quelques microsecondes suffisent pour appliquer un opérateur dans une machine de taille 256×256. Les mêmes opérations prennent un temps de 100 à 1000 fois supérieur sur des architectures classiques.

(I) En outre, les protocoles de poignées de main ne sont utilisables que dans un sous-réseau sans circuit.

1.3 Apports du projet des réseaux associatifs

Après avoir présenté le modèle des réseaux associatifs au § 1.2.1 page 12 et son implémentation au travers de la Maille Associative au § 1.2.2 page 13, nous expliquons maintenant les principales contributions du projet des réseaux associatifs dans ces domaines.

1.3.1 Apport de l'architecture*

La Maille Associative apporte une justification au modèle des réseaux associatifs : ce modèle n'est pas uniquement théorique car il peut être mis en œuvre efficacement. L'originalité de la machine réside dans sa gestion des communications et des calculs sur des zones irrégulières à l'aide des masques, et par la mise en œuvre asynchrone des primitives.

Reconfigurabilité par les masques. Les masques permettent de définir tout type de régions dans la pvar image répartie sur les processeurs de la Maille Associative. Ainsi, on peut coder des objets, ou leurs contours, des arbres de diffusion, etc. Divers schémas de configurations irréguliers peuvent ainsi être positionnés, uniquement en affectant une variable parallèle, ce qui est très rapide (*cf.* figures 116, 117, 118 et 119 page 162). Dans la lignée de AAP, puis de PAPIA 2, du Tore Polymorphe et de ses implémentations Yuppie et PPA (*cf.* figure 8 page 8), la Maille Associative propose donc une implémentation innovante du concept de reconfigurabilité, concept qui semble maintenant incontournable dans les machines en tableaux de processeurs SIMD dédiées au traitement d'images [Li et Stout, 1991]. Ce mécanisme de reconfiguration s'adapte naturellement aux contraintes des applications d'analyse d'images.

Chemin de données asynchrone. Le calcul d'une valeur à partir des données réparties sur une grande zone de l'image est une opération courante en algorithmique d'images. À l'instar de CLIP 4 (figure 5 page 7), la mise en œuvre asynchrone proposée dans la Maille Associative prouve que l'asynchronisme est très intéressant pour obtenir de bonnes performances dans ce type de calcul. Elle permet notamment de contourner la nécessaire synchronisation globale des processeurs, qui est le principal facteur limitant des architectures SIMD.

Asynchronisme fonctionnel. L'asynchronisme de la Maille est plus important qu'une simple technique de communication sans horloge. En effet, il influence la programmation de la machine.

S'il est possible de transmettre n'importe quelle donnée de façon asynchrone, il n'est pas possible de réaliser tout type de calcul de façon asynchrone en attendant sa terminaison : pour certains opérateurs, sur certains réseaux, le calcul ne s'arrête pas, ce qui le rend indéfini. Sans étude fine des caractéristiques de l'opérateur ou du réseau, il est impossible de prédire l'éventuelle terminaison du calcul, ni la date à laquelle elle apparaît, le cas échéant. La terminaison est donc détectée parce qu'elle ne peut être prévue. On appelle cet asynchronisme du calcul *asynchronisme fonctionnel* [Privat et al., 1995].

1.3.2 Apports du modèle*

Les apports du modèle sont de trois ordres. Il code l'irrégularité des données de façon originale avec les masques ; il est entièrement basé sur le concept de réseau calculant et il généralise le problème du préfixe. Nous revenons au chapitre 8 sur les données irrégulières, manipulées par des variables parallèles codant des sous-réseaux . L'intérêt algorithmique de cette technique est illustré au chapitre 7. Nous détaillons ici les deux derniers points.

Réseau calculant. Le modèle fait sien le principe du mélange des communications et des calculs. Nous appelons ce concept *réseau calculant*.

Dans le modèle des réseaux associatifs, il n'est pas possible de distinguer les calculs des communications : une communication «pure» est réalisée (par exemple) par une `max-association` sur le sous-réseau approprié, en plaçant 0 (l'élément neutre de l'opérateur max) sur les nœuds qui n'émettent pas. La donnée du nœud émetteur est ainsi diffusée sur le sous-réseau. Si plusieurs nœuds émettent, les données émises sont combinées à l'aide de l'opérateur (ici max), ce qui revient à un *send avec collision*.

Un réseau calculant est un réseau de communication qui réalise des calculs pendant les communications. Ce procédé élimine un certain nombre des problèmes liés aux communications globales dans les réseaux. Par exemple, lorsque deux données différentes se disputent un canal de communication, une seule donnée, résultant de la combinaison par un opérateur des deux protagonistes, est envoyée sur le canal, et la contention disparaît (voir aussi proposition 77 page 105).

De même, pour calculer un résultat avec toutes les données des processeurs, la première solution consiste à envoyer toutes les données sur le même processeur. Mais dans ce cas, des contentions lors de l'arrivée des données conduisent à une perte d'efficacité, indépendante de la capacité du processeur. On parle de *goulet d'étranglement*. La seconde solution consiste à organiser, au niveau logiciel, la répartition des calculs intermédiaires. Suivant les cas, et surtout lorsque les schémas de communications sont irréguliers, ce travail peut ne pas être simple. Dans un réseau calculant, cette organisation des calculs intermédiaires est inhérente aux communications. En outre, elle conduit à un parallélisme presque maximal.

Ce concept de réseau calculant peut, en un sens, trouver ses origines dans le réseau du *New-York Ultra Computer* développé par A. Gottlieb au Courant Institut de l'Université de New-York [Sansonnet, 1994]. Pour diminuer les goulets d'étranglement dans les accès mémoire (concernant essentiellement les variables de synchronisation), cette machine propose l'instruction `Fetch-and-Add(`a, b`)`, qui retourne la valeur à l'adresse a, et la modifie ensuite en mémoire en lui ajoutant b : $*a += b$. Les nœuds du réseau oméga de la machine (*cf.* figure 19) concentrent les requêtes vers les mêmes emplacements mémoire (concernant les lectures / modification de la même variable) en une seule requête, diminuant ainsi les goulets d'étranglements lors de l'accès à la mémoire.

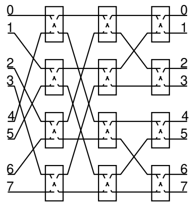


FIG. 19 – Réseau oméga à 8 entrées.

Problème du préfixe généralisé. Enfin, le modèle des réseaux associatifs généralise le problème du préfixe, le modèle du CM-Lisp et celui des *scan*. En effet, un `max-scan` sur une ligne de processeurs n'est autre qu'une `max-prefix-association` sur un sous-réseau en forme de chemin dans le graphe de base (le qualificatif `prefix` vient de là). Si le graphe de base possède un chemin hamiltonien[28] (ce qui peut être un critère dans le choix du graphe de base), un `scan` sur toutes les données sera mis en œuvre avec une `prefix-association` sur le sous-réseau qui a la forme du chemin hamiltonien (figure 20).

28. Chemin hamiltonien : chemin qui passe une et une seule fois par tous les sommets du graphe.

Mais il est aussi possible d'appliquer une `prefix-association` sur une arborescence[(m)], ce qui généralise la ligne des *scans* et du problème du préfixe. En outre, une `association` sur un sous-réseau fortement connexe fournit un seul résultat sur tous les sommets, ce qui correspond à une β-réduction, suivie d'une α-extension pour diffuser le résultat à tous les nœuds. Notons que, si on applique une `prefix-association` sur une arborescence couvrante de la composante fortement connexe, orientée vers la racine, on obtient exactement la β-réduction (le résultat obtenu sur la racine n'est pas diffusé).

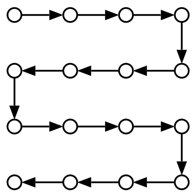


FIG. 20 – Chemin hamiltonien dans la grille, permettant de réaliser un `scan`.

Enfin, l'α-extension des opérateurs est implicite dans le modèle des réseaux associatifs (on note, par exemple, $p = p' + p''$, avec p, p' et p'' des pvars). Il peut en être de même pour les données (on note $p = 1$ pour initialiser partout la pvar p à 1), mais l'α-extension des données peut aussi être explicite. En effet, la diffusion d'une valeur $p[i]$ du processeur i vers tous ses descendants dans le sous-réseau G peut être réalisée par une `max-association(`p_G, p`)`, en initialisant préalablement $p[j]$ à 0, l'élément neutre de max, pour $j \neq i$.

(m) Nous expliquons au § 3.1.2 page 30 que les arborescences ne sont pas les seuls graphes qui permettent de généraliser le problème du préfixe et nous définissons les *graphes préfixes* (définition 4 page 32).

1.4 Conclusion

Le modèle des réseaux associatifs a été défini dans un contexte de traitement d'images, d'architectures parallèles spécialisées, et de modèle de calcul à grain fin, mixant calculs et communications.

L'intérêt de ce modèle réside dans ses primitives de haut niveau (les `associations`), qui admettent une mise en œuvre électronique efficace (la Maille Associative). Le modèle étend celui des calculs préfixes et des *scans* ; il permet d'utiliser des α-extensions et des β-réductions. Il est fortement lié à un graphe de base fixé, mais le codage des sous-réseaux par les masques, utilisés comme des variables parallèles, permet de manipuler des objets irréguliers.

Le modèle présente cependant l'inconvénient d'avoir un grain de parallélisme très fin, ce qui complique son implémentation. Les machines parallèles actuelles étant de type MIMD (*e.g.*, réseau de stations), la mise en œuvre efficace du modèle sur ces plate-formes est un problème de recherche en soi. Les bonnes performances démontrées de la Maille Associative ne doivent pas cacher les difficultés de réalisation du circuit. L'association d'une donnée simple (un pixel) à chaque nœud de calcul est très efficace, et l'asynchronisme optimise les communications longues distances. Mais l'électronique d'un nœud, même simple, est multipliée par le nombre de nœuds (*cf.* § 8.1.1 page 153). L'industrialisation de la Maille Associative serait donc chère.

Au sein du projet de recherche des réseaux associatifs, notre travail consiste à étudier les propriétés du modèle, et à en élargir les applications. Ses bonnes propriétés incitent en effet à l'étudier indépendamment de la Maille Associative. Sa mise en œuvre logicielle, au sein d'une bibliothèque de programmation parallèle, a pour but d'ériger les réseaux associatifs en modèle de programmation. Avant de revenir sur la définition du modèle au chapitre 3, nous illustrons au chapitre suivant le bien fondé des concepts du modèle pour les problèmes de traitement d'images.

Chapitre 2

Exemple : la segmentation de Voronoï

Afin de montrer la bonne adéquation du modèle des réseaux associatifs à l'algorithmique d'images, nous développons dès maintenant un exemple complet. La segmentation d'images basée sur les polygones de Voronoï se situe à la frontière entre algorithmique géométrique et traitement d'images. Cet algorithme permet d'illustrer les possibilités du modèle des réseaux associatifs dans ces deux domaines. De plus, cette application est représentative d'une classe assez large d'algorithmes, qui manipulent des données irrégulières, et dont les évolutions — et donc les traitements — ne sont pas uniformes. Nous avons étudié son adaptation au modèle des réseaux associatifs avant la refonte de ce dernier (que nous abordons au chapitre suivant). L'algorithme a été programmé avec la bibliothèque de Simulation de la Maille Associative SAM. Ces travaux sont rapportés dans l'article [Ducourthial et al., 1998].

La segmentation de Voronoï est une application-test que nous utilisons à plusieurs reprises dans notre étude. Ainsi, la refonte du modèle (chapitre 3), et les r-opérateurs (chapitre 5), améliorent très sensiblement cet algorithme : il bénéficie pleinement de la philosophie de programmation asynchrone (chapitre 7). Enfin, le code du programme, développé pour la bibliothèque de Simulation de la Maille Associative (SAM), est aisément adapté à notre bibliothèque de programmation parallèle ANET ; nous présentons donc des comparaisons entre SAM et ANET au chapitre 8.

Après une présentation de la segmentation de Voronoï, et de nos motivations dans le choix de cet algorithme, nous décrivons son adaptation au modèle des réseaux associatifs.

2.1 Présentation

2.1.1 Segmentation d'images*

Problème à résoudre. La segmentation est un outil couramment utilisé en analyse d'images, dont le but est de décrypter la structure de l'image en vue d'une description symbolique [Jolion, 1998]. La recherche de la structure nécessite l'identification des «objets» ou *régions*. Une région est définie par la donnée d'un ensemble connexe de pixels : à partir d'un pixel de la région, on peut gagner tout autre pixel de la région sans sortir de celle-ci. Pour repérer les objets, on associe à chaque pixel un identifiant, qui lui indique son appartenance à telle ou telle région. La segmentation d'images consiste à trouver une partition de l'image qui vérifie certains critères, définis par les paramètres de l'algorithme.

Les régions peuvent être différenciées par la moyenne ou par l'amplitude de leurs niveaux de gris. On peut n'admettre que des régions ayant au moins une certaine superficie, une certaine forme (zones convexes par exemple), ou une frontière suffisamment courte par rapport à leur surface. Ces critères simples sont souvent combinés entre eux. Par exemple, on recherche les régions d'une certaine géométrie (régions convexes par exemple), et respectant un critère d'homogénéité sur les niveaux de gris (amplitude ou écart à la moyenne borné). D'autres critères plus complexes ont aussi été proposés : similarité statistique de la distribution des pixels de la région, critère de qualité globale de la segmentation (l'objet s'agrandit seulement si cela améliore la qualité globale de la segmentation), etc.

Une fois les paramètres définis, on cherche à minimiser le nombre de régions résultant de la segmentation et respectant les critères. Ainsi, si la réunion de deux objets vérifie les critères de l'algorithme, la segmentation n'est pas terminée car ces deux objets ne doivent plus en faire qu'un seul. La segmentation de l'image dépend des paramètres choisis, et, pour des paramètres fixés, n'est pas unique. L'œil humain pratique couramment et très rapidement cet algorithme. C'est généralement lui qui fixe la norme floue de la «bonne segmentation». D'une image à une autre, les critères permettant d'obtenir une bonne segmentation (jugée comme telle par l'œil humain) peuvent varier. Les critères optimaux peuvent être approchés par des itérations sur les segmentations, nourries par un *feedback* sur la qualité de la précédente segmentation. Nous considérons dans notre travail le cas d'une seule segmentation, paramétrée par des critères fixés [a].

Algorithmes. Divers algorithmes ont été proposés pour la segmentation d'images, chacun étant adapté à une famille de critères particulière. Citons, par exemple, la segmentation par seuillage, qui n'utilise que l'histogramme de l'image. Cette méthode ne prend pas en compte les relations de voisinages entre les pixels ; elle n'est justifiée que si les valeurs des pixels sont indépendantes les unes des autres. Généralement ces valeurs sont liées, et les *méthodes contextuelles* sont plus appropriées. Dans ce cas, la détermination des objets doit prendre en compte le voisinage de chacun d'entre eux, lui-même défini par la segmentation. On procède alors par raffinements successifs des régions. À partir d'une segmentation intermédiaire, plusieurs choix de raffinement sont possibles ; certains choix malheureux pourraient hypothéquer les chances de trouver une bonne solution finale. La segmentation d'images s'apparente alors à une forme d'optimisation combinatoire : minimisation du nombre de régions sous contrainte d'un critère de qualité sur les objets, ou sur la segmentation dans son ensemble. On a recours à des méthodes de relaxation, mais de nombreuses heuristiques plus rapides ont été développées pour respecter les contraintes temporelles des chaînes de traitement d'images (*cf.* figure 1 page 4).

2.1.2 Segmentation de Voronoï*

Découpe et fusion. Les algorithmes qui procèdent par raffinements successifs des régions travaillent par étapes ; chaque étape admet une segmentation temporaire, qu'il faut améliorer pour passer à l'étape suivante. Le processus s'arrête lorsque la segmentation vérifie les critères fixés, et que le nombre de régions ne peut plus être diminué. Il s'agit d'une heuristique car l'algorithme s'arrête lorsque deux régions voisines ne peuvent former une seule région qui vérifie encore les critères. Mais une meilleure solution pourrait peut-être apparaître en remettant en cause plus profondément la segmentation de l'étape courante.

Ces itérations sur les segmentations temporaires peuvent être amorcées soit par les régions réduites à un seul pixel, soit par la plus grande des régions possible : l'image entière. Dans le premier cas, on procède à des *fusions* de régions, et, dans le second, à des *découpes*. Lorsqu'on utilise des critères d'homogénéité, les régions réduites à un pixel vérifient le critère, et la segmentation maximale de l'image est une première solution. On cherche alors à fusionner les petites régions, tant que la réunion de deux régions est encore suffisamment homogène. Mais si les régions finales sont grosses — ce qui arrive si les critères sont lâches (grande amplitude des niveaux de gris autorisée par exemple) —, on a intérêt à partir de l'image entière et à la découper, pour obtenir une solution en moins d'étapes.

Dans [Horowitz et Pavlidis, 1976], les auteurs ont proposé une amélioration du processus en pratiquant d'abord des découpes simples, tant que les régions ne respectent pas des *critères de découpe*, et ensuite des fusions jusqu'à ce que des *critères de fusion* soient respectés. C'est l'algorithme de

(a) Nous n'avons pas cherché à améliorer les résultats des segmentations que nous présentons dans ce document, pour nous concentrer sur leur programmation. En particulier, pour obtenir une bonne segmentation, il faudrait effectuer un prétraitement, utiliser des critères distincts pour les phases de fusion et de découpe, interdire les régions trop petites (quitte à conserver quelques régions non homogènes), et procéder par tâtonnement pour approcher les critères optimaux. Nous avons appliqué les segmentations aux images brutes, avec des critères fixés une fois pour toutes.

segmentation par découpe et fusion[29]. Suivant les plates-formes (machine parallèle ou séquentielle, etc.) ou les critères choisis, l'algorithme offre un gain de temps par rapport aux précédents (découpe ou fusion), ou permet d'obtenir des segmentations de meilleure qualité. En contrepartie, il requiert à la fois des critères pour la découpe et pour la fusion. Or, comme nous l'avons expliqué au § 2.1.1, les critères optimaux dépendent généralement de l'image elle-même, et la multiplication des critères ne simplifie pas la conception de la chaîne d'analyse d'images.

29. En anglais, *Split-and-Merge*.

La phase de découpe consiste à analyser chaque région de la segmentation courante. Si une région ne vérifie pas les critères d'homogénéité, alors elle est découpée — généralement en quatre quadrants (figure 21) — pour donner des régions plus petites, qui auront plus de chance d'être homogènes. Lorsque toutes les régions sont homogènes, la phase de fusion commence. Il s'agit d'examiner tous les couples de régions voisines, et de fusionner celles qui peuvent l'être. Mais si trois régions peuvent fusionner deux à deux, rien ne dit que la réunion des trois régions est, elle, homogène. Il faut donc choisir les régions qui fusionnent. C'est la non remise en cause de ces choix dans les étapes ultérieures qui constitue l'heuristique.

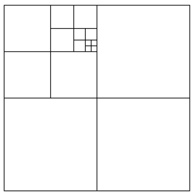


FIG. 21 – Découpe en 4 quadrants.

Régions de Voronoï. Considérons un espace, éventuellement discret, muni d'une distance, et des points g_i de cet espace, appelés *graines*[30]. Les *régions de Voronoï*, chacune centrée sur une graine unique g_i, définissent une partition de l'espace en régions $\mathrm{vor}(g_i)$, telle que la distance $\mathrm{d}(p, g_i)$ de chaque point p à la graine g_i de sa région vérifie $\mathrm{d}(p, g_i) = \mathsf{min}_{g_k \in \{\text{graines}\}} \mathrm{d}(p, g_k)$. En d'autres termes, chaque pixel choisit sa graine parmi les plus proches. Cette partition de l'espace a de nombreuses applications en algorithmique géométrique (*cf.* § 7.5.2 page 143).

30. En anglais, *seed*.

La découpe des régions en quatre quadrants souffre d'être trop régulière, et conduit à des artefacts, notamment sur des zones de dégradé (figure 22). Dans [Ahuja et al., 1985] [Tuceryan et Jain, 1990], les auteurs ont proposé d'ajouter aux critères d'homogénéité de l'algorithme de découpe et fusion une nouvelle contrainte : la segmentation de l'image doit correspondre à une partition en régions de Voronoï. La segmentation obtenue n'est plus régulière, mais elle est plus complexe à construire. L'algorithme de la segmentation de Voronoï est donné ci-dessous. Les figures 23 et 24 sont les régions de Voronoï des première et deuxième itérations de l'algorithme sur l'image *lena*. La série complète est présentée sur la figure 129 page 188. Notons que la métrique utilisée influence la forme des régions ; nous discutons des diverses métriques au § 7.5.1 page 143.

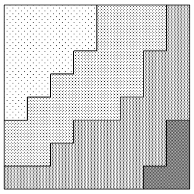


FIG. 22 – Exemple d'artefacts de la segmentation basée sur les quadrants, sur une image de dégradé.

Algorithme 1 Segmentation de Voronoï

▶ *Découpe :*
Choix aléatoire des graines.
répéter
    Détermination des régions de Voronoï centrées sur les graines.
    Calcul du critère d'homogénéité de chaque région.
    Ajout d'une graine au hasard dans chaque région non-homogène.
jusqu'à ce que aucune graine n'ait été ajoutée durant la boucle
▶ *Fusion :*
répéter
    Sélection des régions pouvant fusionner (celles qui sont entourées de régions ayant des niveaux de gris «semblables»)
    Restriction de la sélection pour prévenir les fusions de régions voisines (ce qui évite d'obtenir des régions fusionnées non-homogènes[(b)]).
    Suppression des graines des régions admises à fusionner.
    Agrandissement les régions de Voronoï entourant les régions qui disparaissent.
    Re-calcul du critère d'homogénéité sur les régions agrandies.
jusqu'à ce que aucune graine n'ait été supprimée durant la boucle

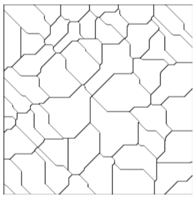


FIG. 23 – Régions de Voronoï (métrique d_8 *cf.* § 7.5.1).

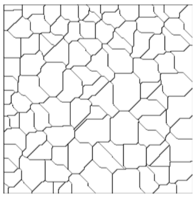


FIG. 24 – Régions de Voronoï plus fines (métrique d_8).

2.1.3 Motivations

La segmentation de Voronoï est un algorithme de traitement d'images qui présente un parallélisme de données difficile à exploiter, pour trois raisons principales.

(b) Trois (ou plus) régions voisines peuvent en effet respecter les critères de fusion deux à deux sans que leur fusion ne donne une région respectant les critères d'homogénéité.

La première vient de l'irrégularité de l'évolution des données. Comme toute segmentation d'images, celle de Voronoï présente une irrégularité des traitements fortement liée à la nature de l'image. Cette nature est, par définition, imprévisible, sauf à exécuter l'algorithme de segmentation lui-même. Ainsi des zones très peu homogènes demanderont beaucoup de traitements, et conduiront à de nombreuses petites régions. Alors que d'autres zones seront «très calmes», et convergeront rapidement vers une partition répondant aux critères géométriques et d'homogénéité de l'algorithme.

La seconde source de difficulté est liée à l'irrégularité de la forme des données. Une segmentation par découpe et fusion en quadrants présente des régions carrées, relativement faciles à manipuler. Au contraire, les régions de Voronoï ont des contours très variés, et ne sont pas simples à coder dans un langage à parallélisme de données classique.

Enfin, une part non négligeable du travail requis par la segmentation de Voronoï porte sur les relations inter-pixels, et non sur les valeurs de ces pixels. Cette propriété est due au critère géométrique des régions : la recherche des régions de Voronoï, centrées sur des graines données, ne prend pas en compte les niveaux de gris, mais les positions relatives de nœuds les uns par rapport aux autres. L'exploitation en parallèle des relations inter-pixels n'est pas un problème trivial.

Ainsi, la segmentation de Voronoï présente des caractéristiques qui nous permettent d'évaluer les performances des réseaux associatifs en matière de parallélisme irrégulier (dans les traitements et dans le codage des données), et de maniement de l'information codée par les relations inter-pixels.

2.2 Implémentation dans le modèle des réseaux associatifs

Dans cette section, nous décrivons l'implémentation dans le modèle des réseaux associatifs de l'algorithme de la segmentation de Voronoï. Le programme est destiné à la Maille Associative ; il a été programmé avec la bibliothèque SAM de simulation de la Maille Associative. Il en utilise donc les primitives [(c)].

On peut distinguer deux types de mouvements de données dans la mise en œuvre de la segmentation de Voronoï. Le premier concerne la manipulation des régions, que l'on représente par les composantes fortement connexes d'un graphe, dont les noeuds sont les pixels, et qui évolue dynamiquement au cours du calcul. Sur ces régions, il est nécessaire d'une part de réaliser des calculs à même de définir les régions de Voronoï, et d'autre part de déterminer les propriétés d'homogénéité de ces ensembles (pour la segmentation). Nous montrons que les primitives de calcul du modèle des réseaux associatifs (les `associations`) permettent d'implémenter efficacement ces opérations, soit directement, soit de manière itérative.

À l'inverse, nous devons également considérer les relations entre régions voisines, notamment pour la phase de fusion. Il s'agit alors de manipuler le *graphe d'adjacence des régions*, dont les nœuds sont formés de régions entières, et dont les arcs relient les régions voisines. Ce type de graphe ne rentre pas directement dans le cadre du modèle des réseaux associatifs, dans lequel les nœuds sont plutôt des pixels que des régions entières. Par contre, il est possible d'étendre le modèle simplement, pour pouvoir appliquer les `associations` sur ces graphes. C'est le mécanisme des *graphes virtuels* que nous présentons ci-dessous.

2.2.1 Graphe virtuel

Codage à l'aide de graphes. Lorsque le traitement des régions fait intervenir leurs relations de voisinage, il est grandement facilité par un codage à base de graphe. À partir des régions de l'étape

(c) La bibliothèque de programmation ANET, présentée au chapitre 8 est compatible avec SAM ; ce programme tourne donc sous ANET. Mais ANET propose aussi la primitive `direct-association` définie au chapitre suivant, et l'algorithme est largement amélioré par cette primitive, que ce soit en temps d'exécution ou en «expressivité». Le principe de l'algorithme reste cependant le même ; l'implémentation décrite ici nous sert d'étalon pour évaluer notre travail.

courante, on cherche les fusions possibles entre régions adjacentes, ce qui revient à définir un sous-graphe [d] du graphe des régions. Les incompatibilités, ou les choix multiples, peuvent être résolus dans ce sous-graphe. Cette approche nécessite des outils de codage et de manipulation de l'image sous forme de graphe, mais également de calculs sur ces graphes. Bien que séduisante, on lui préfère souvent des méthodes plus simples à mettre en œuvre sur des machines en tableau de processeurs, ou moins gourmandes en mémoire pour les bibliothèques de calcul.

Le modèle des réseaux associatifs offre au programmeur les outils de gestion des graphes, ainsi qu'un passage souple du graphe aux pixels des régions qui définissent ses sommets. Cette caractéristique du modèle le rend particulièrement adapté aux algorithmes de segmentation. Afin de faciliter le passage des pixels aux régions, nous avons défini le concept de *graphe virtuel*. Un graphe virtuel est une modélisation macroscopique, obtenue en regroupant par «paquets» les nœuds d'un réseau associatif. Cette technique s'avère utile pour un grand nombre d'algorithmes. Elle sert aussi à plonger diverses topologies dans le réseau (*cf.* annexe C).

Graphe virtuel. Un *graphe virtuel*, ou *vgraph*, $G_v(V_v, E_v)$ est défini à partir de deux sous-graphes [e] du graphe de base $G_b(V_b, E_b)$ (*cf.* § 1.2.1 page 12). Le premier, appelé *graphe des sommets*, est noté $G_s(V, E_s)$ avec $E_s \subset E$. Il est tel que tous les arcs du graphe de base reliant deux sommets de la même composante connexe[31] de G_s sont présents dans E_s. Le second sous-graphe est appelé *graphe des arcs* ou *graphe frontière* et est noté $G_a(V, E_a)$, avec $E_a \subset E$. Les arcs de G_a relient des sommets qui n'appartiennent pas à la même composante fortement connexe du graphe des sommets G_s. On a donc $E_s \cap E_a = \emptyset$. Les sommets du graphe G_a qui ont un arc incident (entrant ou sortant) sont appelés *sommets frontières*. Les sommets du graphe virtuel sont les composantes connexes de G_s. Deux sommets $\widetilde{u}$ et $\widetilde{v}$ du graphe virtuel sont reliés par un arc de E_v s'il existe un arc dans E_a qui relie un sommet de la composante connexe $\widetilde{u}$ à un sommet de la composante $\widetilde{v}$. Un exemple de graphe virtuel dans une maille 4×4 est donné sur la figure 25.

31. Composante connexe (CC) : ensemble maximal de sommets tel qu'il existe une suite d'arcs consécutifs, mais pas forcément de même direction, entre toutes paire de sommets.

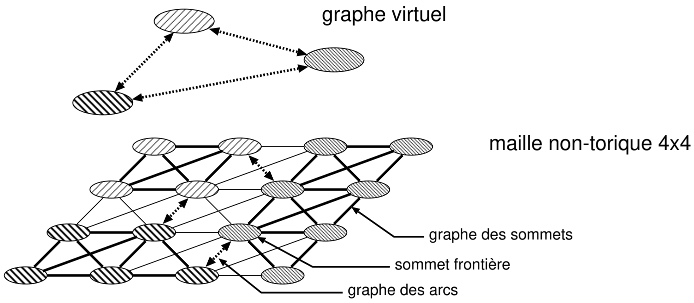


FIG. 25 – *Un graphe virtuel est composé d'un graphe des sommets et d'un graphe des arcs.*

Calcul sur les graphes virtuels. Les primitives du modèle des réseaux associatifs sont étendues aux réseaux virtuels. Cette technique peut être utilisée dans la Maille Associative pour les opérateurs associatifs, commutatifs et idempotents [f]. En effet, l'application d'une $\diamond$-`association` sur un réseau virtuel se traduit par deux `associations` dans le réseau réel. Tout d'abord, en utilisant une `max-association` (par exemple), on obtient une seule valeur par composante connexe du graphe des sommets (ici le nœud virtuel obtient la plus grande valeur des nœuds le composant). Ces valeurs se retrouvent donc également sur les nœuds frontières. Ensuite on pratique une $\diamond$-`association` sur le sous-réseau formé de l'union du graphe des sommets et du graphe des arcs. L'opérateur $\diamond$ doit être

(d) *cf.* note **(f)** page 12.

(e) Un graphe virtuel pourrait aussi être défini à la manière d'un graphe quotient, mais nous préférons la formulation utilisant deux sous-graphes, plus simple à manipuler en pratique.

(f) L'opérateur $\diamond$ est associatif et commutatif par définition de l'`association` (§ 1.2.1 page 12). Il doit être idempotent car les sommets virtuels sont des composantes fortement connexes qui présentent donc des circuits (*cf.* proposition 3 page 31). Il s'agit donc d'un s-opérateur (*cf.* définition 13 page 40). La mise en œuvre des graphes virtuels dans une bibliothèque de programmation peut s'affranchir de cette limitation.

idempotent. Pour effectuer une ⋄-`step-association` sur le graphe virtuel, c'est une ⋄-`step-association` qui est appliquée sur la réunion des deux sous-réseaux[(g)]. Mais, dans ce cas, il faut encore diffuser le résultat obtenu sur les nœuds frontières à tous les nœuds des sommets du graphe virtuel. La figure 26 illustre cette technique.

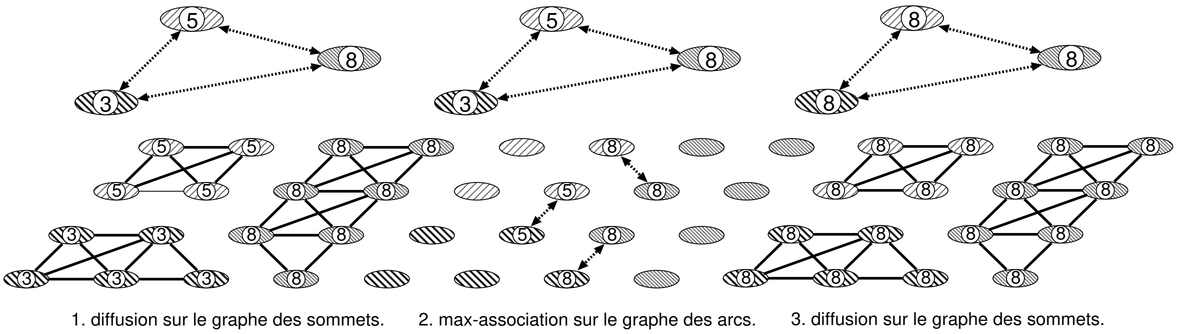


FIG. 26 – max-*`step-association` sur le graphe virtuel.*

2.2.2 Détermination des régions de Voronoï

À chaque itération de la phase de découpe de l'algorithme 1 page 21, il faut obtenir des régions de Voronoï. La construction de ces régions est effectuée par améliorations successives des régions associées à chaque graine. L'algorithme 2 page suivante récapitule ce processus, que nous développons ci-dessous. Le programme correspondant est donné en annexe, au § A.1 page 185 ; la figure 129 page 188 donne la série des images provenant de la découpe de Voronoï sur l'image *lena*.

Les régions initiales sont déterminées par le plongement, à l'aide de la primitive `spanning-tree` (*cf.* § 1.2.2 page 13), d'une forêt couvrante dans le réseau, enracinée aux graines g_i. On diffuse ensuite l'adresse de chaque graine dans l'arborescence correspondante. La primitive `link-with-equals` permet alors de former les régions (*cf.* lignes 1, 2 et 6 de l'algorithme 2, et figure 27). Dans la Maille Associative, cette primitive est implémentée par une propagation asynchrone de jetons ; elle ne garantit donc pas qu'un sommet p_j appartenant à l'arborescence de racine g_i appartienne aussi à la région de Voronoï $\mathrm{vor}(g_i)$. Néanmoins, la construction de la forêt fournit une segmentation initiale qui n'est pas très éloignée d'une partition en régions de Voronoï (figure 27).

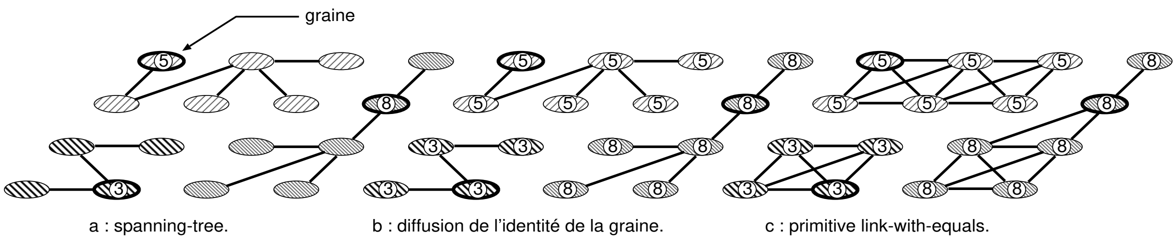


FIG. 27 – *Construction des régions initiales.* ***(a)*** *Plongement d'une forêt couvrante enracinée sur les graines.* ***(b)*** *Diffusion des adresses des graines dans leur arborescence.* ***(c)*** *Connexion des sommets des régions par la primitive* `link-with-equals`.

Les étapes suivantes consistent à améliorer ces régions initiales, jusqu'à ce qu'elles deviennent de réelles régions de Voronoï. En parallèle, chaque région R_a examine ses voisines, successivement (lignes 8 à 14 de l'algorithme 2 page suivante). Les nœuds frontières récupèrent la plus grande adresse de la graine de leurs voisins dans les autres régions (ligne 9, pvar m). La plus grande adresse de la région voisine est ensuite diffusée sur toute la région R_a (ligne 10, pvar M). Soit R_b cette région voisine de plus grande adresse. Chaque pixel de R_a vérifie s'il n'est pas plus près de la graine de R_b que de celle de R_a. La pvar *nouvelleAdresseGraine* note le meilleur choix de chaque nœud. Puis

(g) Pour la `step-association`, tout opérateur peut être utilisé.

les arcs du graphe frontière qui relient les nœuds frontières de R_b à ceux de R_a sont supprimés, de telle sorte qu'une autre région voisine, de plus grande adresse parmi les régions voisines restantes, soit choisie à l'itération suivante. On sort de la boucle quand toutes les régions voisines ont été examinées, ce qui correspond au fait qu'il n'y a plus d'arc dans le graphe frontière. La figure 29 illustre cette boucle interne.

Lorsque toutes les régions voisines ont été examinées, la région R_a n'a conservé que les sommets qui étaient plus proches de sa graine que de celles des régions voisines (lignes 6). Cela ne garantit pas encore que l'image est partitionnée en régions de Voronoï, comme le montre la figure 28 : une région voisine très déformée R_b pourrait faire croire au pixel que la graine de sa région R_a est la plus proche sans que cela soit vrai. On itère donc ce processus jusqu'à ce que toutes les régions soient stables (lignes 5 à 20).

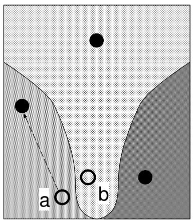
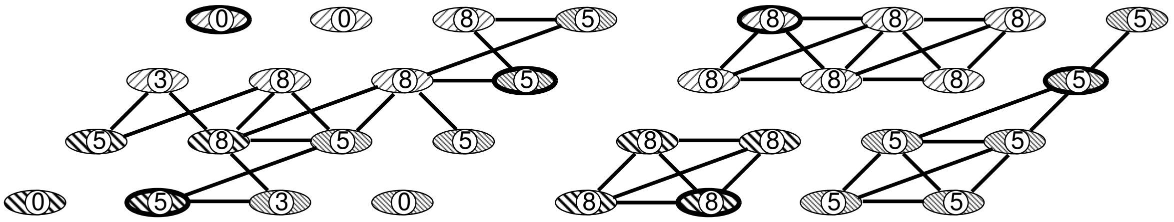
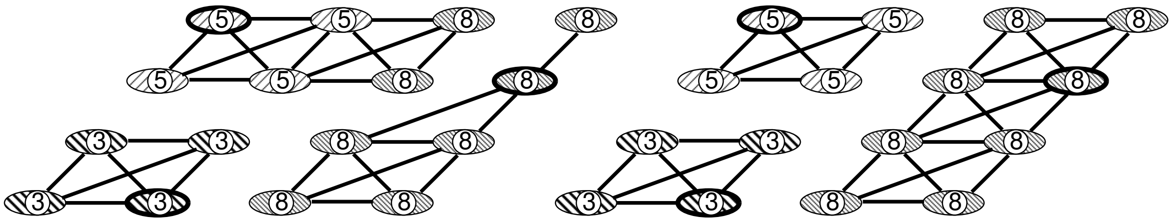


FIG. 28 – Le pixel a compare sa graine avec celle du pixel b, et conclut, à la première itération, que sa graine est la plus proche.



a : Chaque région obtient les numéros des graines de ses voisines avec une max-step-exclusive-association sur le graphe des arcs (ie. les arcs reliant les régions).

b : Chaque région choisit une seule voisine, par la diffusion du maximum des numéros de graines obtenus, avec une max-association sur le graphe des régions.



c : Chaque pixel choisit la graine la plus proche entre celle de sa région et celle de la région voisine choisie.

d : Mise à jour du graphe des sommets (ie. les arcs reliant les pixels d'une même région) : les sommets ayant choisis la même graine sont connectés, avec la primitive link-with-equals.

FIG. 29 – *Une étape de la boucle sur les régions voisines, à partir des régions de la figure 27c. La* `step-exclusive-association` *donne l'élément neutre de l'opérateur (ici 0 pour le* max*) sur les sommets n'ayant pas d'antécédents et qui ne reçoivent donc aucune valeur (figure 29 a). Ici, les régions de graine 3 et 5 choisissent leur plus grande voisine, de graine 8. La région de graine 8 choisit sa plus grande voisine, de graine 5 (figure 29 b). On obtient les régions de Voronoï en effectuant une boucle sur ces boucles, jusqu'à complète stabilisation (plus aucune évolution).*

Algorithme 2 ConstructionRégionsVoronoï(**pvar** *graines*)

▶ *Construction des régions initiales :*
1 **pvar**⟨ **masque** ⟩ *forêt* ← `spanning-tree`(*graines*) ▶ *cf. figure 27 a.*
2 **pvar** *adresseGraine* ← `max-association`(*forêt*, *graines*) ▶ *cf. figure 27 b.*
3 **pvar**⟨ **masque** ⟩ *régions*, *frontière*
4 **pvar** *nouvelleAdresseGraine*
▶ *Boucle jusqu'à stabilité des régions :*
5 **tant que** vrai **faire**
6 *régions* ← `link-with-equals`(*adresseGraine*, *adresseGraine*)
▶ *cf. figures 27 c et 29 d.*
7 *frontière* ← complémentaire de *région*
▶ *Boucle sur les régions voisines*
8 **tant que** *frontière* contient des arcs **faire**
9 m ← `max-step-exclusive-association`(*frontière*, *adresseGraine*) ▶ *cf. figure 29 a.*
10 M ← `max-association`(*régions*, m) ▶ *cf. figure 29 b.*
▶ *La pvar M contient la plus grande adresse de graine des régions voisines (i.e., sélection d'une seule région voisine).*

nouvelleAdresseGraine ← *adresseGraine*
Comparaison des distances aux graines données par *adresseGraine* et M. L'adresse de la graine la plus proche est notée dans *nouvelleAdresseGraine*. ▶ *cf. figure 29 c.*
Suppression des arcs de la pvar de masques *frontière* en provenance de la région traitée (adresse de la graine = M), de façon à choisir de nouvelles régions voisines lors de la prochaine itération.
fin tant que
si *adresseGraine* == *nouvelleAdresseGraine* **alors**
▶ *Egalité partout vérifiée.*
Sortir de la boucle
sinon
adresseGraine ← *nouvelleAdresseGraine*
fin si
fin tant que

2.2.3 Calcul du critère d'homogénéite

Les critères d'homogénéité sont calculés par quelques `associations` sur les régions, à partir des niveaux de gris des pixels contenus dans la pvar *image*. Par exemple, pour calculer la variance d'une région, on calcule la somme des valeurs des pixels par une `plus-prefix-association`($forêt^{-1}$, *image*), sur la forêt couvrante du graphe des régions, préalablement inversée (orientée des feuilles vers les racines). Pour limiter le nombre de bits nécessaires (largeur de la pvar résultat), on peut soustraire à la pvar des niveaux de gris le niveau minimum de leur région, obtenu par une `min-association`(*régions*, *image*). On calcule ensuite le nombre de pixels par une `plus-prefix-association`($forêt^{-1}$, p) avec $p = 1$. Les moyennes *moy* des niveaux de gris des pixels de chaque région sont alors calculées sur les graines (racines de la forêt) ; *moy* vaut 0 ailleurs que sur les graines. On diffuse les moyennes à tous les sommets de la région par une `max-association`(*forêt*, *moy*). Ces derniers calculent alors leur écart à la moyenne (pvar `écart`). Il ne reste plus qu'à diffuser le plus grand écart par une `max-association`(*régions*, *écart*).

Le critère de l'amplitude est calculé par `max-association`(*régions*, *image*) − `min-association`(*régions*, *image*). Il est plus simple et permet de travailler avec des pvars ayant le même nombre de bits que celle de l'image, ce qui n'est pas le cas avec la variance (qui nécessite le calcul de la somme des valeurs des pixels des régions).

2.2.4 Ajout d'une nouvelle graine

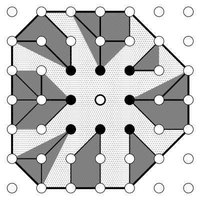


FIG. 30 – Détermination de 8 octants, en plongeant une forêt couvrante, enracinée sur les voisins (en noir) de la graine (au centre). La nouvelle graine sera créée dans l'octant le moins homogène.

L'ajout d'une nouvelle graine est effectué par tirage d'un nombre aléatoire. On peut sélectionner le pixel ayant obtenu le plus grand nombre par une `max-association`(*régions*, *hasard*). Pour éviter les doubles, on ne conserve que les pixels ayant obtenu le plus grand nombre aléatoire de leur région, et qui ont aussi la plus grande adresse. Cette heuristique est suffisante en pratique.

Mais il est aussi possible d'améliorer le placement de la nouvelle graine. En effet, il est inutile qu'elle tombe dans une sous-région homogène de la région. L'algorithme 3 sélectionne la partie de la région la moins homogène, et ajoute une graine vers le mileu de cette zone.

Algorithme 3 AjoutGraine

Marquer les (au plus [(h)]) huit voisins des graines par une `or-step-association`.
Plonger, dans chaque région, (au plus) huit forêts couvrantes, enracinées sur les voisins de la graine ; on obtient huit octants.
Calculer sur ces arborescences le niveau de gris moyen, puis le plus grand écart à ce niveau moyen.
Sélectionner l'octant ayant le plus grand écart.
Désigner comme nouvelle graine l'un des sommets situé à mi-hauteur de l'arborescence sélectionné.

2.2.5 Fusion des régions

La fusion de régions consiste à réduire le nombre total de régions, tout en leur conservant leur homogénéité et la propriété de région de Voronoï. Pour cela, on recherche les régions entourées de

(h) Les graines au bord de la maille ont moins de huit voisins.

régions semblables sur le plan de l'homogénéïté. Lorsqu'une telle région est repérée, on supprime sa graine, et les régions voisines s'agrandissent pour la recouvrir. La segmentation reste donc une partition en régions de Voronoï. Pour éviter les problèmes de fusions de régions voisines, qui peuvent conduire à des régions non homogènes (*cf.* note **(b)** page 21), ou ne respectant plus la propriété de Voronoï, on interdit les suppressions de graines de régions voisines. La technique des graphes virtuels simplifie ce type de contrôle. L'algorithme 4 détaille le processus de la fusion. La figure 130 page 189 donne la série des images provenant de la fusion de Voronoï appliquée à l'image *lena*.

Algorithme 4 FusionVoronoï

▶ vg *est le graphe virtuel des régions ;* m *est la pvar des minimums de chaque région,* M *celle des maximums ; graine est une pvar booléenne vraie sur les graines, et fausse ailleurs*

répéter
  MAX ← `max-step-association`(vg, M)
  MIN ← `min-step-association`(vg, m)
  ▶ *Calcul du critère d'homogénéité :*
  H = MAX − MIN
  partout où H < seuil **faire**
    Création d'un vgraphe *vgh* connectant un nœud à ses voisins ayant H < seuil ; il permet d'interdire les fusions de régions voisines.
    ▶ *Sélection d'un seul sommet de vgh (région) par composante fortement connexe du graphe virtuel. Seules ces régions peuvent disparaitre à cette étape.* $\imath$ *est la pvar index.*
    partout où `max-association`$(vgh, \imath) == \imath$ **faire**
      graine ← faux
    fin partout où
  fin partout où
  Reconstruction les régions de Voronoï
jusqu'à ce que aucune graine n'a été ajoutée

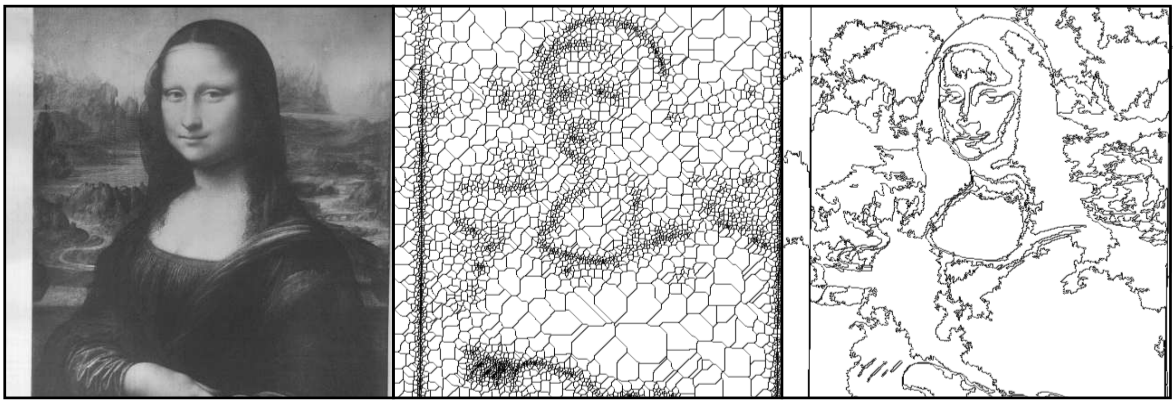


FIG. 31 – *Segmentation de Voronoï sur l'image monalisa,* 432×440 *pixels, 8 bits (256 niveaux), 200 graines initiales, critère d'homogénéité pour la découpe et la fusion de 60. L'image du milieu donne les régions obtenues à la fin de la phase de découpe. L'image de droite donne les régions obtenues après la phase de fusion.*

2.3 Conclusion

La segmentation est un problème important en traitement d'images ; elle est généralement programmée avec une heuristique. Le problème se complique lorsque le critère porte à la fois sur l'homogénéité des régions et sur leur forme, comme c'est le cas avec la segmentation de Voronoï. Les solutions utilisant un codage à base de graphe sont connues pour simplifier l'algorithme.

Le modèle des réseaux associatifs permet un passage souple du codage de l'image par les pixels au codage par les régions, dans un sens ou dans l'autre, grâce aux graphes virtuels. Les `associations` permettent d'exprimer un calcul à la fois au niveau du pixel, et au niveau de la région.

Le modèle des réseaux associatifs se révèle bien adapté pour exprimer des algorithmes parallèles sur des données très irrégulières. En outre, il permet, grâce aux sous-graphes, d'exploiter l'information codée par les relations inter-pixels.

Nous abordons la gestion de l'irrégularité des traitements au chapitre 8. Les opérateurs que nous développons au chapitre 5 permettent d'améliorer l'algorithme que nous avons présenté (*cf.* § 7.6 page 146). Pour utiliser de tels opérateurs, il nous faut tout d'abord définir une nouvelle primitive ; c'est l'ojet du chapitre suivant.

Chapitre 3

Refonte du modèle des réseaux associatifs

L'étude du modèle des réseaux associatifs en tant que tel, conduit à approfondir la relation qui existe entre le modèle théorique et la Maille Associative, son implémentation électronique. Au § 3.1, nous mettons en avant des différences entre le modèle et sa mise en œuvre. Cependant, la Maille Associative n'apparaît pas comme une implémentation partielle du modèle, mais comme la mise en œuvre d'un modèle de calcul différent. Ce modèle a des points communs avec les réseaux associatifs, mais présente aussi un intérêt propre, en ce sens qu'il autorise des calculs non spécifiés dans les réseaux associatifs.

Nous proposons alors au § 3.2 une refonte du modèle des réseaux associatifs, qui se traduit essentiellement par la considération d'une nouvelle primitive de calcul : la `direct-association`.

Au § 3.3, nous étudions le problème de la construction de l'`association` à partir de la `direct-association`, ce qui revient à étudier la mise en œuvre de l'`association` dans la Maille. Les difficultés rencontrées confirment le bien-fondé de la définition d'une nouvelle primitive. Cette redéfinition est à la base de la problématique développée dans notre travail ; nous la détaillons en conclusion, au § 3.4 page 44.

3.1 Nécessité d'une nouvelle définition

Nous examinons dans cette section les différences entre le modèle des réseaux associatifs et sa mise en œuvre dans la Maille Associative.

3.1.1 Restriction du modèle.

On considère ici le modèle des réseaux associatifs comme un modèle de programmation distinct du modèle d'exécution de la Maille Associative.

Associativité et commutativité. La définition de l'`association` précise que l'opérateur utilisé doit être associatif et commutatif (§ 1.2.1 page 12). En effet, la structure de données manipulée par la primitive est l'ensemble (au sens mathématique du terme, sans doublon) des antécédents de chaque nœud. Les ensembles d'antécédents sont définis par le sous-réseau donné en paramètre à la primitive. Mais, ils ne sont pas structurés. En précisant l'instruction $\diamond$-`association`(net_G, p), le programmeur ne peut pas savoir dans quel ordre les calculs intermédiaires seront réalisés, sauf à avoir une connaissance supplémentaire sur la mise en œuvre du modèle, c'est-à-dire sur le modèle d'exécution sous-jacent (par exemple, il programme la Maille Associative et il sait comment le modèle est mis en œuvre dans la machine).

Supposons, par exemple, que le sommet d du graphe G a l'ensemble d'antécédents $\{a, b, c\}$. Le programmeur ne peut pas savoir si l'ordre des calculs intermédiaires sera $a \diamond b$ puis $(a \diamond b) \diamond c$ ou s'il sera $a \diamond c$ puis $(a \diamond c) \diamond b$, etc. Les réseaux associatifs, en tant que modèle de programmation, n'impliquent pas une implémentation particulière des calculs.

À supposer que les données soient ordonnées, si l'ordre des calculs intermédiaires ne peut l'être, l'opérateur binaire utilisé doit être associatif, car on ne peut assurer que les calculs intermédiaires seront réalisés de gauche à droite[(a)]. Si, de plus, les données ne sont pas ordonnées, l'opérateur doit être commutatif, car on ne peut assurer que les calculs seront réalisés en prenant les données dans un ordre précis[(b)]. L'absence d'information sur la mise en œuvre des calculs sur un ensemble mathématique impose donc des contraintes d'associativité et de commutativité à l'opérateur utilisé, de telle sorte que le même résultat soit obtenu pour toutes les implémentations de la primitive de calcul.

Pertinence de la structure de données. Ainsi, les restrictions d'associativité et de commutativité de l'opérateur d'une `association` sont dictées par le manque de connaissance de la structure et de l'ordre des données les unes par rapport aux autres. Le modèle est, sur ce point, plus général que ceux du CM-Lisp et des *scans* (§ 1.1.3 page 7), où les données sont ordonnées sous forme de vecteurs. Nous n'avons cependant pas mentionné cette caractéristique dans la liste des apports du modèle au § 1.3.2 page 16, car elle ne nous paraît pas satisfaisante en l'état.

Au niveau de l'utilisation, si la structure de données des ensembles d'antécédents peut paraître très riche, elle reste néanmoins difficile à manipuler. Jusqu'à présent, la programmation des réseaux associatifs a toujours été imaginée en relation avec le modèle d'exécution de la Maille, ce qui levait les ambiguïtés.

Au niveau de la mise en œuvre, le manque de précision sur la structure des données dans le modèle théorique peut apparaître comme une liberté offerte à l'implémentation. Mais elle se traduit aussi par une contrainte sur les caractéristiques de l'opérateur. En fait, la liberté sur la structure n'est pas utilisée dans la Maille Associative qui a, elle, une seule façon de réaliser les calculs intermédiaires, imposée par le silicium de ses connexions. De plus, les contraintes reportées sur l'opérateur (associativité et commutativité) apparaissent artificielles dans la machine (*cf.* § 3.1.3).

Conclusion. L'étude du modèle des réseaux associatifs incite à distinguer modèle de programmation et modèle d'exécution. Or la stricte lecture de la définition de l'`association` laisse place à une ambiguïté, qui est levée par la contrainte d'associativité et de commutativité de l'opérateur. Ce problème est lié à la structure de données ensembliste utilisée pour la définition de la primitive, mais cette structure est délicate à manipuler au niveau du modèle de programmation sans avoir une connaissance supplémentaire sur le modèle d'exécution sous-jacent.

3.1.2 Restrictions de la Maille

Terminaison du calcul. Dans la Maille Associative, un calcul global — mis en œuvre par l'exécution concurrente des calculs locaux jusqu'à stabilisation du réseau — n'est défini que s'il se termine, c'est-à-dire qu'effectivement le réseau se stabilise. Cette contrainte est, bien sûr, directement liée à la gestion de l'asynchronisme dans la Maille, qui est utilisée sans protocole spécifique, tel que celui de la poignée de main, et qui conduit à un asynchronisme fonctionnel (*cf.* § 1.3.1 page 16).

La définition du calcul global fait donc intervenir la terminaison. L'opérateur doit donc respecter certaines conditions, assurant cette terminaison. Les deux propositions suivantes précisent ces conditions, et nous serviront par la suite.

32. DAG : *Directed Acyclic graph.* Graphe orienté sans circuit.

33. Source : sommet n'ayant pas d'antécédent.

Proposition 1 *Si le sous-réseau n'a pas de circuit, la terminaison du calcul global, défini comme l'exécution concurrente des calculs locaux, prend fin au bout d'un temps fini, quel que soit l'opérateur.*

Preuve. Soit G le graphe orienté qui modélise le sous-réseau. Puisque G n'a pas de circuit, c'est un DAG[32]. Dans ce cas, ses sources[33] ont un calcul stable après un seul calcul local. Par récurrence sur la plus grande distance à une source présente dans les antécédents, on montre

(a) Nous supposons dans tout ce document que les expressions sont évaluées de gauche à droite.

(b) L'opérateur doit aussi être associatif dans ce cas, car il est impossible de préciser un ordre sur les calculs intermédiaires sans un ordre sur les données.

que tous les nœuds finissent par avoir un calcul stable au bout d'un temps fini. Le calcul est donc globalement stable après un délai fini. □

Définition 2 *L'opérateur $\diamond$ défini sur l'ensemble $\mathbb{S}$ est k-idempotent faible*[(c)] *si, pour tout $x \in \mathbb{S}$, il existe $k_x \in \mathbb{N}^*$ tel que :*

$$\underbrace{x \diamond \cdots \diamond x}_{k_x+1} = \underbrace{x \diamond \cdots \diamond x}_{k_x}$$

Proposition 3 *Si l'opérateur $\diamond$ n'est pas k-idempotent faible alors le calcul global, défini comme l'exécution concurrente des calculs locaux, ne peut se terminer sur tous les réseaux initialisés avec des valeurs quelconques.*

Preuve. Il suffit d'exhiber, pour tout opérateur non k-idempotent faible, un réseau et une initialisation qui ne conduisent pas à la terminaison. Soit $\diamond$ un opérateur défini sur $\mathbb{S}$ qui n'est pas k-idempotent faible. Considérons un réseau en forme d'anneau, initialisé sur tout nœud avec la valeur $x \neq e_\diamond$. Les expressions calculées par chaque nœud au cours du calcul seront de la forme $x \diamond \cdots \diamond x$, et il ne peut y avoir convergence vers un résultat sans la k-idempotence faible. □

En particulier, la k-idempotence forte, définie par $\exists k \in \mathbb{N}^*, \forall x \in \mathbb{S}, \overbrace{x \diamond \cdots \diamond x}^{k+1} = \overbrace{x \diamond \cdots \diamond x}^{k}$, et que nous appellerons k-idempotence (*cf.* définition 42 page 84), implique la k-idempotence faible[(d)]. Un opérateur tel que l'addition sur les entiers, qui n'est pas k-idempotent faible, ne peut conduire à la terminaison du calcul global pour toute pvar dans un réseau présentant un circuit.

Cette contrainte de l'idempotence n'existe pas dans la définition du modèle. En effet, l'idempotence est nécessaire ici pour prévenir de la prise en compte multiple d'une donnée. Ce problème n'a pas lieu dans le modèle théorique défini à partir de la notion d'ensemble au sens mathématique du terme, sans doublon. Ainsi, les divergences entre modèle et implémentation sont intimement liées à la difficulté de mise en œuvre de la notion complexe d'ensemble mathématique. L'idempotence est une notion centrale dans notre travail ; nous l'utilisons dans les chapitres 5, 6, 7, et nous présentons l'algèbre idempotent au § 4.1 page 51.

Qualificatif préfixe. Nous avons expliqué au § 1.2.1 page 12 que le qualificatif `prefix` est donné aux `associations` exécutées sur des graphes sans circuit, et que cela généralisait le problème du préfixe (*cf.* § 1.3.2 page 16). En fait, ce qualificatif signifie aussi que l'`association` se termine dans la Maille Associative, même lorsque l'opérateur n'est pas idempotent, grâce à l'absence de circuit. Aux sens sémantique et d'implémentation que nous avions évoqués pour le terme `prefix` (§ 1.2.1 page 12), il faut aussi ajouter ce sens d'indication de la terminaison du calcul global utilisant un opérateur non idempotent dans la Maille Associative.

Concordance du résultat avec le modèle. Néanmoins, l'absence de circuit n'est pas suffisante pour prévenir des problèmes d'implémentations d'une `association` utilisant un opérateur non-idempotent.

Considérons le graphe G de la figure 32. Dans le modèle théorique, le nœud c doit recevoir une seule fois la valeur $p[a]$ du nœud a, car le sommet a apparaît une seule fois dans les antécédents $\Gamma_G^-(c)$ du sommet c. Le résultat du nœud c est donc $p[a] \diamond p[b] \diamond p[c]$, si on admet que l'opérateur utilisé est $\diamond$, et en mettant de côté les problèmes d'ordre des données reçues dans les calculs. Or dans la Maille Associative, le nœud b calcule le résultat $p[b] \diamond p[a]$, comme spécifié par le modèle. Et le nœud c

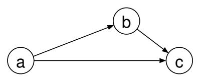


FIG. 32 – `association` sur un DAG qui n'est pas une arborescence.

(c) Dans ce document, et suivant le terme «n-aire», nous préfixons certains termes avec une variable mathématique, qui est un paramètre de la définition. C'est le cas ici avec la k-idempotence faible. Par abus, de même qu'on dira que l'opérateur $\diamond(x_1, \ldots, x_n)$ est n-aire (usage de la même variable n), nous utilisons la variable générique préfixant le terme dans la définition même du terme. Lorsque ce paramètre est déterminé de façon unique, on l'utilise aussi en préfixe. On dira ainsi que l'idempotence correspond à la 1-idempotence (forte), c'est-à-dire à la k-idempotence avec $k = 1$.

(d) Voir aussi la note **(d)** page 128.

finit par calculer $p[c] \diamond p[a] \diamond (p[b] \diamond p[a])$, ce qui veut dire qu'il intègre dans son calcul à la fois la valeur du nœud a reçu par le lien (a, c) et celle reçue par le chemin $(a, b)(b, c)$, à travers le résultat intermédiaire $p[b] \diamond p[a]$. Si l'opérateur $\diamond$ est l'addition, et si $p[a]$ n'est pas l'élément neutre 0 de cet opérateur, alors le calcul réalisé dans la Maille Associative ne correspond pas au calcul spécifié dans le modèle.

Dans les travaux précédents, les `associations` utilisées avec des opérateurs non-idempotents (essentiellement l'addition) ont toutes été réalisées sur des sous-réseaux en forme d'arborescence (plus-`prefix-association`). Néanmoins, les arborescences ne sont pas les seuls réseaux qui assurent l'équivalence entre la définition de l'`association` du modèle et son implémentation dans la Maille, même pour les opérateurs non-idempotent.

Graphe préfixe. Nous définissons donc les sous-réseaux qui permettent de calculer la primitive des réseaux associatifs dans la Maille Associative, même lorsque l'opérateur n'est pas idempotent.

Définition 4 *Un graphe orienté $G(V, E)$ est préfixe si la propriété suivante est respectée : pour toute paire de sommets u et v, s'il existe un chemin de u à v, alors il est unique, et il n'existe pas de chemin de v à u (cf. figures 34 et 35 page suivante).*

On dit qu'il existe un *chemin multiple* entre les sommets u et w s'il existe au moins deux chemins élémentaires[34] différents entre u et w (figure 33). La proposition suivante découle de la définition d'un graphe préfixe. Nous précisons la nature des graphes préfixes ci-après (*cf.* proposition 9 page suivante).

34. Chemin élémentaire : qui ne contient pas deux fois le même sommet.

Proposition 5 *Un graphe orienté est préfixe si et seulement s'il présente ni chemin multiple ni circuit.*

Proposition 6 *Si $\diamond$ est un opérateur associatif, commutatif mais non idempotent, alors les deux propositions suivantes sont équivalentes :*

(a) pour toutes les pvars p, le résultat d'une $\diamond$-`association`(net_G, p) calculée par des calculs locaux concurrents non coordonnés dans la Maille est conforme à la spécification du modèle des réseaux associatifs,

(b) le sous-réseau G est préfixe.

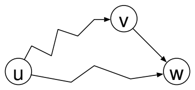


FIG. 33 – Chemin multiple de u à w.

Preuve. (a) ⇒ (b) Supposons que G ne soit pas préfixe, et montrons que **(a)** n'est pas vérifié pour au moins un opérateur associatif, commutatif et non idempotent. Si G n'est pas préfixe, il présente soit un circuit, soit un chemin multiple (proposition 5). Supposons qu'il y ait un circuit dans G, et que l'opérateur $\diamond$ soit l'addition. Dans ce cas, puisque l'équivalence doit apparaître pour toutes les pvars p, il suffit de considérer la pvar $p = 1$, qui ne conduit pas à la terminaison de l'plus-`association`.

Supposons maintenant que G contienne un chemin multiple. Il existe donc trois sommets u, v et w dans G et deux chemins de u à w, dont l'un seulement passe par v (*cf.* figure 33). Puisque l'équivalence doit être vérifiée pour toutes les pvars p, lorsque $p[u]$ n'est pas l'élément neutre[35] de $\diamond$, le sommet w, qui inclut deux fois la valeur de u, n'aura pas un résultat conforme à la définition des réseaux associatifs.

35. Si toutes les données sont élément neutre $e_\diamond$ de $\diamond$, l'ensemble des données est réduit au singleton $e_\diamond$. Ce cas dégénéré ne présente pas d'intérêt.

(b) ⇒ (a) Si maintenant le réseau G est préfixe, alors tous les nœuds recevront par un seul chemin les valeurs de leurs antécédents. Ils incluront donc les valeurs une seule fois dans leur calcul, qui sera ainsi conforme à la définition des réseaux associatifs. □

Le nom de «graphe préfixe» vient du fait que toute `prefix-association` sera correctement calculée dans la Maille Associative. Nous donnons maintenant quelques résultats pour mieux connaître ces graphes. Donner une orientation à un arbre[36] signifie que l'on transforme les arêtes de l'arbre en arcs. Cette opération n'aboutit pas forcément à une arborescence[37] , car une arborescence suppose l'existence d'un sommet particulier antécédent de tous les autres : la racine. Or une orientation d'un arbre peut donner plusieurs racines.

36. Arbre : graphe non orienté sans circuit.

37. Arborescence : graphe orienté sans circuit, tel qu'il existe un, et un seul, chemin entre un sommet r et tout autre sommet v. r est appelé la racine.

Proposition 7 *Toute orientation d'un arbre donne un graphe préfixe ; la réciproque est fausse.*

Preuve. Dans un arbre, il y a une seule chaîne entre toute paire de sommets u et v. Cela donnera au plus un seul chemin de u à v, et aucun circuit lorsqu'on donnera une orientation quelconque à l'arbre. Pour montrer que la réciproque est fausse, il suffit de considérer le graphe préfixe constitué de deux arborescences identiques orientées des feuilles vers la racine et ayant les feuilles confondues (*cf.* figure 34). C'est un graphe préfixe mais il ne provient pas d'un arbre auquel on aurait donné une orientation. □

Proposition 8 *Un graphe préfixe G ayant une seule source s est une arborescence.*

Preuve. Supposons que le sommet v ne soit atteint par aucun chemin partant de s. Dans ce cas, puisqu'il n'y a pas de circuit dans le graphe préfixe, les plus lointains antécédents de v n'ont pas d'antécédents, ce qui contredit l'hypothèse. Le sommet s est donc une source principale[38]. Puisqu'il n'y a ni circuit ni double chemin, G est une arborescence enracinée en s. □

38. Source principale s de G : s est une source telle qu'il existe un chemin entre s et tout autre sommet de G.

Proposition 9 *Un graphe est préfixe si et seulement si il est constitué d'arborescences $T_1, \ldots, T_p$ interconnectées par des arcs, de telle façon que ces arcs de liaisons ne créent pas de chemins multiples entre une paire de sommets du graphe obtenu, ni de circuit.*

Preuve. Supposons que le graphe provenant de l'assemblage des arborescences présente un circuit ou un chemin multiple (*cf.* figure 35). Ce chemin multiple ou circuit n'est pas interne à l'une des arborescences, sinon elle n'en serait pas une. Ce chemin multiple ou circuit emprunte donc des arcs de liaisons inter-arborescences. Cela signifie qu'il y a des arcs de liaisons qui ont contribué à créer un chemin multiple ou un circuit entre certains sommets du graphe, ce qui est contraire à l'hypothèse.

Pour montrer la réciproque, il suffit d'exhiber un ensemble d'arborescences correctement interconnectées dans chaque graphe préfixe. En fait, toute décomposition en arborescences d'un graphe préfixe convient, et il suffit de considérer la décomposition en arborescences réduites à un seul sommet. □

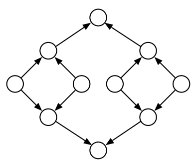


FIG. 34 – Graphe préfixe ne provenant pas de l'orientation d'un arbre.

Cette proposition nous procure un procédé pour obtenir tous les graphes préfixes. Ces graphes incluent les arborescences et les chemins ; ils généralisent le problème du préfixe (*cf.* § 1.1.3 page 7). Nous avons ainsi caractérisé les graphes sur lesquels les `prefix-associations` utilisant des opérateurs non-idempotents peuvent être mis en œuvre dans la Maille Associative.

Idempotence. La propriété d'idempotence seule ne résout pas le problème de la duplication des valeurs prises en compte dans les calculs. En effet, pour supprimer une des valeurs $p[a]$ dans l'expression $p[c] \diamond p[a] \diamond (p[b] \diamond p[a])$, il faut tout d'abord pouvoir supprimer les parenthèses — ce qu'autorise l'associativité — pour obtenir l'expression $p[c] \diamond p[a] \diamond p[b] \diamond p[a]$. Puis il faut permuter les valeurs — ce qu'autorise la commutativité — et placer les doublons en tête de l'expression pour obtenir $p[a] \diamond p[a] \diamond p[c] \diamond p[b]$. Enfin, l'idempotence peut être utilisée pour supprimer l'un des doublons. Si les propriétés d'associativité et de commutativité, spécifiées dans la définition des réseaux associatifs, sont suffisantes pour supprimer les doublons en utilisant l'idempotence, elles ne sont cependant pas nécessaires. Nous proposons des propriétés plus faibles (et plus générales) que celles-ci au chapitre 5, qui autorisent encore la suppression des doublons par l'idempotence.

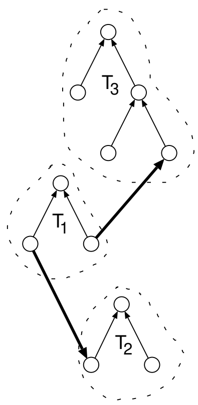


FIG. 35 – Connexions d'arborescence pour obtenir un graphe préfixe. Les arcs de liaisons sont en gras.

Conclusion. La mise en œuvre dans la Maille de la primitive `association` définie dans les réseaux associatifs n'est possible que si **(a)** l'opérateur est idempotent (en plus d'être associatif et commutatif, comme imposé par la définition des réseaux associatifs), ou si **(b)** le sous-réseau considéré est préfixe[(e)].

(e) Nous n'avons pas mentionné le fait que les opérateurs devaient avoir un élément neutre à droite. Cet élément est nécessaire lorsqu'un arc est masqué et ne fournit pas de valeur au calcul local (*cf.* § 3.2 page 35). Tous les opérateurs habituellement utilisés admettent un tel élément ; ceux que nous définissons au chapitre 5 aussi.

Ces problèmes n'étaient pas apparus jusqu'à présent parce que les `prefix-associations` utilisant des opérateurs non-idempotents sur des graphes sans circuit étaient en fait appliquées sur des arborescences (et avec l'addition). Nous verrons qu'il peut être intéressant d'appliquer des `associations` avec des opérateurs non idempotents sur des DAGs qui ne sont pas des arborescences (par exemple pour l'algorithme du tri topologique, *cf.* § 7.2.2 page 127).

3.1.3 Avantage du modèle d'exécution de la Maille Associative.

À ce stade de notre analyse, nous avons mis en avant des différences importantes entre le modèle théorique des réseaux associatifs et son implémentation dans la machine. Le modèle d'exécution de la Maille peut apparaître comme une implémentation incomplète du modèle des réseaux associatifs. Cependant, ce modèle d'exécution a des qualités intrinsèques, qui ne sont pas présentes dans le modèle des réseaux associatifs.

Opérateur non-idempotent sur un DAG. Dans la Maille Associative, une plus-`association` sur un DAG qui n'est pas un graphe préfixe est parfaitement définie, même si elle ne correspond pas à la définition du modèle de réseaux associatifs. Or cette primitive de calcul se révèle utile pour un grand nombre d'algorithmes. Elle correspond en fait à l'exécution d'un calcul sur son graphe de précédence[39 (f)].

39. Graphe de précédence d'un calcul : DAG représentant la suite logique des tâches à accomplir pour réaliser le calcul. Une tâche est antécédente d'une seconde si elle doit être achevée avant le début de la deuxième.

Par exemple, considérons l'algorithme du calcul des coefficients du binôme de Newton (figure 36). Le réseau est un DAG non préfixe (car il y a plusieurs chemins entre la source du DAG et certains sommets). Seul le nœud source est initialisé à 1 ; les autres sont initialisés à 0. Chaque nœud du réseau réalise l'addition des valeurs qu'il reçoit. Cela permet de calculer en un calcul global dans la Maille Associative les coefficients $C_n^p = n!/(p! \times (n-p)!)$. La figure 36 donne le résultat du calcul. La valeur du nœud source a été prise en compte plusieurs fois dans les calculs des nœuds, sinon le résultat du calcul serait 1 partout, et c'est ce que donne la définition de la plus-`prefix-association` sur ce réseau.

Le problème du *sac à dos*, implémenté à l'aide de la programmation dynamique, est un autre exemple (présentant plus d'applications) d'un calcul où il est important d'inclure plusieurs fois la valeur d'un nœud, sans pour autant que l'opérateur soit idempotent (*cf.* § 7.3.2 page 136).

Ces algorithmes ne peuvent être spécifiés à l'aide d'une seule `association` dans le modèle des réseaux associatifs.

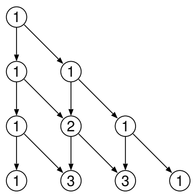


FIG. 36 – Calcul des coefficients du binôme de Newton.

Opérateurs non associatifs et non commutatifs. Nous avons expliqué au § 3.1.1 page 29 que l'associativité et la commutativité sont nécessaires au niveau du modèle, car la structure de données est un ensemble (peu) structuré par un graphe.

Un calcul global dans la Maille Associative est défini par les calculs locaux concurrents non synchronisés. Les nœuds qui réalisent ces calculs locaux n'ont pas une connaissance globale du sous-réseau, mais uniquement une connaissance de leurs antécédents directs. C'est là une différence essentielle avec la structure de données des réseaux associatifs. En effet, dans les deux cas — modèle des réseaux associatifs et modèle d'exécution de la Maille —, la structure de données est un graphe. Mais dans le modèle théorique, le graphe est connu par l'ensemble de tous les antécédents de chaque nœud, tandis que dans la Maille, le graphe est connu par l'ensemble des antécédents directs de chaque nœud.

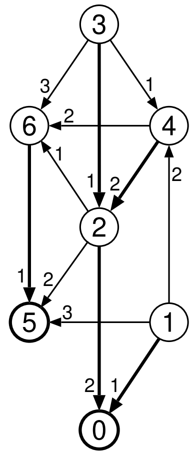


FIG. 37 – Ordre global sur un graphe à partir d'ordre locaux.

Le codage de la structure dans les réseaux associatifs conduit à plusieurs graphes possibles car la relation entre les ensembles des antécédents et le dessin du graphe n'est pas univoque. En effet, il suffit de considérer l'anneau à n sommets et le graphe complet à n sommets, pour lesquels les ensembles d'antécédents de chaque nœud sont tous égaux à l'ensemble de tous les sommets. Par contre, le codage de la structure par les masques dans la Maille est univoque et conduit à un seul graphe.

(f) Ce graphe est parfois appelé *graphe flot de données*. La distinction est cependant faite dans [Galilée et Roch, 1997] : le graphe flot de données représente les dépendances entre les données, tandis que celui de précédence représente celles entre les tâches.

Lorsque le graphe est unique, il est possible d'ordonner les arcs incidents à chaque sommet et donc de donner un ordre aux données incluses dans les calculs locaux. Mais si les sommets induisent chacun un ordre local, on peut alors définir un ordre global sur le réseau. En effet, partant d'un sommet donné (tel que celui qui a l'adresse la plus petite dans la Maille), on peut construire un parcours en profondeur orienté vers la racine (on remonte les arcs) en choisissant toujours, à partir du sommet courant, l'arc d'indice le plus petit parmi ceux qui conduisent à des sommets non encore visités. Par exemple, sur le graphe G de la figure 37 page précédente, tous les arcs incidents aux sommets ont une numérotation locale. On choisit le premier sommet du parcours avec le critère de l'adresse la plus petite. Ce sommet de départ porte le numéro 0 (il est représenté en gras pour indiquer que c'est une racine du parcours). On remonte les arcs en commençant toujours par celui d'indice le plus petit. Si des sommets ne sont pas atteints par ce premier arbre, on recommence à partir d'une nouvelle racine (ici 5), toujours choisie sur la base de l'adresse la plus petite de tous les sommets non encore atteints. C'est un parcours en profondeur sur le graphe inversé G^{-1}, obtenu en inversant le sens de tous les arcs.

Les données entrantes au niveau d'un nœud sont ordonnées par l'ordre des arcs entrants, aussi la commutativité de l'opérateur n'est plus nécessaire. De même, les sommets sont repérés les uns par rapport aux autres de façon univoque avec l'ordre global du graphe ; il est donc possible de prévoir dans quel ordre les calculs intermédiaires seront réalisés, et l'associativité n'est plus non plus nécessaire. Aussi la commutativité et l'associativité de l'opérateur, imposées par le modèle des réseaux associatifs, apparaissent inutiles dans le modèle d'exécution de la Maille Associative.

C'est en remettant en cause ces propriétés de l'opérateur utilisé lors d'un calcul global que nous définissons les r-opérateurs, au chapitre 5.

3.1.4 Conclusion

Cette étude a permis de cerner avec précision les équivalences entre le modèle et son implémentation dans la Maille. Nous avons ainsi défini les graphes préfixes, et mis en avant l'importance de l'idempotence.

Les réseaux associatifs sont incomplètement mis en œuvre dans la Maille Associative. À l'inverse, le modèle d'exécution de la machine possède des qualités de calcul intéressantes, qui ne sont pas présentes dans le modèle des réseaux associatifs. En outre, l'utilisation de la notion d'ensemble des antécédents est délicate, en l'absence d'informations sur le modèle d'exécution.

Ces constatations, ajoutées à celle de la difficulté d'implémenter exactement le modèle théorique (*cf.* § 3.3 page 40), nous conduisent à reformuler la définition des réseaux associatifs. Pour ce faire, nous considérons une nouvelle primitive : la `direct-association`.

3.2 Nouvelle définition

Nous définissons dans cette section un modèle de calcul proche dans son fonctionnement du modèle d'exécution de la Maille Associative, mais plus général. Ce modèle diffère du modèle des réseaux associatifs, essentiellement par la nouvelle primitive `direct-association`, aussi conservons-nous le nom de «réseau associatif».

Il est défini par un réseau d'entités calculantes dont les comportements ne sont pas coordonnés. Ce relâchement de contrainte quant au contrôle du réseau permet des implémentations diverses et intéressantes. Le modèle possède des qualités intéressantes pour la programmation. Dans la suite du document, sauf mention contraire, c'est cette définition que nous utiliserons.

3.2.1 Réseau

Nœud. Un *nœud*[g] est la plus petite entité produisant un calcul à partir de données reçues en entrée. La quantité de calcul réalisée par un nœud est déterminée comme étant suffisamment élémentaire pour qu'il n'y ait pas intérêt à la confier à plusieurs entités. Un nœud ne produit donc qu'un seul résultat à la fois, car la production de deux résultats différents pourrait être confiée à deux entités calculantes. Notons que le grain du calcul n'est pas précisé ; il est donné par la complexité de la donnée et de sa production (scalaire, liste, ensemble ou listes d'ensembles, selon les algorithmes, *cf.* chapitre 7).

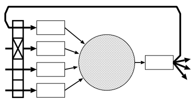


FIG. 38 – Nœud.

Un nœud v a une mémoire privée de taille finie. La case mémoire $b^0_{\text{in}}[v]$, contient, au début des calculs, la *valeur initiale* du nœud. Les cases $b^1_{\text{in}}[v]$ à $b^{\delta^-_v}_{\text{in}}[v]$ servent à recevoir les valeurs des antécédents[h] . On appelle *tampons entrants*[i] ces emplacements mémoires b_{in}. Suivant les implémentations, il s'agit d'entrées 1-bit de portes électroniques (*cf.* § 1.2.2 page 13), de ROM (pour b^0_{in}, *cf.* § 6.2 page 109), de registres (*cf.* § 4.3.3 page 74) ou de variables en RAM (*cf.* § 8.3.1 page 160), etc. Puisque l'entité ne produit qu'un seul résultat à la fois, elle dispose d'un seul *tampon de sortie*, noté $b_{\text{out}}[v]$. Sur la figure 38, les tampons sont représentés par les rectangles $\square$.

Liens. Le *réseau de base*, ou *graphe de base*, est constitué de l'ensemble des connexions entre les entités. Il est modélisé par un graphe orienté[j] $G_b(V, E)$, où V est l'ensemble des entités calculantes[k], et où un arc (u, v) de E représente une connexion depuis le tampon de sortie $b_{\text{out}}[u]$ du nœud u jusqu'à l'un des tampons d'entrée $b^i_{\text{in}}[v]$ $(1 \le i \le \delta^-_v)$ du nœud v. On note $\mathbf{i}_v(u)$ — ou simplement $\mathbf{i}(u)$ quand il n'y a pas d'ambiguïté possible — l'indice du tampon entrant de v sur lequel aboutit l'arc (u, v) : ce dernier relie donc les tampons $b_{\text{out}}[u]$ et $b^{\mathbf{i}_v(u)}_{\text{in}}[v]$. Un tampon entrant ne peut être l'extrémité finale que d'un seul lien. De plus, sur chaque entité u, il existe un lien, appelé *boucle*, depuis le tampon $b_{\text{out}}[u]$ vers le tampon $b^0_{\text{in}}[u]$; ce lien est interne au nœud. En particulier, il n'affecte pas son degré entrant δ^-_u.

Pvars et masques. Une *pvar* est définie comme au § 1.2.1 page 12 : c'est un $|V|$-uplet réparti à raison d'une valeur par nœud. Par exemple, les valeurs contenues dans les tampons b^0_{in} de chaque entité constituent la pvar b^0_{in}, et la valeur de cette pvar sur le nœud v est notée $b^0_{\text{in}}[v]$. Si les données sont des éléments de $\mathbb{S}$, alors une pvar p composée de ces éléments est un élément de $\mathbb{S}^{|V|}$. La *taille de la pvar* correspond au nombre de nœuds $|V|$ du réseau. La *largeur de la pvar* correspond à la taille d'une donnée de $\mathbb{S}$ (bit, mot, liste, etc.).

Un *masque* pour le nœud v est une donnée de $(\delta^-_v + 1)$ bits, qui servent à noter les connexions entrantes masquées[l] de l'entité v. Sur la figure 38, le masque est représenté devant les tampons entrants par un rectangle vertical ; s'il comporte une croix, le lien entrant est masqué. Une *pvar de masques*, ou **pvar**$\langle$ **masque** $\rangle$, est une pvar de mots de $(\delta^-_{G_b} + 1)$ bits[40] notée sous la forme net_G. Une pvar de masques définit un sous-réseau G dans le réseau de base G_b.

40. Dans le calcul du degré entrant δ^-_v d'un nœud v, et donc dans le calcul du degré maximal du graphe $\delta^-_{G_b}$, on ne prend pas en compte la boucle sur chaque nœud.

Dans certains cas, nous utilisons l'*hypothèse de l'index* définie comme suit :

Hypothèse 10 (Index) *Tous les nœuds ont un identifiant unique. Le $|V|$-uplet formé par ces identifiants donne la pvar index, notée $\imath$.*

Cette hypothèse est peu contraignante car, en pratique, les entités calculantes ont toujours un identifiant unique : dans une machine multi-processeur telle que la Maille, chaque processeur connaît

---

(g) Suivant les implémentations, un nœud est appelé PE, processus, processeur, entité calculante, etc.

(h) L'ordre des tampons entrants est important, car on ne suppose pas toujours la commutativité de l'opérateur utilisé dans le calcul local.

(i) Nous supposons que les cases mémoires et les tampons sont de taille finie, et implicitement adaptés aux données traitées ; nous ne parlerons pas du *type* de ces variables. Par ailleurs nous notons de la même façon le tampon et la valeur qu'il contient.

(j) En règle générale, nous supposons qu'il n'y a pas deux arcs différents pour relier un sommet à un autre. En effet, cette fonctionnalité est inutile avec les opérateurs que nous utilisons (*cf.* note **(a)** page 104). Aussi le réseau est modélisé par un *graphe simple*, n'ayant pas d'arcs multiples.

(k) Par la suite, on notera de la même façon un nœud de calcul et le sommet qui lui correspond dans le graphe. Cependant le terme «sommet» se réfère au graphe qui modélise le réseau, tandis que le terme «nœud» ou «entité» se réfère au réseau de calcul.

(l) On parle indifférement d'«arc masqué», ou de «tampon entrant masqué».

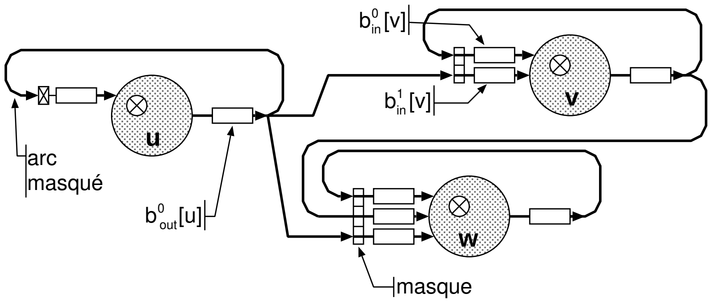


FIG. 39 – *Un réseau à trois nœuds, qui utilisent l'opérateur $\otimes$. L'arc (u, u) est masqué.*

son adresse qui est «codée en dur» dans le silicium de par sa position physique ; dans un système d'exploitation, le `pid`**41** détermine de façon unique les processus ; dans un réseau de communication tel qu'Ethernet, l'adresse de chaque processeur (plus précisément, de chaque carte réseau) est «codée en dur» sur la carte d'interface réseau. Quand l'hypothèse de l'index est utilisée, chaque nœud peut se référer à sa propre adresse par la variable `self`. On a : $\imath[\texttt{self}] = \texttt{self}$. On parle indifféremment de l'index, de l'indice ou de l'adresse d'un nœud.

41. *Processus Identifier* ou *Process Id.*

Rôle du réseau. Pour chaque lien (u, v) du réseau, si le masque de l'entité v indique qu'il n'est pas masqué, alors le réseau «recopie» les données du tampon de sortie $b_{\text{out}}[u]$ dans le tampon d'entrée $b_{\text{in}}^{\mathbf{i}_v(u)}[v]$. Les recopies ne sont pas synchronisées ; l'ordre dans lequel elles sont réalisées n'est pas précisé. En outre, le temps nécessaire à une recopie est fini mais non borné. Dans certains cas cependant, nous admettons l'*hypothèse du délai borné*, définie comme suit :

Hypothèse 11 (Délai borné) *Le délai séparant la date de production d'une donnée dans le tampon $b_{out}[u]$ et celle à laquelle cette donnée est disponible dans le tampon $b_{in}^{\mathbf{i}_v(u)}[v]$ est borné par une constante, que l'on note τ_r.*

Cette hypothèse est utile pour évaluer la complexité en temps de la primitive de calcul du modèle (*cf.* § 6.1.2 page 105). En pratique, quand le réseau est supposé ne pas avoir de pannes, cette hypothèse est souvent vérifiée.

Enfin, tout tampon d'entrée $b_{\text{in}}^{i}[v]$ $(1 \leq i \leq \delta_v^-)$, correspondant à une connexion masquée, est supposé contenir un *élément spécial*, qui est toujours le même durant toute la durée du calcul, et qui est le même dans tous les tampons d'entrée des arcs masqués. Nous précisons ci-après la nature de cet élément. Le tampon $b_{\text{in}}^{0}[v]$, s'il est masqué, contient toujours la valeur initiale du nœud.

3.2.2 Opérations

Calcul élémentaire et local. Un calcul élémentaire est défini comme au § 1.2.1 page 12 : il consiste en la modification d'une pvar sur tous les nœuds du réseau. Un *calcul local* sur le nœud u par l'opérateur binaire associatif à gauche $\diamond$ est défini comme la combinaison par $\diamond$ des valeurs contenues dans les tampons entrants de u et de la valeur courante du nœud contenue dans $b_{\text{in}}^{0}[u]$; le résultat est stocké dans le tampon de sortie $b_{\text{out}}[u]$:

$$b_{\text{out}}[u] \leftarrow b_{\text{in}}^{0}[u] \diamond b_{\text{in}}^{1}[u] \diamond \cdots \diamond b_{\text{in}}^{\delta_u^-}[u] \tag{3.1}$$

De même, lorsque l'opérateur est n-aire **(m)** , le calcul local est défini par :

$$b_{\text{out}}[u] \leftarrow \diamond\left(b_{\text{in}}^{0}[u], \ldots, b_{\text{in}}^{\delta_u^-}[u]\right) \tag{3.2}$$

Suivant les implémentations, le calcul local est «constament» effectué (circuit électronique tel que la Maille Associative, *cf.* § 1.2.2 page 13), ou effectué à l'arrivée d'une nouvelle valeur (système

(m) Le terme «n-aire» est générique pour désigner «à plusieurs variables». Ici, l'opérateur est $(\delta_v^- + 1)$-aire. Voir aussi la note **(c)** page 31.

distribué et actions gardées, *cf.* § 4.3.3 page 74), ou effectué périodiquement (bibliothèque de calculs, *cf.* chapitre 8).

Élément spécial. Tous les tampons entrant étant utilisés lors de chaque calcul, il est nécessire de disposer d'un élément spécial pour initialiser les tampons masqués (qui ne recoivent aucune valeur). Lorsque le calcul local fait intervenir un opérateur binaire, l'élément spécial placé dans les tampons d'entrée $b^i_{\text{in}}[u]$ $(1 \leq i \leq \delta^-_u)$ des arcs masqués sera l'élément neutre à droite[42] de l'opérateur $\diamond$ utilisé pour le calcul. Nous précisons la nature de l'élément spécial pour les opérateurs n-aires que nous utilisons au § 5.3 page 92.

42. Elément neutre à droite $e_\diamond$ **:** pour tout élément x de l'ensemble de définition de $\diamond$, on a $x \diamond e_\diamond = x$.

Calcul global : `direct-association`. Un calcul global par l'opérateur $\diamond$ consiste en l'exécution concurrente des calculs locaux par $\diamond$ sur tous les nœuds du réseau, ainsi qu'en la recopie par le réseau des valeurs produites dans les tampons b_{out} vers les tampons b_{in} correspondants. De plus, les nœuds et le réseau agissent sans coordination.

Ainsi, les calculs locaux ne sont pas synchronisés ; l'ordre des recopies par le réseau n'est pas spécifié, et l'imbrication des recopies et des calculs locaux non plus. Le calcul global progresse par l'exécution simultanée des calculs locaux et des recopies des valeurs des antécédents directs de chaque nœud, dans le sous-réseau défini par les masques. Suivant la terminologie des réseaux associatifs, nous notons cette primitive de calcul par

$$\diamond - \texttt{direct} - \texttt{association}(\mathit{net}_G, p)$$

où **(a)** le terme «direct» indique que ce sont les valeurs des antécédents directs qui sont prises en compte (et non celles de tous les antécédents comme pour l'`association`), **(b)** net_G désigne la pvar des masques et **(c)** p désigne la pvar des valeurs courantes, placées dans les tampons b^0_{in} au début du calcul global.

Notons que le mode de fonctionnement d'un réseau associatif est proche du modèle d'exécution *data-driven* (dirigé par les données) [Sansonnet, 1994]. Ces modèles supposent généralement que le réseau est acyclique. Ce n'est pas le cas ici.

Quand aucun calcul local ne peut produire de nouvelle valeur, et que le réseau ne peut copier ne nouvelles valeurs dans les tampons, le réseau est dit *stable*, et le calcul global est dit *terminé*. Le *résultat* de ce calcul est disponible dans les tampons b_{out}, qui forment la pvar résultat.

Lorsqu'une `direct-association` se termine, elle définit un opérateur de $|V|$ dans $|V|$ qui, à partir de la pvar b^0_{in}, a fabriqué la pvar b_{out}.

Notons que si, sur le nœud u, le lien interne qui relie le tampon de sortie $b_{\text{out}}[u]$ au tampon de la valeur initiale $b^0_{\text{in}}[u]$ est masqué, la valeur initiale ne sera pas modifiée par la `direct-association`. Dans le cas contraire, la valeur initiale sera égale à la valeur résultat ($b^0_{\text{in}}[u] = b_{\text{out}}[u]$).

Les `step-associations` et `step-exclusive-associations` (*cf.* 1.2.1 page 12) consistent à obtenir les valeurs des antécédents, puis à effectuer un seul calcul local (avec ou sans la valeur locale). Ces primitives ne posent pas de problèmes de mise en œuvre particuliers ; nous les conservons sans changement[(n)]. Nous n'utilisons pas le qualificatif `prefix` pour la `direct-association`, car ici il ne donnerait qu'une information sur l'opérateur (idempotence), ce qui serait redondant avec l'opérateur lui-même.

3.2.3 Utilisation

Style de programmation. Le mode de programmation d'un réseau associatif, tel que défini dans ce chapitre, est de type parallélisme de données. Le programme indique un flot séquentiel d'instructions. Parmi ces instructions, on trouve celles habituellement utilisées dans les langages à parallélisme de données pour les entrées-sorties, les déclarations de variables de contrôle (indices par exemple) et de pvars, ainsi que pour le contrôle du programme (conditionnelles, boucles...). L'instruction de contrôle

(n) Lorsqu'il n' y a pas d'ambiguïté, toutes ces primitives sont désignées par le terme générique `association`, qui inclut donc aussi la `direct-association`.

`Where( condition sur une pvar ) ElseWhere EndWhere` permet d'appliquer un traitement uniquement là où la condition sur la pvar est vérifiée.

Les instructions de calcul sont *data-parallèles* ; elles s'appliquent sur les pvars. Il s'agit des calculs élémentaires tels que $p = p + q$ (où p et q sont deux pvars), des `step-associations` (recopie + calcul local) et des `direct-associations`.

Ainsi, un algorithme écrit dans le paradigme des réseaux associatifs est une suite synchrone (séquentielle) de phases de calcul, qui sont asynchrones lorsqu'il s'agit de `direct-associations`. Des exemples sont donnés au chapitre 7 et un fragment de code pour la bibliothèque ANET se trouve en annexe, au § A.2 page 187.

Utilisation de l'index. Lorsque l'hypothèse de l'index (*cf.* page 36) est admise, il est possible d'utiliser des opérateurs différents d'un nœud à l'autre. Supposons que l'on veuille utiliser, sur chaque nœud u du réseau, l'opérateur $\diamond_u$. En utilisant la pvar index $\imath$, on peut construire l'opérateur global $\Diamond_G$ qui, sur chaque nœud u du réseau, est égal à l'opérateur $\diamond_u$ (o) :

$$\Diamond_G = \bigvee_{u \in V} (\imath == u) \wedge \diamond_u \tag{3.3}$$

Un tel opérateur, appelé *opérateur de réseau*, peut, par exemple, être codé par une construction `switch-case`. Ce formalisme est utile pour pouvoir noter $\Diamond_G$-`direct-association` un calcul global réalisé avec des opérateurs différents d'un nœud à l'autre.

Notons que la définition d'un opérateur de réseau n'est intéressante que dans le cadre d'une «utilisation SIMD ou SPMD» du modèle, où l'on souhaite distribuer la même instruction, ou le même programme, sur tous les nœuds du réseau. Cependant, dans un mode de programmation MPMD[43] où les programmes exécutés par les nœuds sont différents les uns des autres, l'index peut n'être utilisé qu'au moment des initialisations, pour donner à chaque nœud le programme qui lui revient. Il n'est dans ce cas pas utile de définir des opérateurs de réseau basés sur l'index.

43. *Multiple Program Multiple Data.*

Conditions sur l'opérateur et le réseau. Il existe des combinaisons de réseau G, d'opérateur $\diamond$ et de pvar p telles que la $\diamond$-`direct-association`(net_G, p) ne se termine pas. Il suffit de considérer l'opérateur d'addition sur un anneau initialisé à 1 partout.

De plus, de par l'exécution asynchrone, et sans coordination, du calcul global, il se peut que le résultat ne soit pas déterministe, c'est-à-dire qu'il ne soit pas le même d'une exécution à une autre.

Pour s'en convaincre, on peut considérer l'opérateur binaire défini sur $\mathbb{Q}^+ \cup \{+\infty\}$ par $(x, y) \mapsto \mathsf{max}(x, 1/x + y)$, et le réseau de la figure 40. Avant l'arrivée de 2 et de 3, le nœud c produit la valeur 1 car $\mathsf{max}(1, 1/1+0) = 1$. Si 2 arrive le premier, le nœud c produit la valeur 3 car $\mathsf{max}(1, 1/1+2) = 3$ et $\mathsf{max}(3, 1/3 + 2) = 3$. Puis, lorsque 3 arrive en c, le nœud c se stabilise sur le résultat 10/3 car $\mathsf{max}(3, 1/3 + 3) = 10/3$ et $\mathsf{max}(10/3, 3/10 + 3) = 10/3$. Si maintenant 3 arrive avant 2 en c, le nœud c se stabilise sur 4 car $\mathsf{max}(1, 1/1 + 3) = 4$ et $\mathsf{max}(4, 1/4 + 3) = 4$. L'arrivée de 2 n'y change rien car $\mathsf{max}(4, 1/4 + 2) = 4$. Le résultat final dépend donc ici de l'ordre d'arrivée des données, c'est-à-dire des temps d'acheminements des données dans le réseau.

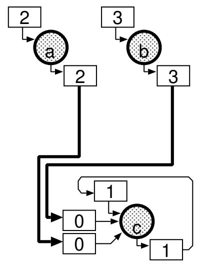


FIG. 40 – L'opérateur $(x, y) \mapsto \mathsf{max}(x, 1/x + y)$ ne conduit pas à un résultat déterministe sur ce réseau.

Enfin, étant donné un opérateur définissant un calcul local particulier, et qui conduit à un calcul global déterministe, il faut encore reconnaître le résultat obtenu au niveau du réseau ; il s'agit de donner une signification au calcul global défini localement au niveau du nœud. Ce problème ne peut être étudié dans le cas général, et nous le laissons à la responsabilité du programmeur (qui définirait un opérateur).

Il est donc nécessaire de rechercher des opérateurs (ou des réseaux) qui assurent à la fois la terminaison et le déterminisme d'une `direct-association` ; nous donnons maintenant une définition formelle à ce problème dans le cas général d'un opérateur de réseau (qui inclut celui d'un opérateur unique $\Diamond_G = \Diamond$).

(o) Par exemple, si le nœud 1 utilise l'opérateur min, et le nœud 2 l'opérateur max, alors $\Diamond_G = ((\imath == 1) \wedge \mathsf{min} \;\vee\; (\imath == 2) \wedge \mathsf{max})$.

Définition 12 *La* $\diamond_G$-`direct-association`(net_G, p) *se termine sur un résultat déterministe, noté par la pvar* q*, sur le sous-réseau* G *initialisé avec la pvar* p*, si : pour toute exécution* e *de ce calcul global sur le réseau, il existe une date* t_e *telle que, pour toute entité* u *du réseau, et pour tout délai positif* τ*, on a*

$$q[u] \quad = \quad (b_{out}[u])^{t_e} \quad = \quad (b_{out}[u])^{t_e+\tau} \tag{3.4}$$

où l'exposant indique la date à laquelle la valeur du tampon est considérée.

La $\diamond_G$-`direct-association`(net_G, p) *se termine sur tout réseau, initialisé avec toute pvar, si l'affirmation est vraie pour toute combinaison de réseaux et de pvars.*

3.3 Construction de l'`association`

Dans cette section, nous étudions la construction de la primitive `association` de la première définition des réseaux associatifs (*cf.* § 1.2.1 page 12), à l'aide de la primitive `direct-association`, définie dans la section précédente.

3.3.1 Présentation du problème

L'`association` est un calcul global, défini au niveau de l'ensemble des antécédents (*cf.* § 1.2.1 page 12). Cette primitive est incomplètement mise en œuvre dans la Maille Associative, comme expliqué au § 3.1.2 page 30. La primitive `direct-association` est une généralisation du calcul global tel qu'il est réalisé dans la Maille, à savoir par des calculs locaux concurrents. Mettre en œuvre l'`association` en n'utilisant que des `direct-associations` signifierait donc qu'il est possible de réaliser des `associations` dans la Maille Associative.

Indépendamment de ce problème d'implémentation dans la machine, il est intéressant de mieux connaître les difficultés de l'implémentation d'une primitive de haut niveau, définie avec des ensembles, à l'aide d'une primitive imposant peu de contraintes, et donc facilement mise en œuvre. De plus, notre modèle à base de `direct-associations` s'en trouverait enrichi.

3.3.2 s-opérateurs

Définition. La définition de la $\diamond$-`association` précise que l'opérateur $\diamond$ doit être associatif et commutatif (*cf.* § 1.2.1 page 12). Or la proposition 3 page 31 indique que, si l'opérateur n'est pas idempotent, le calcul d'une `direct-association` ne se terminera pas sur tous les graphes. Nous nous intéressons donc aux opérateurs associatifs, commutatifs et idempotents. En outre, l'existence d'un élément neutre est nécessaire pour avoir une valeur à mettre dans les tampons masqués, sans perturber pour autant le calcul (*cf.* § 3.2 page 35). Par la suite, tous les opérateurs sont considérés définis sur l'ensemble de données $\mathbb{S}$; toutes les données véhiculées dans le réseau et calculées par les entités appartiennent donc à cet ensemble.

Définition 13 *L'opérateur* $\oplus$*, défini sur l'ensemble de données* $\mathbb{S}$*, est un* s*-opérateur*[(p)] *s'il est associatif, commutatif et idempotent et possède un élément neutre* $e_\oplus$ *dans* $\mathbb{S}$*.*

On montre qu'un s-opérateur définit un ordre partiel[44] sur $\mathbb{S}$ (voir par exemple [Tel, 1991]).

44. Ordre partiel : relation binaire réflexive, anti-symétrique et transitive.

Théorème 14 *Si* $\oplus$ *est un* s*-opérateur sur* $\mathbb{S}$*, alors la relation binaire* $\preceq_\oplus$ *définie par* $(x \preceq_\oplus y) \equiv (x \oplus y = x)$ *est une relation d'ordre partiel sur* $\mathbb{S}$*. De plus, le plus petit élément d'un ensemble* $\{x_1, \ldots, x_n\}$ *(au sens de* $\preceq_\oplus$*), noté* $\perp_\oplus\{x_1, \ldots, x_n\}$*, est obtenu par* $x_1 \oplus \cdots \oplus x_n$*.*

(p) Dans [Tel, 1991] et [Tel, 1994], un tel opérateur est appelé *infimum*. En algèbre, il est appelé *loi d'un demi-groupe abélien idempotent* (*cf.* § 4.1.2 page 54). Le terme s-opérateur vient de sort, car un tel opérateur peut être utilisé pour ordonner les éléments de $\mathbb{S}$ (*cf.* théorème 14).

Preuve. Nous devons monter que $\preceq_\oplus$ est une relation binaire transitive, antisymétrique et réflexive. Supposons que $x \preceq_\oplus y$ et $y \preceq_\oplus z$. Dans ce cas, $x \oplus y = x$ et $y \oplus z = y$. D'où $x \oplus z = (x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y = x$ et $\preceq_\oplus$ est transitive. Supposons que $x \preceq_\oplus y$ et $y \preceq_\oplus x$. Dans ce cas, $x \oplus y = x$ et $y \oplus x = y$, *i.e.*, $x = y$ et $\preceq_\oplus$ est antisymétrique. Puisque $x \oplus x = x$, nous avons $x \preceq_\oplus x$ et $\preceq_\oplus$ est réflexive. Finalement, $(x \oplus y) \oplus x = x \oplus (x \oplus y) = x \oplus y$ et $(x \oplus y) \oplus y = x \oplus (y \oplus y) = x \oplus y$. D'où $(x \oplus y) \preceq_\oplus x$ et $(x \oplus y) \preceq_\oplus y$. Maintenant, considérons $z \in \mathbb{S}$ tel que $z \preceq_\oplus x$ et $z \preceq_\oplus y$. Dans ce cas, $z \oplus x = z$ et $z \oplus y = z$. Donc $z \oplus (x \oplus y) = (z \oplus x) \oplus y = z \oplus y = z$. D'où $z \preceq_\oplus (x \oplus y)$ et $\perp_\oplus\{x, y\} = x \oplus y$. Par récurrence, nous étendons facilement ce résultat à un ensemble quelconque $\{x_1, \ldots, x_n\}$. $\square$

Terminaison déterministe. L'ordre partiel défini par un s-opérateur nous sert à montrer la terminaison de la `direct-association` utilisant un tel opérateur. Une preuve matricielle de ce résultat est donnée au § 4.2.4 page 68, et une preuve par réduction d'expressions est donnée au § 5.1.3 page 80. Nous faisons également le lien entre ce résultat et les travaux de Tel au § 6.1.4 page 107.

Théorème 15 *Si $\oplus$ est un* s*-opérateur sur $\mathbb{S}$, alors toute $\oplus$-`direct-association`(net_G, p) se termine sur un résultat déterministe sur tout réseau G, initialisé avec toute pvar p de $\mathbb{S}^{|V|}$. De plus, le résultat sur le nœud v du réseau est $\perp_\oplus\{p[u], u \in \Gamma^-_G(v)\}$.*

Preuve. Un calcul local sur un nœud u est défini par $b_{\text{out}}[u] \leftarrow b^0_{\text{in}}[u] \oplus b^1_{\text{in}}[u] \oplus \cdots \oplus b^{\delta^-_u}_{\text{in}}[u]$ (*cf.* équation 3.1 page 37). Suivant le théorème 14, on a donc $b_{\text{out}}[u] = \perp_\oplus\{b^0_{\text{in}}[u], \ldots, b^{\delta^-_u}_{\text{in}}[u]\}$, soit $b_{\text{out}}[u] \preceq_\oplus b^i_{\text{in}}[u]$, pour tout i, $0 \leq i \leq \delta^-_u$. Les valeurs successives contenues dans les tampons sont donc décroissantes (au sens de $\preceq_\oplus$). Or le réseau est alimenté avec un nombre fini de valeurs initiales. Chaque nœud v finira donc avec la plus petite valeur qu'il lui est possible d'obtenir, c'est-à-dire $\perp_\oplus\{p[u], u \in \Gamma^{-1}_G(v)\}$. $\square$

Nous venons de montrer que le résultat d'une $\oplus$-`direct-association` est exactement celui d'une $\oplus$-`association` lorsque l'opérateur $\oplus$ est un s-opérateur. Les opérateurs min, max, and, or, PPCM, PGCD, $\cup$, $\cap$ sont des exemples de s-opérateurs. Le théorème 15 justifie donc l'emploi des `associations` avec les opérateurs max, min, and et or dans la Maille Associative. Ce résultat vient en complément de la proposition 6 page 32 qui justifie l'usage des plus-`associations` sur les arborescences (qui sont des graphes préfixes). Les algorithmes écrits dans le modèle des réseaux associatifs jusqu'à présent, qui se contentaient de ces opérations, ont ainsi une preuve formelle de leur bonne implémentation dans la Maille.

3.3.3 Opérateur non-idempotent sur une composante fortement connexe

Une `association` sur un sous-réseau fortement connexe (figure 41 **a**), avec un opérateur non-idempotent, ne peut être implémentée directement dans la Maille, ou avec une `direct-association` (proposition 6 page 32). Cependant, le résultat est, par définition, identique sur tous les nœuds de la composante fortement connexe (puisqu'ils ont tous les mêmes ensembles d'antécédants). On peut donc calculer ce résultat sur un nœud particulier, puis le diffuser à toute la composante, ce qui procure un procédé de calcul assez simple.

On construit tout d'abord une arborescence couvrante de la composante fortement connexe (figure 41 **b**), que l'on oriente vers la racine (figure 41 **c**). Dans la Maille Associative ou dans les bibliothèques de programmation, on utilise la primitive `spanning-tree` pour cela. Au chapitre 7, nous donnons plusieurs algorithmes qui permettent d'obtenir facilement une telle arborescence, indépendamment de cette primitive.

L'arborescence couvrante est un graphe préfixe, et on peut appliquer la `direct-association` avec l'opérateur non-idempotent (*cf.* proposition 6 page 32 et figure 41 **c**). Le résultat de l'`association` est obtenu sur la racine. Il ne reste plus qu'à le diffuser avec une max-`direct-association` (par exemple, ici pour des valeurs entières), en ayant mis toutes les valeurs des nœuds qui ne sont pas la racine à 0 (figure 41 **d** puis **e**).

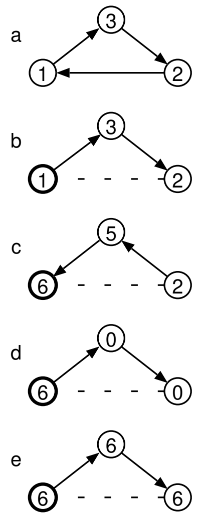


FIG. 41 – plus-`association` sur une composante fortement connexe (CFC).

Si le sous-réseau se présente sous forme de plusieurs composantes fortement connexes indépendantes, on pourra appliquer la même technique en parallèle sur chacune d'elles.

3.3.4 Opérateur non-idempotent sur un graphe non-préfixe

Il nous reste le cas des `associations` utilisant des opérateurs $\oplus$ non-idempotents sur un sous-réseau G non-préfixe, qui n'est pas formé de composantes fortement connexes indépendantes **(q)** (figures 42 et 43).

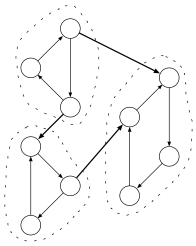


FIG. 42 – Premier DAG de CFCs G_1.

Méthode. Suivant la proposition 6 page 32, la seule solution pour mettre en œuvre ce calcul global est de considérer des sous-réseaux préfixes. Nous essayons donc ici de calculer l'`association` à l'aide de `direct-associations` appliquées sur des sous-réseaux préfixes de G. Pour supprimer les circuits, nous nous ramenons au graphe réduit $\widetilde{G}$ de G (voir ci-après). Pour supprimer les doubles chemins, nous recherchons le squelette $S(\widetilde{G})$ du graphe réduit. Mais il faut que le graphe obtenu après ces modifications, définisse le même ensemble d'antécédents que celui de départ.

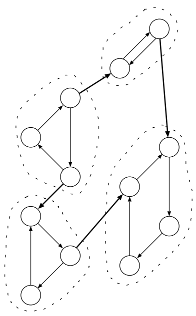


FIG. 43 – Second DAG de CFCs G_2.

Développement. Les sous-réseaux de G que nous devons considérer doivent définir les mêmes ensembles d'antécédents que G lui-même, sinon il ne sera pas possible de retrouver le résultat de l'`association` en appliquant des `direct-associations`.

Définition 16 *Le graphe $H(V_H, E_H)$ est un graphe représentatif des antécédents du graphe $G(V_G, E_G)$ si et seulement si $V_H = V_G$ et, pour tout chemin de u à v dans G, il existe un chemin de u à v dans H.*

Ainsi, si H est un graphe représentatif des antécédents de G, les deux graphes G et H définissent les mêmes ensembles d'antécédents. Nous recherchons un graphe représentatif des antécédents du réseau G qui soit également préfixe.

Puisqu'il s'agit de supprimer des chemins multiples ou des circuits, nous recherchons plutôt ce graphe préfixe parmi les sous-graphes **(r)** de G qui sont représentatifs des antécédents de G. Partant du graphe G, on obtient, en supprimant certains arcs dans G, un sous-graphe $G' \subset G$ qui est représentatif des antécédents de G. À partir de G', on obtiendra G'' en enlevant des arcs à G', tout en conservant les mêmes ensembles d'antécédents, et ainsi de suite. Il nous faut rechercher le sous-graphe représentatif des antécédents qui soit minimal, *i.e.*, qui ne contient plus de chemin multiple.

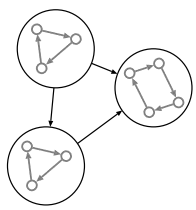


FIG. 44 – Graphe réduit du premier DAG de CFCs G_1.

On montre que l'ensemble de ces sous-graphes représentatifs des antécédents admet un minimum (au sens de la relation d'ordre d'inclusion des arcs) si et seulement si le graphe G est sans circuit [Xuong, 1992]. On se ramène donc au graphe réduit $\widetilde{G}$ de G.

Définition 17 *Le graphe réduit de G, noté $\widetilde{G}$, aussi appelé graphe quotient, est le graphe obtenu à partir de G en concentrant en un seul sommet chaque composante fortement connexe de G (cf. figures 44 et 45). Si u et v sont des sommets de G n'appartenant pas à la même CFC, alors l'arc $(\widetilde{u}, \widetilde{v})$ existe dans $\widetilde{G}$ si et seulement si l'arc (u, v) existe dans G ($\widetilde{w}$ désigne la CFC d'un sommet w de G).*

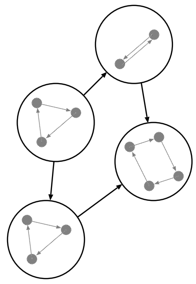


FIG. 45 – Graphe réduit du second DAG de CFCs G_2.

Au § 3.3.3 page précédente, nous donnons une technique simple pour implémenter une `association` à l'aide de `direct-associations` sur une composante fortement connexe. Aussi, le passage au graphe réduit ne pose pas de problème majeur. Le graphe réduit de G est sans circuit, et dans ce graphe l'ensemble des graphes partiels représentatifs des antécédents admet un minimum.

Définition 18 *Le minimum, au sens de la relation d'ordre de l'inclusion des arcs, de tous les sous-graphes du graphe sans circuit $\widetilde{G}$, qui sont représentatifs des antécédents de $\widetilde{G}$, s'appelle squelette de $\widetilde{G}$, ou bien diagramme de Hasse de la relation d'ordre définie par $\widetilde{G}$. Il est noté $S(\widetilde{G})$ (cf. figures 46 page suivante et 47 page ci-contre).*

(q) Nous appelons un tel graphe *DAG de CFCs* pour graphe orienté sans circuit reliant des composantes fortement connexes.

(r) Suivant la note de bas de page **(f)** page 12, nous appelons sous-graphe de G (ou sous-réseau) un graphe obtenu en enlevant des arcs à G ; un tel graphe est appelé *graphe partiel* par exemple dans [de Rumeur, 1994].

Par définition, le squelette $S(\widetilde{G})$ est représentatif des antécédents de $\widetilde{G}$. Le squelette d'un graphe sans circuit s'obtient en supprimant tous les arcs de transitivité de la relation d'ordre définie par le graphe.

Définition 19 *Un arc (u, w) de G est de transitivité s'il existe, dans le graphe, un sommet v différent de u et de w, tel qu'il existe un chemin de u à w passant par v (cf. figure 48).*

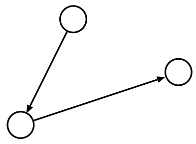


FIG. 46 – Squelette de G_1.

Proposition 20 *Un graphe préfixe ne possède pas d'arcs de transitivité ; il est égal à son squelette.*

Preuve. Soit (u, w) une arc de transitivité du graphe préfixe H. Par définition, il existe un sommet v de H tel qu'il y a un chemin de u à w dans H passant par v (figure 48). Ce chemin double l'arc (u, w) ; H n'est donc pas préfixe. □

Les arcs de transitivité créent des doubles chemins. En les supprimant dans $\widetilde{G}$, on peut espérer obtenir un squelette $S(\widetilde{G})$ qui soit préfixe. En fait, le squelette $S(\widetilde{G})$ est le seul sous-graphe représentatif des antécédents de $\widetilde{G}$ qui soit aussi préfixe.

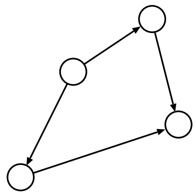


FIG. 47 – Squelette de G_2.

Proposition 21 *Soit H un sous-graphe du graphe sans circuit $\widetilde{G}$, représentatif des antécédents de $\widetilde{G}$. Alors, si H est un graphe préfixe, H est le squelette de $\widetilde{G}$.*

Preuve. Soit H un sous-graphe de $\widetilde{G}$, représentatif de ses antécédents et préfixe. Supposons que H ne soit pas le squelette de $\widetilde{G}$. Alors il existe un sous-graphe H' de H, différent de H (c'est-à-dire qui a moins d'arcs que H) tel que H' soit le squelette de $\widetilde{G}$. Mais dans ce cas, H' est également le squelette de H car H' conserve les antécédents de $\widetilde{G}$, et H a les mêmes ensembles d'antécédents que $\widetilde{G}$. Puisque H' est le squelette de H et que H' est différent de H, il existe un arc de transitivité dans H, qui n'est donc pas préfixe (proposition 20). □

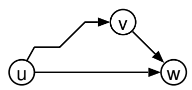


FIG. 48 – Arc de transitivité (u, w).

Cependant, le squelette $S(\widetilde{G})$ n'est pas toujours un graphe préfixe. Pour qu'il en soit un, il faut que les chemins multiples de $\widetilde{G}$ soient tous dus à des arcs de transitivité.

Proposition 22 *Le squelette de $\widetilde{G}$ est un graphe préfixe si et seulement si les chemins multiples de $\widetilde{G}$ sont dus à des arcs de transitivité.*

Preuve. Si les chemins multiples de $\widetilde{G}$ sont dus à des arcs de transitivité, alors ils n'apparaissent plus dans le squelette, obtenu en supprimant ces arcs. Réciproquement, un chemin multiple qui n'est pas dû à un arc de transitivité ne sera pas supprimé lorsqu'on enlèvera ces arcs, et il apparaîtra encore dans le squelette. □

Le chemin multiple de $\widetilde{G_1}$ (figure 44) est dû à un arc de transitivité. En le supprimant, on obtient un squelette $S(\widetilde{G_1})$ qui est un graphe préfixe (figure 46). Par contre, $\widetilde{G_2}$ présente un chemin multiple qui n'est pas dû à un arc de transitivité (figure 45) ; dans ce cas, le squelette $S(\widetilde{G_2})$ n'est pas préfixe (figure 47).

Conclusion. Partant d'un sous-réseau modélisé par le graphe G, on obtient son graphe réduit $\widetilde{G}$ en concentrant en un seul sommet ses composantes fortement connexes (définition 17), ce qui revient à passer au graphe virtuel (*cf.* § 2.2.1 page 22). Le calcul sur ces composantes fortement connexes ne pose pas de problème (§ 3.3.3 page 41). À partir du graphe réduit $\widetilde{G}$ qui n'a plus de circuit, on cherche un sous-graphe qui soit représentatif des antécédents de $\widetilde{G}$ (définition 17) sans chemin multiple, de telle sorte qu'il soit préfixe (proposition 5), ce qui permet de calculer l'`association` avec une `direct-association` (proposition 6). Pour ce faire, on supprime un à un les arcs de transitivité (définition 19) qui engendrent des doubles chemins, et dont la suppression ne modifie pas les ensembles des antécédents. On arrive ainsi au squelette $S(\widetilde{G})$ de $\widetilde{G}$ (définition 18), qui est le seul sous-graphe de $\widetilde{G}$ qui soit représentatif des antécédents et qui soit préfixe, s'il en existe un (proposition 21). Néanmoins le squelette $S(\widetilde{G})$ n'est pas toujours un graphe préfixe (proposition 22).

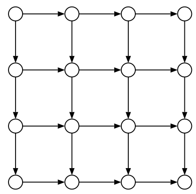


FIG. 49 – Exemple de graphe où la mise en œuvre de l'$\diamond$-`association` par des calculs locaux concurrents nécessite trop de contrôle pour être envisagée.

Nous avons ainsi un procédé pour calculer une `association` avec un opérateur non-idempotent, sur un DAG de CFCs. Mais il ne fonctionne que si le réseau est un DAG de CFCs dont le graphe réduit ne présente que des doubles chemins dus à des arcs de transitivité. Les autres réseaux nécessitent de découper le calcul global en plusieurs phases, qui peuvent être extrêmement nombreuses (*cf.* figure 49 page précédente). Dans ce cas, le calcul de l'`association` mis en œuvre avec des calculs locaux concurrents (comme dans la Maille Associative ou avec la `direct-association`) est particulièrement inefficace, et nous ne l'envisagerons pas.

3.3.5 Bilan

Il n'est pas toujours possible de mettre en œuvre l'`association` à l'aide de `direct-associations`. La table de la figure 50 récapitule les résultats de cette section[s]. Cette étude illustre la difficulté de mettre en œuvre une primitive de calcul manipulant des ensembles (au sens mathématique) dans un modèle d'exécution peu contraint.

	Graphe préfixe	**Composante fortement connexe (CFC)**	**DAGs de CFCs,** tels que $\widetilde{G}$ a des chemins multiples dus à des arcs de transitivités uniquement	**Autres DAGs de CFCs**
Opérateur associatif, commutatif et idempotent (s-opérateur)	pas de précaution particulière (§ 3.1.2 page 30 ou § 3.3.2 page 40)	pas de précaution particulière (§ 3.3.2 page 40)	pas de précaution particulière (§ 3.3.2 page 40)	pas de précaution particulière (§ 3.3.2 page 40)
Opérateur associatif, commutatif mais non idempotent	pas de précaution particulière (§ 3.1.2 page 30)	sur une arborescence couvrante orientée vers la racine, puis diffusion sur la CFC (§ 3.3.3 page 41)	sur les CFCs isolées, puis sur le squelette du graphe réduit $S(\widetilde{G})$, puis diffusion sur les CFCs (§ 3.3.4 page 42)	impossible (§ 3.3.4 page 42)

FIG. 50 – *Construction de l'`association` à l'aide de `direct-associations`.*

3.4 Conclusion

3.4.1 Résumé

Au § 3.1, nous avons analysé le modèle des réseaux associatifs tel que défini dans le premier chapitre, ainsi que sa mise en œuvre dans la Maille Associative. Cette analyse nous a conduit à redéfinir le modèle § 3.2. Cette redéfinition se traduit essentiellement par l'introduction d'une nouvelle primitive de calcul : la `direct-association`. Au § 3.3, nous avons étudié la mise en œuvre

(s) Nota : comme indiqué au 3.2.2, tous les opérateurs doivent aussi avoir un élément neutre pour autoriser les tampons masqués.

de l'association à partir de la direct-association. Cette étude montre la difficulté de l'implémentation complète de l'association dans la Maille Associative.

Ainsi, la direct-association est une primitive de bas niveau ; elle change la philosophie d'utilisation du modèle des réseaux associatifs. Nous présentons tout d'abord les avantages de la redéfinition du modèle, avant d'évoquer la nouvelle problématique qui s'en dégage.

3.4.2 Avantages de la redéfinition

Mise en œuvre. Le nœud fournit un seul résultat. Il s'agit de la plus petite entité de calcul imaginable. Il offre donc un parallélisme potentiel maximal. De même, le contrôle de l'exécution d'une direct-association est minimal car ni les calculs, ni les communications, ni les imbrications des calculs et des communications, n'ont besoin d'être supervisés. Cette hypothèse d'absence de coordination calcul-communication est très faible. A contrario, nous avons vu au § 3.3 que la mise en œuvre d'une association dans la Maille Associative nécessitait parfois plusieurs étapes intermédiaires (arborescence couvrante à plonger, recherche du graphe réduit, puis du squelette, etc.), ce qui implique un contrôle plus important.

La primitive direct-association est de bas niveau et peut être mise en œuvre efficacement ; elle correspond au calcul global effectué dans la Maille Associative. Ainsi, le problème de la mise en œuvre est d'emblée résolu, ce qui n'était pas le cas avec l'association (*cf.* § 3.1.2 et § 3.3). Nous étudions une mise en œuvre de la primitive dans un contexte de calculs distribués au § 4.3.3 page 74 et § 6.1.4 page 107, dans la Maille Associative au § 8.1.1 page 153 et dans une bibliothèque de programmation parallèle au chapitre 8. Dans ce contexte, la direct-association est une primitive data-parallèle offrant un grain de calcul plus conséquent.

Algorithmes. La direct-association permet d'exécuter en une seule instruction les algorithmes exprimés sur leur graphe de précédence (*cf.* § 3.1.3). Nous avons présenté le calcul des coefficients du binôme de Newton, mais d'autres algorithmes peuvent être réalisés de cette façon (*cf.* § 7.3 page 136).

En ce qui concerne la construction de la primitive association, les mêmes restrictions que celles dues à la Maille sont inhérentes à la direct-association. Nous avons cependant élargi la famille des graphes qui permettaient de mettre en œuvre l'association à l'aide de direct-associations (*cf.* § 3.1.2).

La nouvelle définition du modèle autorise l'utilisation des opérateurs n-aires et de réseau, ce qui ouvre la voie à de nouvelles possibilités algorithmiques. Elle permet aussi d'utiliser l'asynchronisme dans la programmation (*cf.* chapitre 7).

Structure de données. La primitive direct-association est définie à partir des antécédents directs des nœuds, ce qui ne donne qu'un seul dessin du réseau sous-jacent, contrairement aux ensembles des antécédents (directs ou non) utilisés pour l'association. Cette structure est riche en applications, comme l'indiquent les utilisations de l'association sur un modèle d'exécution fixé (tel que celui de la Maille, au chapitre 2), et les algorithmes du chapitre 7.

Puisque la primitive n'est pas définie à partir d'un ensemble, les opérateurs n'ont pas besoin d'être associatifs ni commutatifs, comme c'était le cas avec l'association (*cf.* § 3.1.1). Cela nous permet de définir au chapitre 5 des opérateurs non-associatifs et non-commutatifs (les r-opérateurs), qui offrent une meilleure exploitation de la structure du réseau. Quand ils sont idempotents, ils peuvent être utilisés sur des réseaux présentant des circuits, et peuvent donner plusieurs résultats identiques par composante fortement connexe (contrairement aux s-opérateurs).

Comme dans la première définition, les masques permettent de considérer tout sous-réseau du réseau de base. Cela autorise le codage d'objets très irréguliers, tels que les contours d'une région dans une image, ou la région tout entière, ou des arbres, etc. En utilisant le principe du graphe virtuel (*cf.* § 2.2.1 page 22), on peut utiliser toute sorte de topologies dans le réseau de base en forme de maille ; la seule restriction est la taille du réseau de base (*cf.* annexe C page 195).

3.4.3 Nouvelle problématique

Nouveau modèle. Partant d'un modèle de calcul de haut niveau — les réseaux associatifs définis au § 1.2.1 page 12 — et de ses difficultés de mise en œuvre, nous avons défini une nouvelle primitive de calcul : la `direct-association`. Ce faisant, le modèle de calcul ainsi complété est, en pratique, profondément modifié, car son implémentation et son utilisation sont maintenant très différentes. À la place d'une opération sur des ensembles structurés par un graphe, on utilise maintenant des calculs sur un réseau défini au niveau du nœud. On passe de la définition macroscopique d'un calcul global — l'`association` — à une vision microscopique : la `direct-association`. Cette primitive définit un calcul global à partir de calculs locaux très simples, répétés jusqu'à stabilisation.

Le modèle de calcul est à parallélisme de données, mais autorise des objets codés par des graphes quelconques et gère donc l'irrégularité. La `direct-association` est un opérateur data-parallèle asynchrone. Un calcul n'est pas spécifié par son résultat, mais par le calcul local qui, une fois appliqué sur chaque nœud et jusqu'à stabilisation, donne le résultat. Ces nuances tendent à rapprocher le modèle de certains aspects des systèmes distribués, d'autant que rien n'interdit, dans sa définition, de le mettre en œuvre dans un contexte SPMD, voire distribué, plutôt que SIMD. Le modèle des réseaux associatifs est donc hybride, entre parallélisme de données et calculs distribués, entre flot d'instructions synchrone, et asynchrone, entre SIMD et SPMD.

Nouveaux problèmes. Ce modèle est de «bas niveau» sur une échelle d'abstraction des modèles (figure 51 page suivante). Il est ainsi mis en œuvre sans trop de difficulté (ce qui ne veut pas dire efficacité), et la Maille Associative, de par son modèle d'exécution semblable à la `direct-association`, en est une illustration. Mais plusieurs questions restent, à ce stade, en suspens.

Tandis que l'`association` est toujours définie, la `direct-association` n'est définie que si elle se termine ; le problème des conditions de terminaison de cette primitive n'est pas trivial, étant donné sa nature asynchrone. L'extension des propriétés des opérateurs admises pour cette primitive — la commutativité et l'associativité ne sont plus nécessaires — ne doit pas cacher le problème central de la terminaison : quelles sont les propriétés des opérateurs qui assurent la terminaison de la `direct-association` ? Nous traitons ce problème au chapitre 5.

Afin d'utiliser le modèle sur d'autres plates-formes que la Maille Associative, il nous faut étudier le contrôle d'un calcul dans un réseau associatif, pour nous affranchir du contrôleur de la machine. Nous abordons ce problème au chapitre 6. Dans ce chapitre, nous étudions aussi les propriétés de sûreté du modèle (perte de messages, duplication, pannes).

Puisque le modèle est défini à partir d'un réseau de base, il est nécessaire d'étudier au moins une topologie adéquate, assurant des propriétés minimales au modèle. À l'annexe C, nous montrons que le graphe maille est une topologie dont le modèle s'accommode très bien.

Si les modèles de haut niveau sont difficiles à mettre en œuvre, ils présentent cependant des propriétés intéressantes pour l'écriture d'algorithmes. Au chapitre 2, nous avons conclu sur la bonne adéquation des réseaux associatifs et de l'`association` au traitement d'images. Il ne faudrait pas que la nouvelle définition et la `direct-association` perdent cet attrait en gagnant celui d'une mise en œuvre aisée. Nous étudions l'algorithmique de la `direct-association` au chapitre 7.

Pour finir, au chapitre 8, nous montrons, en développant la bibliothèque ANET, que le modèle ne reste pas cantonné en une étude théorique ; il constitue une solution intéressante à la programmation data-parallèle irrégulière.

Nous commençons l'étude du modèle par la recherche des modélisations voisines de la `direct-association`, au chapitre suivant.

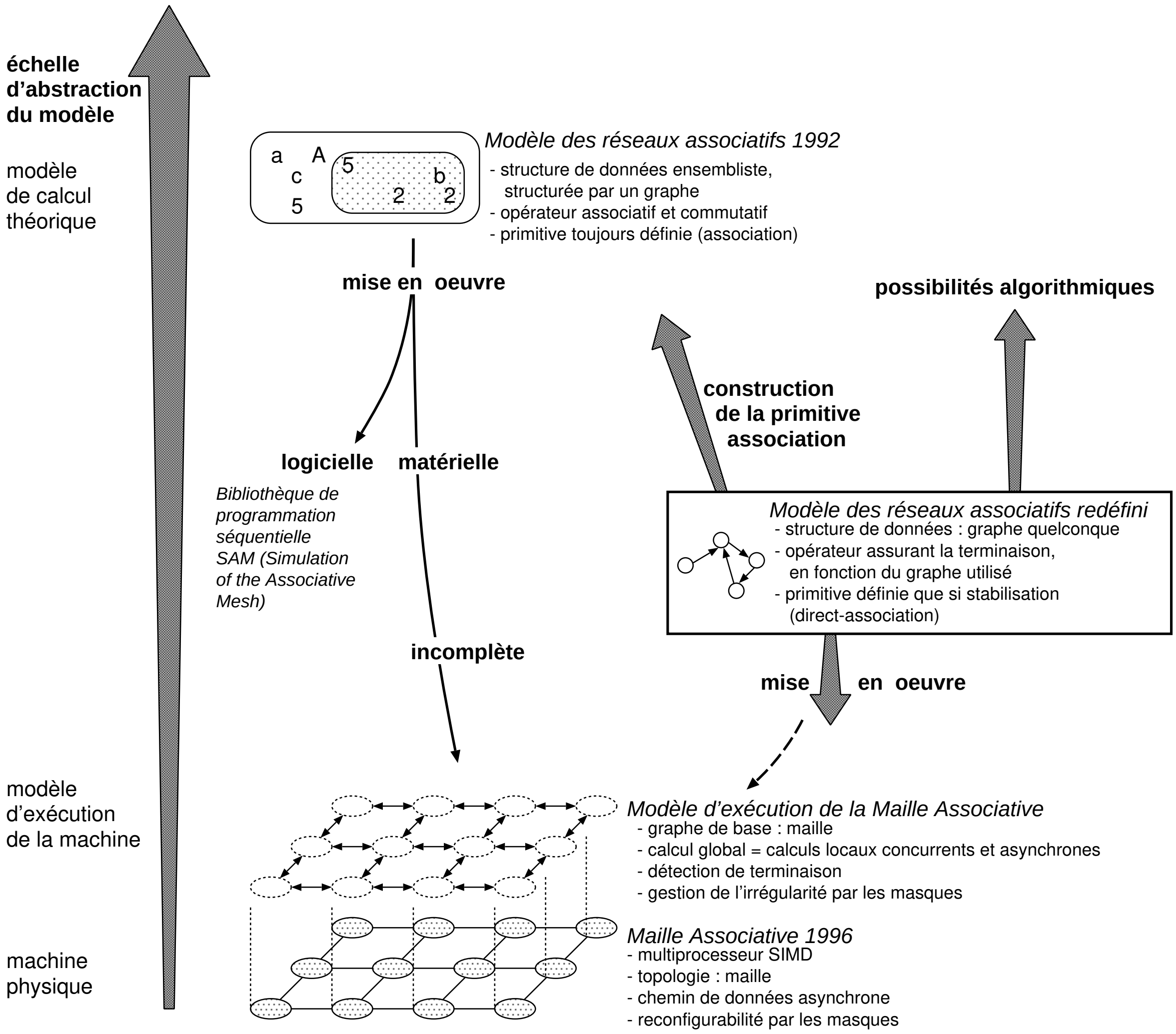


FIG. 51 – *Hiérarchie des modèles et problèmes à résoudre.*

Deuxième partie

Étude du Modèle

Chapitre 4

Modélisation

Tel que nous l'avons défini au chapitre précédent, un réseau associatif se présente sous la forme d'un ensemble d'entités calculantes, et communicantes grâce à des liens de communications. Un tel «objet informatique» peut être étudié sous divers modèles formels. Nous présentons certains d'entre eux dans ce chapitre, consacré à l'étude de la modélisation des réseaux associatifs.

Nous commençons au § 4.1 par présenter l'algèbre max-plus, qui peut modéliser, sous certaines réserves quant à la synchronisation, les réseaux associatifs sous un angle macroscopique. Cette section introduit des rappels sur les structures algébriques et sur l'algèbre de chemins qui nous servent par la suite. Au § 4.2, nous abordons les itérations asynchrones qui permettent de s'affranchir de ces contraintes de synchronisation, et nous obtenons une vision macroscopique du calcul dans un réseau associatif.

Enfin, nous développons au § 4.3 une modélisation par les systèmes distribués des réseaux associatifs. Cette modélisation nous est utile pour étudier certaines propriétés liées au contrôle du réseau, au chapitre 6.

Nous pourrons alors conclure sur les apports de ces modélisations, tant pour l'étude du contrôle des réseaux associatifs que pour celle de ses aspects calculatoires, ainsi que sur les liens existants entre les domaines évoqués.

4.1 Algèbre max-plus

Nous avons expliqué dans le chapitre précédent, le rôle important joué par les s-opérateurs dans le modèle des réseaux associatifs (*cf.* § 3.3 page 40). Or un s-opérateur est aussi la loi d'un demi-groupe abélien idempotent. Dans cette section nous décrivons ces structures, qui ont été étudiées dans d'autres contextes.

Cette présentation nous permet d'établir des comparaisons entre les s-opérateurs et les r-opérateurs que nous définissons dans le chapitre suivant. Ensuite, les structures scalaires des demi-anneaux ont été étendues à des structures matricielles, et le grand intérêt de l'algèbre max-plus est de rendre linéaire des équations qui ne l'étaient pas dans l'algèbre classique. Ainsi, même lorsque les calculs locaux dans le réseau associatif sont basés sur un s-opérateur, le calcul global — sous certaines réserves quant à la synchronisation — peut être vu comme un calcul matriciel, et la terminaison du calcul (stabilisation du réseau) comme la recherche d'un point fixe matriciel.

Il nous semble utile de rappeler quelques définitions d'algèbre, car les auteurs ne sont pas tous d'accord sur la terminologie. Une présentation des structures algébriques simples peut être trouvée dans [Gaubert, 1998] ou dans [Baccelli et al., 1992]. Néanmoins ces présentations sont orientées vers les structures à deux lois (demi-anneaux, dioïdes). Puisque nous faisons appel à des structures à une loi dans le chapitre 5, nous proposons tout d'abord une synthèse des structures à une loi au § 4.1.2 page 54, avant de passer en revue les structures à deux lois au § 4.1.3. Nous abordons ensuite les structures à trois lois et les matrices au § 4.1.4, la résolution d'équations au § 4.1.5 et l'algèbre des chemins (§ 4.1.6 et 4.1.7), que nous évoquons aux chapitres 5 et 7. Finalement au § 4.1.8 nous présentons une modélisation matricielle de la `direct-association` supposée synchrone.

La présentation qui suit n'est pas complète ; elle se veut juste suffisante pour les développements des chapitres suivants.

4.1.1 Introduction*

Le terme *algèbre max-plus* recouvre un domaine de recherche très vaste, qui inclut l'algèbre à proprement parler, mais aussi ses applications, telles que les problèmes de cheminement dans les graphes, les files d'attentes, les chaînes de Markov, les problèmes de production (flux dans un atelier, ...), les communications et les synchronisations dans les réseaux, l'évaluation de performance (systèmes électroniques, machines systoliques ou dirigées par les données, les réseaux de Pétri, et, en règle générale, les systèmes à événement discrets.

Historique. Les prémices de l'algèbre max-plus semblent remonter aux années cinquante avec les expressions régulières et les automates (Kleene en 1956, McNaughton et Yamada en 1960, cités dans [Aho et al., 1974]).

On trouve ensuite divers travaux exprimant les problèmes de cheminement dans les graphes à l'aide de structures algébriques adéquates (Roy en 1959, Maghout en 1963, Peteanu en 1967, Benzaken en 1968, cités dans [Gondran, 1975]). Puis, dans les années soixante-dix, on trouve les travaux de Carré en 1971 et ceux de Gondran et Minoux [Gondran, 1975] [Gondran et Minoux, 1977]. Un chapitre est consacré à cette algèbre et à ses applications en algorithmique de graphes dans les livres [Aho et al., 1974] et [Gondran et Minoux, 1979].

À partir de l'article de Cuninghame-Green en 1979 (cité dans [Gaubert et Plus, 1997]), l'algèbre max-plus a été étudiée pour ses applications dans les systèmes. C'est dans [Cohen et al., 1985] que la relation entre cette algèbre et la théorie des systèmes a été clairement établie. Le livre [Baccelli et al., 1992] présente une synthèse à ce sujet.

Parallèlement à l'étude de ses applications, l'algèbre max-plus a été étudiée en tant que telle. Des travaux récents ont montré l'intérêt de l'idempotence en mathématiques [Gunawardena, 1998]. Partant des structures max-plus, une algèbre complète, comparable à l'algèbre classique, est en construction [Gondran et Minoux, 1978] [Akian et al., 1994] [Gaubert et Loiseau, 1997]. Des extensions en analyse [Gondran, 1996a] [Gondran, 1996b] et en probabilité [Quadrat, 1998] ont également été proposées.

Exemple de modélisation par l'algèbre max-plus. Avant de détailler certains aspects de l'algèbre max-plus, nous proposons un exemple simple qui donne un aperçu concret de ses possibilités.

Au ciné-club, on aime voir le même film plusieurs fois, aussi longtemps qu'il est projeté. Le nombre de spectateurs est toujours faible au départ mais il augmente de séance en séance par le bouche à oreille. Le gérant du ciné-club cherche à savoir à partir de quelle séance il y aura affluence (salles combles).

Il constate que tout spectateur qui a vu un film le conseille à un ami (*cf.* figure 52 page ci-contre). En outre, alléché par la présentation de l'autre film, tout spectateur du second film ira voir le premier film, en invitant deux amis. Par contre, les spectateurs du premier film iront voir le second en invitant une seule personne.

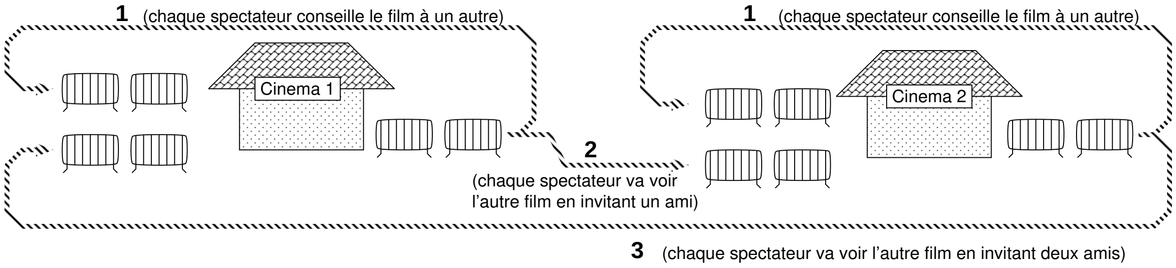


FIG. 52 – *Flux des spectateurs entre les deux cinémas.*

Le gérant note ses constatations de flux de spectateurs dans une matrice $\mathbf{A}$ telle que[(a)] $\mathbf{A}[i][j]$ soit égal au nombre de spectateurs qui arrivent au cinéma i en provenance du cinéma j. Lors de la première séance, il compte trois spectateurs dans le premier cinéma et quatre dans le second, ce qu'il note dans le vecteur $\mathtt{X}_1$. Il calcule alors le nombre de spectateurs pour les séances suivantes par la formule $\mathtt{X}_{k+1} = \mathbf{A} \times \mathtt{X}_k$. Ses calculs lui donnent :

$$\mathbf{A} = \begin{pmatrix} 1 & 3 \\ 2 & 1 \end{pmatrix} \quad \mathtt{X}_1 = \begin{pmatrix} 3 \\ 4 \end{pmatrix} \quad \mathtt{X}_2 = \mathbf{A} \times \mathtt{X}_1 = \begin{pmatrix} 15 \\ 10 \end{pmatrix} \quad \mathtt{X}_3 = \begin{pmatrix} 45 \\ 40 \end{pmatrix} \quad \ldots$$

Connaissant le nombre de spectateurs pour les séances prochaines, il peut alors savoir à partir de quelle projection les salles seront combles.

Maintenant le gérant se demande à quelles heures programmer les séances, de façon à permettre aux spectateurs d'enchaîner les films. Le premier film dure 1h 30 et le second 3h 30. Les spectateurs d'un film retournent dans la salle pour revoir le même film après une demi-heure. Ceux qui quittent un cinéma pour l'autre afin de voir l'autre film font le trajet en une 1h 30, pause comprise (*cf.* figure 53).

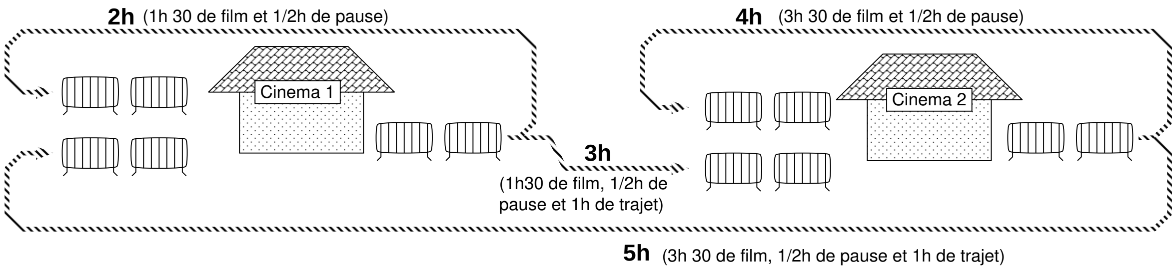


FIG. 53 – *Durée des trajets des spectateurs.*

La première séance commence à l'heure h_1 dans le premier cinéma, et à h_2 dans le second. Pour appliquer une technique de calcul semblable à la précédente, il lui faut résoudre des équations du type $\mathtt{X}_1[1] = \mathsf{max}(2 + h_1, 5 + h_2)$ et $\mathtt{X}_1[2] = \mathsf{max}(3 + h_1, 4 + h_2)$, qui ne sont plus linéaires dans l'algèbre classique. Si maintenant il remplace l'opération d'addition par le maximum et l'opération de multiplication par l'addition, son calcul devient (résultats donnés avec $h_1 = 15$ et $h_2 = 13$) :

$$\mathbf{A} = \begin{pmatrix} 2 & 5 \\ 3 & 4 \end{pmatrix}, \mathtt{X}_1 = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix}, \mathtt{X}_2 = \mathbf{A} \times \mathtt{X}_1 = \begin{pmatrix} \mathsf{max}(2 + h_1, 5 + h_2) \\ \mathsf{max}(3 + h_1, 4 + h_2) \end{pmatrix} = \begin{pmatrix} 19 \\ 19 \end{pmatrix}, \mathtt{X}_3 = \begin{pmatrix} 23 \\ 22 \end{pmatrix}$$

ce qui lui donne les heures des séances dans les deux salles. Son calcul devient semblable au précédent, mais, ce faisant, il est passé de l'algèbre classique à l'algèbre max-plus.

(a) La composante située à la $i^{\text{ème}}$ ligne et à la $j^{\text{ème}}$ colonne d'une matrice $\mathbf{A}$ est noté par $\mathbf{A}[i][j]$ (notations commodes du langage C).

4.1.2 Structures à une loi*

Afin de mieux cerner les s-opérateurs (et les algèbres idempotentes), nous décrivons maintenant les structures algébriques à une loi.

Magma, monoïde et demi-groupe. Un *magma* $(\mathbb{S}, \diamond)$ est un ensemble muni d'une loi de composition interne**45**. Un *monoïde* est un magma dont la loi est associative**(b)**. Un élément $x \in \mathbb{S}$ est dit *régulier***47** (à gauche et à droite) s'il vérifie, pour tous y et z de $\mathbb{S}$, $(x \diamond y = x \diamond z) \Rightarrow (y = z)$ et $(y \diamond x = z \diamond x) \Rightarrow (y = z)$. Lorsque tous les éléments sont réguliers, le monoïde est dit *libre***46**. Un *demi-groupe***48** est un monoïde ayant un élément $e_\diamond$ qui est neutre à gauche et à droite**49**. Un demi-groupe est un groupe auquel il manque la notion d'inverse**50**. Lorsque la loi est commutative, la structure est dite commutative ou abélienne. Lorsque la loi est idempotente, la structure est dite idempotente**51**.

45. Loi de composition interne sur $\mathbb{S}$: $\forall x, y \in \mathbb{S}, x \diamond y \in \mathbb{S}$.

46. En anglais, *cancellative monoid* [Gaubert, 1998] ou *free monoid* [Zielonka, 1987]. Le terme *monoïde libre* est utilisé dans [Bourbaki, 1964].

47. En anglais, *cancellative* [Gaubert et Plus, 1997]. Le terme *régulier* est utilisé dans [Bourbaki, 1964].

48. En anglais, *semi-group*. Traduction de [Reinhardt et Soeder, 1997].

49. Elément neutre : à gauche : $\forall x \in \mathbb{S}$, $e_\diamond \diamond x = x$, à droite : $x \diamond e_\diamond = x$.

50. Inverse : $\forall x \in \mathbb{S}, x \diamond x^{-1} = x^{-1} \diamond x = e_\diamond$.

51. Idempotence : $\forall a \in \mathbb{S}, x \diamond x = a$.

Exemples. La structure $(\mathbb{N} \setminus \{0\}, +)$ est un monoïde libre. Lorsqu'on ajoute l'élément 0, qui est le neutre de l'opération $+$, on obtient le demi-groupe $(\mathbb{N}, +)$. Il ne manque plus que les inverses (les entiers négatifs) pour obtenir le groupe $(\mathbb{Z}, +)$. La structure $(\mathbb{N}, \min)$ est un monoïde qui n'est pas libre. Lorsqu'on ajoute l'élément $+\infty$, on obtient le demi-groupe $(\mathbb{N} \cup \{+\infty\}, \mathsf{min})$, qui est idempotent (on note $\overline{\mathbb{N}} = \mathbb{N} \cup \{+\infty\}$). Nous appelons *demi-groupe tropical* cette structure, par conformité avec le terme de demi-anneau tropical (*cf.* § 4.1.3 page 56). Elle nous servira pour illustrer les différents concepts de cette section.

Relation d'ordre. Tout demi-groupe admet une relation de pré-ordre**52** $\preccurlyeq_\diamond$ définie**(c)** par $a \preccurlyeq_\diamond b \equiv \exists c, a = b \diamond c$. Puisque $a = e_\diamond \diamond a$, on a $a \preccurlyeq_\diamond e_\diamond$ pour tout $a \in \mathbb{S}$. Lorsque la relation binaire $\preccurlyeq_\diamond$ est anti-symétrique (*i.e.*, $x \preccurlyeq_\diamond y$ et $y \preccurlyeq_\diamond x$ implique $x = y$), on dit que le demi-groupe est *naturellement ordonné* par $\preccurlyeq_\diamond$, qui est alors une relation d'ordre partiel. La relation d'ordre $\preccurlyeq_\diamond$ est compatible**53** avec $\diamond$. C'est une relation d'ordre totale si et seulement si $a \diamond b$ est soit égal à a, soit égal à b.

On montre qu'un demi-groupe naturellement ordonné, n'a pas de *somme nulle* : si $a \diamond b = e_\diamond$, alors $a = b = e_\diamond$. En effet, $a \diamond b = e_\diamond$ implique $e_\diamond \preccurlyeq_\diamond a$. Mais puisque $a \preccurlyeq_\diamond e_\diamond$, on a $a = e_\diamond$ (par anti-symétrie de $\preccurlyeq_\diamond$). D'où $e_\diamond \diamond b = e_\diamond$ et $b = e_\diamond$.

De même, un demi-groupe libre et sans somme nulle est naturellement ordonné. En effet, il suffit de montrer que la relation de pré-ordre est anti-symétrique. Or, si $a \preccurlyeq_\diamond b \preccurlyeq_\diamond a$, il existe c et d tels que $a = b \diamond c$ et $b = a \diamond d$, d'où $a = a \diamond d \diamond c$. En simplifiant par a (demi-groupe libre), on obtient $e_\diamond = d \diamond c$, ce qui donne $c = d = e_\diamond$ (pas de somme nulle) et donc $a = b$.

Un demi-groupe naturellement ordonné est dit idempotent lorsque l'opération $\diamond$ est idempotente. Dans ce cas, la relation d'ordre $\preccurlyeq_\diamond$ vérifie $a \preccurlyeq_\diamond b \Leftrightarrow a = b \diamond a$. En effet, $a \preccurlyeq_\diamond b \Rightarrow a = b \diamond c$ pour un $c \in \mathbb{S}$, d'où $b \diamond a = b \diamond b \diamond c = b \diamond c = a$ grâce à l'idempotence. L'élement $b \diamond a$ coïncide avec le plus grand élément de $\mathbb{S}$ qui est à la fois plus petit que a et que b**(d)**. La présence d'un élément neutre interdit au demi-groupe d'être à la fois idempotent et libre, à moins d'être trivial. En effet, $a = a \diamond e_\diamond = a \diamond a \Rightarrow e_\diamond = a$ et le demi-groupe est réduit au singleton $\{e_\diamond\}$.

52. Relation de pré-ordre : relation binaire réflexive ($x \preccurlyeq_\diamond x$) et transitive ($x \preccurlyeq_\diamond y$ et $y \preccurlyeq_\diamond z$ implique $x \preccurlyeq_\diamond z$).

53. Compatibilité : si $x \preccurlyeq_\diamond y$, alors $\forall z \in \mathbb{S}, x \diamond z \preccurlyeq_\diamond y \diamond z$.

(b) Dans [Bourbaki, 1964], un monoïde est défini par la donnée d'une loi de composition interne associative sur un ensemble. C'est aussi le cas dans [Gondran, 1975]. Dans [Gaubert, 1998], une telle structure est un demi-groupe, tandis qu'un monoïde est obtenu par l'ajout d'un élément neutre. On trouve la même définition du monoïde dans [Akian et al., 1994]. Dans [Reinhardt et Soeder, 1997], monoïde est synonyme de demi-groupe. Nous adoptons la définition du monoïde de [Bourbaki, 1964] et nous appelons demi-groupe un monoïde ayant un neutre à gauche et à droite. Ainsi, il ne manque que l'inversion au demi-groupe pour être un groupe. Notons que tant que le la loi n'est pas commutative ni inversible, il est nécessaire de préciser que l'élément neutre est neutre à gauche et à droite (la commutativité garantit en effet que le neutre à gauche l'est aussi à droite ; il en est de même pour la notion d'inverse : $x = e_\diamond \diamond x = (x \diamond x^{-1}) \diamond x = x \diamond (x^{-1} \diamond x) = x \diamond e$). Par contre, s'il y a un un neutre à gauche $e_\diamond$ et un autre à droite $e'_\diamond$, alors ils sont identiques : $e_\diamond = e_\diamond \diamond e'_\diamond = e'_\diamond$.

(c) Puisque nous basons notre présentation sur le demi-groupe tropical, et que en règle générale nous aurons affaire au minimum, il nous semble plus intuitif de définir la relation $\preccurlyeq_\diamond$ comme signifiant «plus petit que». En règle général, les présentations de l'algèbre max-plus [Baccelli et al., 1992] [Gaubert et Plus, 1997] [Gaubert, 1998] prennent le parti inverse parce qu'elles s'intéressent principalement à la structure de demi-anneau $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \mathsf{max}, +)$. Dans $(\overline{\mathbb{N}}, \mathsf{min})$, $e_\diamond = e_{\mathsf{min}} = +\infty$.

(d) *cf.* preuve du théorème 14 page 40. Ce théorème est établi pour les s-opérateurs $\oplus$ (commutatifs) ; la relation $\preceq_\oplus$ est définie par $x \preceq_\oplus \equiv x = x \oplus y$.

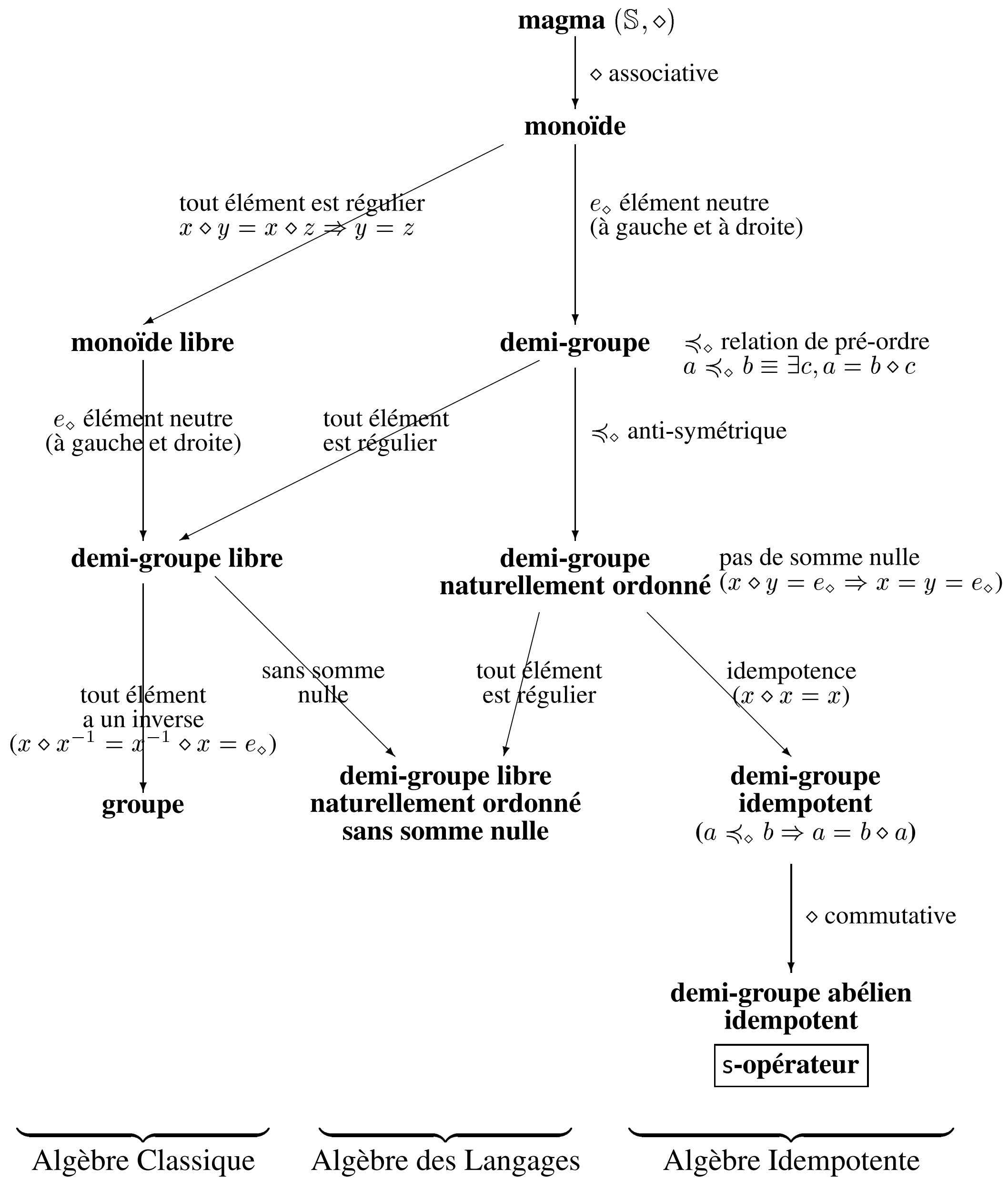


FIG. 54 – *Structures à une loi.*

54. Demi-treillis inférieur / supérieur : ensemble ordonné tel que toute paire d'éléments admet une plus petite / grande borne supérieure / inférieure. Il est dit *complet* s'il existe une plus petite / grande borne supérieure / inférieure pour tout sous-ensemble fini ou infini.

Demi-treillis. Un demi-groupe idempotent est un demi-treillis inférieur[54] : la borne inférieure de deux éléments a et b est donnée par $a \diamond b$. Si le demi-groupe tropical est un demi-treillis inférieur complet, il n'en est pas de même pour tous les demi-groupes idempotents. Par exemple le demi-groupe idempotent $(\mathbb{Z} \cup \{+\infty\}, \mathsf{min})$ n'est pas un demi-treillis inférieur complet car le sous-ensemble des entiers négatifs pairs $\{-2k, k \in \mathbb{N}\}$ n'a pas de borne inférieure. Pour obtenir un demi-groupe idempotent qui soit un demi-treillis inférieur complet, il faut parfois ajouter un plus petit élément. Ainsi $(\mathbb{Z} \cup \{-\infty, +\infty\}, \mathsf{min})$ est un demi-groupe idempotent qui est aussi un demi-treillis inférieur complet. L'intérêt de la complétude des demi-groupes idempotents apparaîtra lors de la résolution d'équations dans les demi-anneaux (*cf.* § 4.1.5 page 58).

Schismes. L'absence de somme nulle est incompatible avec la notion d'inverse. De même, l'idempotence est incompatible avec la notion de régularité.

On distingue donc trois grandes familles de structures à une loi (*cf.* figure 54 page précédente). La première est celle du demi-groupe libre admettant des sommes nulles et qui est destinée à devenir un groupe. Cette structure est étudiée dans le cadre de l'algèbre «classique». Elle admet des sommes nulles et la loi ne peut être idempotente.

La seconde est celle du demi-groupe libre naturellement ordonné et sans somme nulle. Cette structure ne peut admettre d'inverses, et la loi ne peut être idempotente. On obtient une telle structure, par exemple, en munissant l'ensemble Σ^* des mots construits sur un alphabet Σ de l'opération de concaténation. Cette opération n'est pas idempotente. L'élément neutre est le mot vide ε ; il est neutre à gauche et à droite ($\varepsilon u = u\varepsilon = u$). La relation de pré-ordre est donnée par la notion de préfixe : le mot u est plus petit que le mot w si le mot w admet u comme préfixe ($w = uv$). Cette relation est anti-symétrique car deux mots préfixes l'un de l'autre sont égaux. Le demi-groupe est libre et n'admet pas de somme nulle. Ce type de structure est étudié en théorie des langages ; c'est pourquoi nous l'appelons *algèbre des langages* sur la figure 54 page précédente.

La troisième famille est celle des demi-groupes idempotents, qui ne peuvent être libres et qui ne pourront pas être complétés par des inverses[(e)]. L'algèbre max-plus concerne cette dernière famille. Les s-opérateurs présentés au § 3.3.2 page 40, sont des lois de demi-groupes abéliens et idempotents. Il en est de même des *infimum* [Tel, 1991] (*cf.* § 6.1.4 page 107). Comme cela a été signalé dans [Baccelli et al., 1992], l'absence d'inverse est compensée par l'idempotence, qui se révèle assez utile pour simplifier et résoudre les équations. L'idempotence apparaît comme une notion fondamentale [Gunawardena, 1998]. L'intérêt de cette propriété pour notre travail est lié à la terminaison des calculs dans les réseaux présentant des circuits (*cf.* proposition 3 page 31).

4.1.3 Structures à deux lois*

55. En anglais, *semi-ring*.

56. Distributivité : $(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$.

57. Absorbant : $x \otimes e_\oplus = e_\oplus \otimes x = e_\oplus$.

Demi-anneaux et demi-corps. Un *demi-anneau*[55] est un ensemble $\mathbb{S}$ muni de deux lois $\oplus$ (addition) et $\otimes$ (multiplication) tel que $(\mathbb{S}, \oplus)$ est un demi-groupe abélien et $(\mathbb{S}, \otimes)$ est un demi-groupe[(f)]. La multiplication $\otimes$ doit en outre être distributive[56] par rapport à l'addition $\oplus$, et l'élément neutre $e_\oplus$ (appelé *zéro*) doit être absorbant[57] pour la multiplication. Cette dernière condition est superflue si le demi-groupe $(\mathbb{S}, \oplus)$ est libre[(g)] : $e_\oplus \oplus a \otimes e_\oplus = a \otimes e_\oplus = a \otimes (e_\oplus \oplus e_\oplus) = a \otimes e_\oplus \oplus a \otimes e_\oplus$, et par régularité de $a \otimes e_\oplus$, on obtient $a \otimes e_\oplus = e_\oplus$. L'élément neutre $e_\otimes$ est appelé *unité*. Cet élément, ainsi que le zéro, sont déterminés de façon unique, ce qui permet de noter, suivant [Gaubert, 1998], le demi-anneau[(h)] $(\mathbb{S}, \oplus, \otimes)$.

Les qualificatifs «libre», «sans somme nulle» et «idempotent» d'un demi-anneau concernent le demi-groupe additif $(\mathbb{S}, \oplus)$. Par contre le qualificatif «commutatif» concerne le demi-groupe multi-

(e) Des travaux ont cependant défini la notion d'inverse dans un demi-anneau idempotent, en considérant des couples ; cette construction s'apparente à celle de l'ensemble $\mathbb{C}$ à partir de $\mathbb{R}$.

(f) La première loi étant commutative, elle est appelée addition tandis que la deuxième est appelée multiplication, par analogie avec les structures classiques.

(g) On admet la priorité de la multiplication sur l'addition, ce qui permet de supprimer des parenthèses : $a \otimes b \oplus c = (a \otimes b) \oplus c$.

(h) Dans [de la Torre et Kruskal, 1995], une demi-anneau est noté $\mathcal{S} = (S, \oplus, \odot, \overline{0}, \overline{1})$. De même, dans [Aho et al., 1974], un demi-anneau est noté $(\mathbb{S}, +, \cdot, 0, 1)$. Les symboles $+$ et $\cdot$ peuvent être confondus avec les lois habituelles (sur $\mathbb{Z}$ par exemple) pour certaines applications. C'est pourquoi les lois sont généralement notées $\oplus$ et $\otimes$.

plicatif $(\mathbb{S}, \otimes)$. Un demi-anneau idempotent est appelé *dioïde*[(i)]. Un *demi-corps* est un demi-anneau dans lequel tout élément sauf le zéro a un inverse pour la multiplication.

Exemples. La structure $(\mathbb{R}^+, +, \times)$ est un demi-corps libre sans somme nulle. $(\mathbb{R}^+, \mathsf{max}, +)$ est un demi-anneau idempotent. La structure $\mathbb{R}_{\mathsf{max}} = (\mathbb{R} \cup \{-\infty\}, \mathsf{max}, +)$ est un demi-corps idempotent. Il est appelé *demi-corps max-plus*. La structure $\mathbb{N}_{\mathsf{min}} = (\mathbb{N} \cup \{+\infty\}, \mathsf{min}, +)$ est un demi-anneau idempotent appelé *demi-anneau tropical*[(j)]. La structure $\mathbb{Z}_{\mathsf{min}} = (\mathbb{Z} \cup \{+\infty\}, \mathsf{min}, +)$ est un demi-corps idempotent, que nous appellerons *demi-corps tropical*.

Relation d'ordre. L'ordre partiel d'un dioïde défini par $a \preccurlyeq_\oplus b \equiv a \oplus b = a$ (*cf.* § 4.1.2 page 54), est compatible à gauche et à droite avec la multiplication $\otimes$. On dit alors que la multiplication est *isotone* à gauche et à droite, c'est-à-dire qu'elle vérifie, pour tout $c \in \mathbb{S}$, $a \preccurlyeq_\oplus b \Rightarrow a \otimes c \preccurlyeq_\oplus b \otimes c$ et $c \otimes a \preccurlyeq_\oplus c \otimes b$.

Un dioïde — ou plus généralement un demi-anneau — est *complet* si le demi-groupe idempotent $(\mathbb{S}, \oplus)$ est un demi-treillis inférieur complet (*i.e.*, tout sous-ensemble a une borne inférieure, *cf.* § 4.1.2) et si la multiplication est distributive par rapport aux sommes infinies. Dans un dioïde complet, le plus petit élément existe toujours ; on le note $\perp$. Il est égal à la somme de tous les éléments au sens de $\oplus$: $\perp = \oplus\{x, x \in \mathbb{S}\}$. Il est toujours absorbant pour l'addition ($\perp \oplus a = \perp$). Par contre, puisque $e_\oplus$ est absorbant pour la multiplication, on a toujours $\perp \otimes e_\oplus = e_\oplus$. Dans $\overline{\mathbb{Z}}_{\mathsf{min}} = (\mathbb{Z} \cup \{-\infty, +\infty\}, \mathsf{min}, +)$, où le plus petit élément est $\perp = -\infty$ et où $e_\oplus = +\infty$, cela signifie que $-\infty \otimes +\infty = +\infty$).

Treillis. Un dioïde complet muni de la relation d'ordre $\preccurlyeq_\oplus$ est un demi-treillis inférieur complet (*cf.* § 4.1.2 page 54). De plus, $e_\oplus$ joue le rôle de plus grand élément. Or tout demi-treillis inférieur ayant un plus grand élément est en fait un treillis[58] complet. La borne inférieure de deux éléments est donnée par $\oplus$ (*cf.* § 4.1.2 page 54). On dénote alors la borne supérieure de a et b par $a \vee b$, et on a :

58. Treillis : à la fois demi-treillis inférieur et supérieur.

$$a \preccurlyeq_\oplus b \quad \Leftrightarrow \quad a \oplus b = a \quad \Leftrightarrow \quad b = a \vee b$$

4.1.4 Structures à trois lois, matrices*

Moduloïdes. Les moduloïdes peuvent être comparés aux espaces vectoriels de l'algèbre classique construits sur des corps. Ici, le demi-corps remplace le corps. Un moduloïde[(k)] $(\mathbb{M}, \oplus, \cdot)$ sur le demi-corps $(\mathbb{S}, \oplus, \otimes)$ est un ensemble de *vecteurs*[(l)] $\mathtt{X}$ dont les composantes $\mathtt{X}[i]$ sont des éléments du demi-corps ($\mathtt{X}[i] \in \mathbb{S}$), et dont les deux lois vérifient :

(a) $(\mathbb{M}, \oplus)$ est un demi-groupe abélien ; l'opération $\oplus$ est notée (par abus) de la même façon que l'addition du demi-corps ; elle est définie comme l'addition terme à terme de deux vecteurs ; son élément neutre est noté[59] $\mathtt{E}_\oplus = (\dots, e_\oplus, \dots)^t$;

(b) $\cdot$ est une loi de composition externe[60], dont le symbole est souvent omis, et qui correspond à la multiplication des composantes d'un vecteur $\mathtt{X}$ de $\mathbb{M}$ par un scalaire de $\mathbb{S}$;

(c) $\oplus$ et $\cdot$ sont distributives l'une par rapport à l'autre, et $\cdot$ vérifie $\alpha(\beta\mathtt{X}) = \beta(\alpha\mathtt{X})$ pour tous α et β dans $\mathbb{S}$;

(d) $e_\oplus\mathtt{X} = \mathtt{E}_\oplus = (\dots, e_\oplus, \dots)^t$ et $e_\otimes\mathtt{X} = \mathtt{X}$.

59. Transposée : un vecteur $\mathtt{V}$ est noté sous forme colonne. Pour les noter sous forme ligne, on utilise la transposée $\mathtt{V}^t$, qui transforme les lignes en colonnes.

60. Loi de composition externe : application de $\mathbb{S} \times \mathbb{M}$ dans $\mathbb{M}$.

(i) Un demi-groupe idempotent ne peut être complété pour devenir un groupe (*cf.* § 4.1.2 page 54). De même, un demi-anneau idempotent ne peut être complété pour devenir un anneau (un anneau est un demi-anneau dont les éléments ont un inverse pour l'addition, appelé *opposé*). Pour Gondran et Minoux, le terme de demi-anneau semble suggérer que la structure peut être complétée pour devenir un anneau. Aussi préfèrent-ils désigner un demi-anneau par le terme *dioïde*, construit en remplaçant le préfixe *mono-* par *di-* dans le terme monoïde. Ce terme a été introduit par Kuntzmann en 1972 (cité dans [Baccelli et al., 1992]). Cependant les auteurs semblent maintenant s'accorder sur le fait qu'un dioïde est un demi-anneau idempotent [Baccelli et al., 1992], [Gaubert et Plus, 1997], [Akian et al., 1994].

(j) L'adjectif «tropical» a été proposé par I. Simon, sur une suggestion de C. Choffrut [Simon, 1998]. Notons que dans [Gaubert, 1998], les demi-anneaux présentés sont qualifiés d'«exotiques».

(k) Le terme de moduloïde est utilisé dans [Baccelli et al., 1992]. Cependant cette structure est appelée demi-module dans [Gaubert, 1998] (en anglais *semimodule*).

(l) Le terme « vecteur » est abusif ici : *cf.* exemple des matrices un peu après. Il est employé dans [Gaubert, 1998].

Si de plus le moduloïde admet une seconde loi interne, notée $\otimes$ comme celle du demi-corps, et dont l'élément neutre est noté $\mathtt{E}_\otimes$, telle que $(\mathbb{M}, \otimes, \oplus)$ est un demi-anneau, alors la structure $(\mathbb{M}, \otimes, \oplus, \cdot)$ est appelée *demi-algèbre* sur le demi-corps $(\mathbb{S}, \oplus, \otimes)$. Lorsque le demi-corps $(\mathbb{S}, \oplus, \otimes)$ est idempotent, le moduloïde $(\mathbb{M}, \oplus, \cdot)$ est dit *idempotent* (son addition $\oplus$ est effectivement idempotente dans ce cas). De même, $(\mathbb{M}, \oplus, \otimes, \cdot)$ est appelée *demi-algèbre idempotente*[(m)].

61. Élément neutre $\mathbf{E}_\oplus$ de l'addition matricielle :

$$\begin{matrix} e_\oplus & \cdots & e_\oplus \\ \vdots & \ddots & \vdots \\ e_\oplus & \cdots & e_\oplus \end{matrix}$$

62. Élément neutre $\mathbf{E}_\otimes$ de la multiplication matricielle :

$$\begin{matrix} e_\otimes & e_\oplus & \cdots & e_\oplus \\ e_\oplus & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & e_\oplus \\ e_\oplus & \cdots & e_\oplus & e_\otimes \end{matrix}$$

63. Élément neutre $\mathbf{E}_{\mathsf{min}}$ de l'addition matricielle dans le demi-corps tropical :

$$\begin{matrix} +\infty & \cdots & +\infty \\ \vdots & \ddots & \vdots \\ +\infty & \cdots & +\infty \end{matrix}$$

64. Élément neutre $\mathbf{E}_+$ de la multiplication matricielle dans le demi-corps tropical :

$$\begin{matrix} 0 & \infty & \cdots & \infty \\ \infty & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty \\ \infty & \cdots & \infty & 0 \end{matrix}$$

Exemple : vecteurs de $\mathbb{Z}_{\mathbf{min}}$. L'ensemble des vecteurs de taille n à coefficients dans le demi-corps tropical $\mathbb{Z}_{\min} = (\mathbb{Z} \cup \{+\infty\}, \mathsf{min}, +)$ est noté $(\mathbb{Z}_{\min})^n$. Lorsqu'on munit cet ensemble du minimum terme à terme de deux vecteurs (loi interne), et de l'addition d'un scalaire de $\mathbb{Z}_{\min}$ aux composantes d'un vecteur (loi externe), on obtient un moduloïde idempotent. L'élément neutre pour la loi interne min est le vecteur $\mathtt{E}_{\mathsf{min}} = (\dots, +\infty, \dots)^t$.

Exemple : matrices de $\mathbb{Z}_{\mathbf{min}}$. L'ensemble des matrices $n \times n$ à coefficients dans le demi-corps tropical $\mathbb{Z}_{\min}$ est noté $(\mathbb{Z}_{\min})^{n \times n}$. Lorsqu'on munit cet ensemble du minimum terme à terme de deux matrices (loi interne additive) et de l'addition d'un scalaire de $\mathbb{Z}_{\min}$ à chacune des composantes de la matrice (loi externe), on obtient le moduloïde idempotent tropical $(\mathbb{Z}, \mathsf{min}, +)$. Pour obtenir une demi-algèbre idempotente, que nous appelons *demi-algèbre tropicale* et que nous notons $(\mathbb{Z} \cup \{+\infty\}, \oplus, \otimes, \cdot) = (\mathbb{Z} \cup \{+\infty\}, \mathsf{min}, \otimes, \cdot)$, on définit une multiplication matricielle $\otimes$ (seconde loi interne) de la façon suivante :

$$(\mathbf{A} \otimes \mathbf{B})\,[i][j] = \mathbf{MIN}_{k=1}^{n}\,(\mathbf{A}[i][k] + \mathbf{B}[k][j])$$

La matrice $\mathbf{E}_\oplus$ est l'élément neutre de l'addition matricielle[61 63] ; elle est remplie de $e_\oplus$. La matrice $\mathbf{E}_\oplus$ est l'élément neutre de la multiplication matricielle[62 64] ; elle est composée d'une diagonale de $e_\otimes$, et de $e_\oplus$ aux autres entrées.

Multiplication matricielle. On généralise la multiplication matricielle à tout moduloïde $(\mathbb{M}, \oplus, \otimes, \cdot)$ par la formule :

$$(\mathbf{A} \otimes \mathbf{B})\,[i][j] = \bigoplus_{k=1}^{n} \mathbf{A}[i][k] \otimes \mathbf{B}[k][j]$$

4.1.5 Résolution d'équations*

L'élément a^*. Pour tout élément a d'un demi-anneau $(\mathbb{S}, \oplus, \otimes)$, on pose $a^* = e_\otimes \oplus a \oplus a^2 \oplus \cdots$. Par exemple dans le demi-corps $\mathbb{Z}_{\min} = (\mathbb{Z} \cup \{+\infty\}, \mathsf{min}, +)$, si a^* existe, il vaut $a^* = \mathsf{min}(0, a, 2a, \dots) = 0$. De même, dans une demi-algèbre de matrices — qui est un demi-anneau pour les deux lois internes (*cf.* § 4.1.4 page précédente) —, on pose :

$$\begin{array}{rcl} \mathbf{A}^{(k)} & = & \mathbf{E}_\otimes \oplus \mathbf{A} \oplus \cdots \oplus \mathbf{A}^k \\ \mathbf{A}^* & = & \lim_{k \to +\infty} \mathbf{A}^{(k)} = \mathbf{E}_\otimes \oplus \mathbf{A} \oplus \cdots \oplus \mathbf{A}^k \oplus \cdots \end{array} \tag{4.1}$$

Des conditions assurant l'existence de la matrice $\mathbf{A}^*$ sont données au § 4.1.6 page suivante.

Point fixe. Le résultat suivant est donné notamment dans [Gaubert, 1998]. Pour l'obtenir, on utilise la règle de Fubini, qui n'est vraie que lorsque le demi-anneau est complet : $\bigoplus_{x \in X} x = \bigoplus_{i \in I} \bigoplus_{y \in X_i} y$, avec $\sqcup_{i \in I} X_i = X$ (unions disjointes). Si a et b sont deux éléments d'un demi-anneau $(\mathbb{S}, \oplus, \otimes)$ complet (*cf.* § 4.1.3 page 56), alors

$$\begin{array}{ll} ax \oplus b \preccurlyeq_\oplus x \Rightarrow a^* b \preccurlyeq_\oplus x & x = ax \oplus b \Rightarrow x = a^* b \\ \mathbf{A}\mathtt{X} \oplus \mathtt{B} \preccurlyeq_\oplus \mathtt{X} \Rightarrow \mathbf{A}^*\mathtt{B} \preccurlyeq_\oplus \mathtt{X} & \mathtt{X} = \mathbf{A}\mathtt{X} \oplus \mathbf{B} \Rightarrow \mathtt{X} = \mathbf{A}^*\mathtt{B} \end{array} \tag{4.2}$$

(m) Dans [Baccelli et al., 1992], une telle structure est appelée algèbre. Nous préférons cependant le terme de demi-algèbre par analogie avec les termes demi-groupes, demi-anneaux et demi-corps qui traduisent tous le fait que la première loi n'est pas inversible.

Exemple. La figure 55 illustre ce qu'est la recherche du point fixe dans le demi-corps tropical $\mathbb{Z}_{\mathsf{min}}$, plongé dans $\mathbb{R}_{\mathsf{min}}$ pour des raisons de continuité et de tracé des droites. Par exemple, si $a = 2$ et $b = 3$, l'équation considérée est $\mathsf{min}(x + 2, 3) = x$. On a $a^* = 2^* = \mathsf{min}(0, 2, 2 + 2, \dots) = 0$, et $a^*b = 2^* + 3 = 3$.

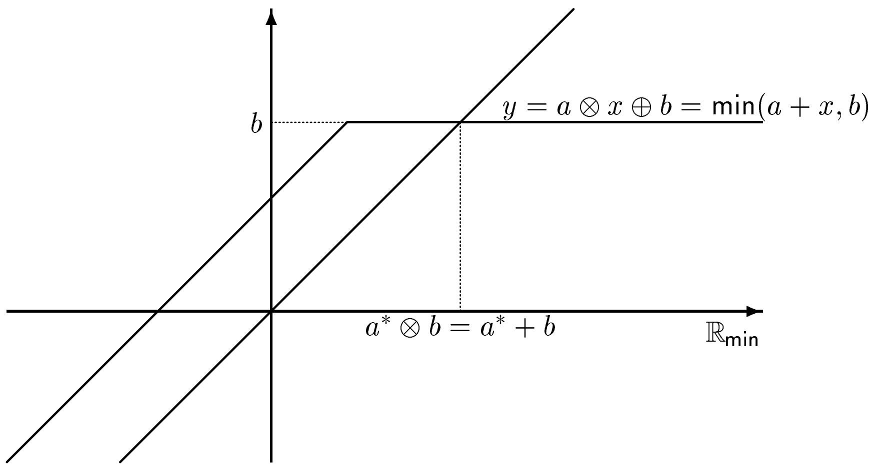


FIG. 55 – *Résolution d'une équation de point-fixe dans* $(\mathbb{R}_{\mathsf{min}}, \oplus, \otimes) = (\mathbb{R}_{\mathsf{min}}, \mathsf{min}, +)$.

4.1.6 Algèbre de chemins*

Les opérations matricielles peuvent se traduire en terme de graphe. Et réciproquement, divers problèmes de graphe, tels que la recherche des chemins de poids minimaux, peuvent se traduire en termes matriciels dans un moduloïde correctement choisi. Ces applications font parfois appeler *algèbre de chemins* les structures algébriques que nous venons de présenter.

Graphe de précédence. Partant d'une matrice $n \times n$ $\mathbf{A}$ dans un moduloïde $(\mathbb{M}, \oplus, \otimes, \cdot)$, on définit le *graphe de précédence*$^{\mathbf{(n)}}$ $G[\mathbf{A}]$ comme étant le graphe orienté pondéré à n sommets tel qu'il y a un arc de j à i si et seulement si $\mathbf{A}[i][j] \neq e_\oplus$, et dans ce cas le poids de cet arc est égal à $\mathbf{A}[i][j]$. Réciproquement, on définit la *matrice de précédence* $\mathbf{A}[G]$ du graphe G.

Poids des chemins. Le poids d'un chemin du graphe d'incidence d'une matrice d'une demi-algèbre est le produit (au sens de $\otimes$) des poids des arcs du chemin.

Par exemple, si $\mathbf{A}$ est une matrice de la demi-algèbre tropicale $(\mathbb{Z} \cup \{+\infty\}, \mathsf{min}, +, \cdot)$, alors le poids d'un chemin est la somme (au sens de $+$, qui est la multiplication ici) des poids des arcs le composant.

Relation entre le calcul matriciel et les graphes. $(\mathbf{A}^k)[i][j]$ désigne la composante de la $i^{\text{ème}}$ ligne et de la $j^{\text{ème}}$ colonne de la matrice $\mathbf{A}$ à la puissance k. De plus, au § 4.1.5 page précédente, nous avons posé $\mathbf{A}^{(k)} = \mathbf{E}_\otimes \oplus \mathbf{A} \oplus \cdots \oplus \mathbf{A}^k$ et $\mathbf{A}^* = \lim_{k \to +\infty} \mathbf{A}^{(k)}$. Si $\mathbf{A}$ est une matrice dans une demi-algèbre, on a [Gondran, 1975] :

- $(\mathbf{A}^k)[i][j]$ est la somme (au sens de $\oplus$) des poids des chemins de j à i ayant exactement k arcs,
- $(\mathbf{A}^{(k)})[i][j]$ est la somme des poids des chemins de j à i ayant au plus k arcs.

L'existence de la matrice $\mathbf{A}^*$ — qui se révèle utile pour la résolution d'équations (*cf.* § 4.1.5 page ci-contre) —, dépend de la présence ou non de circuits dits *absorbants* dans le graphe $G[\mathbf{A}]$: le circuit

(n) Tel que défini ici, les arcs entrants du $i^{\text{ème}}$ sommet du graphe de précédence $G[\mathbf{A}]$ sont codés par la $i^{\text{ème}}$ ligne de la matrice $\mathbf{A}$. Cette convention semble naturelle pour les modélisations telles que celle que nous avons proposée dans l'introduction (§ 4.1.1 page 52). C'est aussi celle utilisée dans le livre [Baccelli et al., 1992]. Cependant, dans les références [Gondran, 1975], [Gondran et Minoux, 1979] et [Gaubert, 1998] notamment, la convention inverse est utilisée : on définit le *graphe d'incidence* tel que la $i^{\text{ème}}$ ligne de $\mathbf{A}$ code les arcs sortants du sommet i.

C, de poids $w(C)$, est dit *absorbant* si $e_\otimes \oplus w(C) \neq e_\otimes$. On montre que si $G[\mathbf{A}]$ est sans circuit absorbant, alors il existe $p < n$ tel que $\mathbf{A}^* = \mathbf{A}^{(p)}$ [Gondran, 1975].

Exemple. La recherche des chemins de poids minimum entre toutes paires de sommets dans un graphe orienté pondéré à n sommets est effectuée dans la demi-algèbre tropicale $(\mathbb{Z}\cup\{+\infty\}, \mathsf{min}, +, \cdot)$. La matrice $\mathbf{A}$ donne les poids minimaux des chemins de longueur 1 (arcs) entre chaque paire de sommets. $\mathbf{A}^2$ donne les poids minimaux des chemins de longueur 2, et ainsi de suite. Pour obtenir les chemins de poids minimaux entre toutes paires de sommets, on calcule $\mathsf{min}(\mathbf{A}, \mathbf{A}^2, \dots)$.

Quand tous les poids sont positifs, il est préférable dans la recherche des poids minimaux, de rester sur le sommet i plutôt que de parcourir un circuit partant de i et finissant en i. Aussi considère-t-on la matrice $\mathbf{E}_\otimes \oplus \mathbf{A} = \mathsf{min}(\mathbf{E}_\otimes, \mathbf{A})$ qui est de diagonale nulle ($e_\otimes = 0$ ici).

Puisque $\oplus = \mathsf{min}$ est commutative et idempotente, les coefficients du binôme de Newton disparaissent en développant $(\mathbf{E}_\otimes \oplus \mathbf{A})^k$. Par exemple, pour $k = 2$, on a :

$$\begin{aligned}(\mathbf{E}_\otimes \oplus \mathbf{A})^2 &= (\mathbf{E}_\otimes \oplus \mathbf{A}) \otimes (\mathbf{E}_\otimes \oplus \mathbf{A}) = \mathbf{E}_\otimes^2 \oplus \mathbf{E}_\otimes \otimes \mathbf{A} \oplus \mathbf{A}_\otimes \otimes \mathbf{E}_\otimes \oplus \mathbf{A}^2 \\ &= \mathbf{E}_\otimes \oplus \mathbf{A} \oplus \mathbf{A}^2\end{aligned}$$

On cherche donc $\mathbf{A}^* = \lim_{k\to+\infty} \mathbf{A}^{(k)} = \mathbf{E}_\oplus \oplus \mathbf{A} \oplus \mathbf{A}^2 \oplus \cdots$.

On a $e_\otimes = e_+ = 0$, et un circuit C est absorbant s'il est de poids négatif : $e_\otimes \oplus w(C) = \mathsf{min}(0, w(C)) \neq 0$ si $w(C) \leq 0$. Si tous les poids des circuits sont positifs, il existe $p < n$ tel que $\mathbf{A}^* = \mathbf{A}^p$, ce qui permet de trouver tous les chemins de poids minimaux dans le graphe de n sommets en moins de n étapes de calcul matriciel.

Dans [de la Torre et Kruskal, 1995], les auteurs utilisent l'algèbre de chemins sur des listes de sommets, formant des mots, et ordonnés par l'ordre lexicographique. Ils obtiennent un algorithme pour le tri topologique et le parcours en profondeur sur les DAGs. Nous étendons ces résultats au chapitre 7 page 123. Au § 4.2 page 63, nous expliquons comment utiliser ces techniques pour des calculs distribués dans un réseau.

4.1.7 Généralisation de l'algèbre de chemins*

Dans [Gondran et Minoux, 1979], une structure plus générale que les dioïdes est présentée. Au lieu d'étiqueter l'arc (i, j) du graphe de précédence avec des scalaires fixés, on les étiquettes avec des fonctions h_{ij}, ce qui permet de considérer des problèmes où l'arrivée en j dépend de la date de départ de i : si la date de départ de i est notée t_i, la date d'arrivée en j sera $h_{ij}(t_j)$.

On considère donc l'ensemble $\mathbb{S}$ des valeurs des sommets et $\oplus$ une opération associative et commutative sur $\mathbb{S}$, d'élément neutre $e_\oplus$. On appelle $\mathbb{H}$ l'ensemble des endomorphismes de $\mathbb{S}$ utilisés comme étiquette des arcs : pour tout h de $\mathbb{H}$ et tous a et b dans $\mathbb{S}$, on a $h(a \oplus b) = h(a) \oplus h(b)$ et $h(e_\oplus) = e_\oplus$. L'opération $\oplus$ de $\mathbb{S}$ induit une opération associative et commutative sur $\mathbb{H}$ par $(h \oplus g)(a) = h(a) \oplus g(b)$. Elle admet l'élément neutre $h_\oplus \in \mathbb{H}$ défini par $h_\oplus(a) = e_\oplus$ pour tout $a \in \mathbb{S}$, car ainsi $h_\oplus \oplus h = h \oplus h_\oplus = h$. $(\mathbb{H}, \oplus)$ est donc un demi-groupe abélien.

On munit $\mathbb{H}$ de la loi $\otimes$ de composition de deux endomorphismes : $(h_1 \otimes h_2)(a) = h_2(h_1(a))$. Cette loi admet l'endomorphisme identité $h_\otimes : x \mapsto x$ comme élément neutre. $(\mathbb{H}, \otimes)$ est donc un demi-groupe.

On vérifie que les deux lois $\oplus$ et $\otimes$ de $\mathbb{H}$ sont distributives l'une par rapport à l'autre. De plus $h_\oplus$ est absorbant pour la composition de deux endomorphismes $\otimes$. Ainsi $(\mathbb{H}, \oplus, \otimes)$ est un demi-anneau. Si de plus toutes les étiquettes $h_{ij} \in \mathbb{H}$ vérifient $h_\otimes \oplus h_{ij} = h_\otimes$, le graphe ne présente pas de circuit absorbant et $\mathbf{A}^*$ existe (*cf.* § 4.1.6 page précédente).

Néanmoins, même lorsque $\mathbf{A}^*$ existe, elle ne peut pas toujours être calculée directement car les endomorphismes, ou leur composition, peuvent n'être connus que par leurs effets sur les éléments de $\mathbb{S}$. Au § 5.5 page 101, nous comparons certains aspects des r-opérateurs (définis dans le chapitre suivant) à cette généralisation de l'algèbre de chemins.

4.1.8 Les réseaux associatifs et l'algèbre max-plus

Intérêt de la comparaison. Nous avons souligné au chapitre 3 l'importance de l'idempotence dans notre étude. Les algèbres idempotentes ont été étudiées sous le terme générique d'algèbre max-plus, et la synthèse que nous venons de présenter sur ce domaine donne des bases que nous utilisons par la suite.

Ainsi la présentation des structures algébriques simples permet de resituer l'algèbre déformée des r-opérateurs que nous introduisons dans le chapitre suivant. Les structures matricielles permettent de faire le lien entre notre travail et l'algèbre de chemin. Nous pourrons récupérer des algorithmes de l'algèbre de chemins matriciel pour l'algorithmique des réseaux associatifs au chapitre 7 (algorithmes 20, 21, 22, 23, *cf.* tableau récapitulatif de la figure 111 page 151). De même, certains des algorithmes que nous développons pour les réseaux associatifs pourront, en retour, s'appliquer à l'algèbre de chemins matriciel (algorithmes 24, 25). La comparaison entre l'algèbre de chemins généralisée et les r-opérateurs est présentée au § 5.5 page 101.

Nous montrons également, dans la section suivante, que la matrice $\mathbf{A}^*$ peut être calculée par une itération asynchrone. Tout en soulignant les limites de la modélisation matricielle des calculs dans le réseau (rendue possible grâce à l'algèbre max-plus), nous montrons comment modéliser une `direct-association` par un tel calcul. Nous montrons aussi que la preuve de l'auto-stabilisation de la `direct-association` (*cf.* chapitre 6) peut être obtenue par la modélisation matricielle, de façon plus concise, mais avec des hypothèses plus fortes sur le réseau.

Nous commençons par préciser les deux applications possibles de l'algèbre max-plus à notre étude : contrôle et calculs. Nous utilisons cette algèbre uniquement pour les calculs. Nous soulignons alors les limites de la modélisation matricielle utilisée pour les aspects calculatoires. Nous proposons ensuite une modélisation de la `direct-association` supposée synchrone, avant de présenter les itérations asynchrones dans la section suivante. Celles-ci permettent de lever cette contrainte de synchronisation, qui n'existe pas dans la définition de la `direct-association`.

Aspect contrôle et calculatoire. L'algèbre max-plus permet d'étudier le contrôle de systèmes comparables aux réseaux associatifs, c'est-à-dire des ensembles d'entités non synchronisées. Généralement on suppose que les entités requièrent toujours le même délai pour accomplir leur tâche. On peut s'affranchir de cette difficulté par exemple avec la généralisation de l'algèbre des chemins présentée au § 4.1.7. Cependant, dans ces modélisations, les données manipulées se réfèrent à des dates et non à des quantités : on peut savoir que les trains seront synchronisés à telle date, mais on ne sait pas combien de personnes en descendront.

Ainsi, lors de l'étude du contrôle, les scalaires manipulés sont des dates : le vecteur $\mathtt{X}_k$ donne les dates de début de la $k^{\text{ème}}$ activité de chacune des entités du système. Ce n'est pas cet aspect que nous recherchons ici. Nous étudions le contrôle des réseaux associatifs avec d'autres outils au chapitre 6 (*cf.* § 4.3 page 70).

La seconde approche est l'étude des flux échangés entre des entités, semblables aux nœuds dans le réseau associatif. Les calculs des nœuds correspondent aux modifications de valeurs telles que le flot de spectateurs d'un cinéma à un autre, dans notre exemple introductif (§ 4.1.1 page 52). Cette approche est intéressante car elle modélise les aspects calculatoires du réseau, ce qui pourrait permettre d'étudier la terminaison d'une `direct-association`.

Restriction de la modélisation matricielle. La modélisation du calcul par une matrice est cependant réductrice, car elle induit des phases dans les calculs, qui correspondent au passage du vecteur $\mathtt{X}_k$ à $\mathtt{X}_{k+1}$. Entre chaque phase, chaque entité a évolué, ce qui correspond à la multiplication matricielle qui englobe tous les nœuds. En effet, si on modélise les flux du réseau, les scalaires manipulés correspondent aux flux, et les indices k des vecteurs d'état $\mathtt{X}_k$ donnent le numéro de la phase globale. La multiplication matricielle modélise donc une évolution synchrone du réseau, tandis que nous voudrions modéliser des calculs locaux asynchrones. En effet, dans un réseau associatif, les calculs globaux sont tels que, par exemple, un nœud peut réaliser un nombre arbitraire de fois son calcul local pendant qu'un autre ne l'exécutera qu'une seule fois.

Modélisation matricielle d'une `direct-association` (synchrone). Malgré sa limitation aux systèmes synchrones, la modélisation matricielle a le mérite de conduire à des calculs et des preuves concises. Nous proposons donc ici une modélisation du réseau associatif sous forme matricielle. Dans la section suivante, et en particulier au § 4.2.4 page 68, nous indiquons comment la contrainte de synchronisation des nœuds peut être levée. Nous utilisons des raisonnements matriciels aux § 6.2.6 page 117 et 7.7.3 page 150. Nous admettons donc (provisoirement) que les nœuds du réseau sont synchronisés.

La modélisation matricielle implique d'utiliser deux lois, alors qu'un seul opérateur est spécifié pour une $\oplus$-`direct-association`. Nous supposons ici que l'opérateur $\oplus$ est un s-opérateur[(o)] défini sur l'ensemble $\mathbb{S}$. Considérons donc une seconde loi $\otimes$ sur $\mathbb{S}$, telle que $(\mathbb{S}, \oplus, \otimes)$ soit un demi-anneau idempotent[(p)]. On considère la matrice de précédence $\mathbf{A}$ du réseau, dans laquelle toute absence de lien entre le sommet j et le sommet i est notée par $\mathbf{A}[i][j] = e_\oplus$, et toute présence par $\mathbf{A}[i][j] = e_\otimes$.

Les valeurs à l'étape k des registres b_{out} des nœuds sont notées par un vecteur $\mathtt{X}_k$. Le calcul global au cours d'une étape — qui correspond à une `step-association` — donne une nouvelle valeur à chaque nœud v (qui est stockée dans son registre $b_{\text{out}}[v]$), et donc un nouveau vecteur $\mathtt{X}_{k+1}$. Ce calcul est modélisé par

$$\mathtt{X}_{k+1} \quad = \quad \mathbf{A} \otimes \mathtt{X}_k \tag{4.3}$$

où $\otimes$ est la multiplication matricielle définie comme au § 4.1.4 page 57, en utilisant les lois scalaires $\oplus$ et $\otimes$ de $\mathbb{S}$. On remarque que l'opérateur $\otimes$ n'est utilisé ici que pour les besoins de la modélisation, et que seule la relation $x \otimes e_\otimes = x$ est nécessaire.

Si maintenant la boucle (i, i) de certains nœuds i est masquée, alors la composante $\mathbf{A}[i][i]$ est égale à $e_\oplus$. Mais le registre correspondant, $b^0_{\text{in}}[i]$, fournit tout de même une valeur au calcul sur i. Cette valeur est constante et correspond à la valeur initiale du nœud i. Nous introduisons un vecteur $\mathtt{B}$, dont la $j^{\text{ème}}$ composante contient $e_\oplus$, si la boucle (j, j) n'est pas masquée, et la valeur initiale du nœud j, sinon. La modélisation d'un pas de calcul (`step-association`) devient :

$$\mathtt{X}_{k+1} \quad = \quad \mathbf{A} \otimes \mathtt{X}_k \quad \oplus \quad \mathtt{B} \tag{4.4}$$

La `direct-association`, supposée ici synchrone, fournit un vecteur résultat $\mathtt{X}_l = \lim_{k \to +\infty} \mathtt{X}_k$, qui correspond aux valeurs des registres de sortie b_{out} des nœuds. En supposant que le réseau est synchrone, la stabilisation du calcul de la $\oplus$-`direct-association` peut donc être étudiée par les méthodes de point fixe matriciel (*cf.* § 4.1.5 page 58). La proposition suivante découle directement des applications de l'algèbre de chemins ; nous l'utilisons au § 4.2.4 page 68 pour étendre le résultat au cas asynchrone.

Proposition 23 *Soit $\oplus$ un* s*-opérateur. Si le calcul de la $\oplus$-`direct-association` est synchrone, alors il converge.*

Preuve. Puisque le poids $w(C)$ d'un circuit C est donné par $w(C) = e_\otimes \otimes \cdots \otimes e_\otimes = e_\otimes$, on a $e_\otimes \oplus w(C) = e_\otimes$ et il n'y a donc pas de circuit absorbant. La matrice $\mathbf{A}^*$ existe donc (*cf.* § 4.1.6 page 59) et il existe un point fixe à l'équation 4.4, qui est donné par $\mathbf{A}^* \otimes B$ (*cf.* § 4.1.5 page 58). La $\oplus$-`direct-association` converge donc. □

(o) Au § 5.5 page 101, nous indiquons comment adapter cette modélisation au cas d'un r-opérateur.
(p) La définition de ce second opérateur a peu d'importance ici car seul son élément neutre $e_\otimes$ nous intéresse.

4.2 Itérations Asynchrones

Dans cette section, nous présentons les travaux effectués dans le domaine des itérations asynchrones, qui constituent, à l'origine, un ensemble de techniques permettant d'optimiser les calculs sur les ordinateurs parallèles.

Deux raisons majeures justifient cette présentation. Tout d'abord nous avons conclu dans la section précédente sur les limitations du calcul matriciel pour un système synchrone : les phases du calcul induisent des synchronisations des entités calculantes du réseau. Or les itérations asynchrones, si elles s'appliquent aux multiplications matricielles, abolissent cependant la notion de phase définie par la multiplication elle-même.

La seconde raison est liée à l'usage de l'asynchronisme. Dans les itérations asynchrones, un gain certain est obtenu sur la vitesse de convergence du calcul en supprimant les contraintes impliquées par la découpe du calcul en phases. Ces techniques motivent l'usage de l'asynchronisme en data-parallélisme.

4.2.1 Introduction*

Beaucoup de problèmes de physique, d'optimisation combinatoire, ou d'algorithmique en général peuvent se ramener à l'équation $\mathtt{X} = \mathsf{F}(\mathtt{X})$, où $\mathtt{X}$ est un vecteur d'un ensemble à plusieurs dimensions $\mathbb{S}^n$, et F un opérateur de $\mathbb{S}^n$ dans $\mathbb{S}^n$. Ainsi modélisé, le problème revient à trouver le point fixe de l'opérateur F. Sur ordinateur, la technique classique est celle dite de la *relaxation du point fixe*, aussi appelée *méthode des approximations successives*, et qui consiste à calculer les termes d'une suite définie par $\mathtt{X}_{k+1} = \mathsf{F}(\mathtt{X}_k)$, en partant d'un vecteur initial $\mathtt{X}_0$ correctement choisi.

Dans un système parallèle, tel qu'un multi-processeur à mémoire partagée, on assigne à chaque processeur une ou plusieurs composantes du vecteur $\mathtt{X}_k$. Un processeur calcule donc une sous-partie du vecteur, à chaque itération. La stricte application de l'équation $\mathtt{X}_{k+1} = \mathsf{F}(\mathtt{X}_k)$ implique une synchronisation des processeurs entre chaque itération puisque, sauf pour certains cas triviaux, l'opérateur induit des dépendances de données entre les composantes. Or les synchronisations sont une des premières sources de ralentissement dans ce type de parallélisation [Axelrod, 1986]. Aussi a-t-on cherché à réduire autant que possible les dépendances de données et les synchronisations. La $k^{\text{ème}}$ itération pourra, par exemple, utiliser des composantes calculées à l'étape $k-2$ et d'autres calculées à l'étape $k-3$, au lieu de ne prendre les composantes qu'à l'étape précédente $k-1$, ce qui implique d'attendre qu'elles soient toutes disponibles. Les étapes ne sont plus à proprement parler des itérations, mais, par abus de langage, on continuera à parler d'itérations.

On peut remarquer que l'idée des itérations asynchrones était, dans un certain sens, déjà présente dans les méthodes de résolutions des systèmes linéaires dites itératives. En effet, comme nous allons l'expliquer, la méthode de Gauss-Seidel introduit une dé-synchronisation par rapport à celle de Jacobi. Afin de bien comprendre l'idée des itérations asynchrones, nous rappelons maintenant ces méthodes. Puis nous résumons les recherches dans le domaine des itérations asynchrones. Finalement, nous dressons le bilan de ce domaine et indiquons quelles sont ses connexions avec les réseaux associatifs.

4.2.2 Les méthodes de résolution itératives*

Cette courte présentation des méthodes itératives introduit le concept de relâchement de contraintes, qui favorise la convergence des calculs itératifs. Cet exposé est basé sur les références [Ciarlet, 1982] et [Reinhardt et Soeder, 1997].

Introduction. Le problème de base consiste à résoudre l'équation $\mathbf{A}\mathtt{X} = \mathtt{B}$, où $\mathbf{A}$ est une matrice inversible, $\mathtt{B}$ un vecteur donné, et $\mathtt{X}$ le vecteur inconnu, sur un espace vectoriel quelconque $\mathbb{K}^n$.

Si on peut décomposer la matrice $\mathbf{A}$ en deux matrices $\mathbf{M}$ et $\mathbf{N}$ de telle sorte que $\mathbf{A} = \mathbf{M} - \mathbf{N}$ et que $\mathbf{M}$ soit inversible, alors le problème passe de la résolution de l'équation $\mathbf{A}\mathtt{X} = \mathtt{B}$ à celle de $\mathtt{X} = \mathbf{M}^{-1}\mathbf{N}\mathtt{X} + \mathbf{M}^{-1}\mathtt{B}$. Le problème revient donc dans ce cas à trouver un point fixe à l'application $\mathsf{f} : \mathbb{K}^n \to \mathbb{K}^n$ définie par $\mathsf{f}(\mathtt{X}) = \mathbf{M}^{-1}\mathbf{N}\mathtt{X} + \mathtt{B}'$ (en posant $\mathtt{B}' = \mathbf{M}^{-1}\mathtt{B}$). Cette recherche peut alors être menée en utilisant la méthode des approximations successives, dite aussi méthode de Picard : on cherche la limite $\mathtt{X}_l$ de la suite définie par $\mathtt{X}_{k+1} = \mathbf{M}^{-1}\mathbf{N} \times \mathtt{X}_k + \mathtt{B}'$, partant d'un vecteur initial $\mathtt{X}_0$ donné.

On montre que cette suite converge si et seulement si[65] $\rho(\mathbf{M}^{-1}\mathbf{N}) < 1$, ce qui est également équivalent à avoir $||\mathbf{M}^{-1}\mathbf{N}|| < 1$ pour au moins une norme matricielle $||.||$. Cette condition revient à supposer que l'application $\mathtt{f}$ est une *contraction* relativement à la norme matricielle $||.||$ car on peut lui subordonner une norme vectorielle vérifiant $||\mathtt{f}(\mathtt{X}) - \mathtt{f}(\mathtt{Y})|| \leq ||\mathbf{M}^{-1}\mathbf{N}|| \times ||\mathtt{X} - \mathtt{Y}||$. On montre aussi que la suite $||\mathtt{X}_k - \mathtt{X}_l||$ converge d'autant plus vite que $\rho\left(\mathbf{M}^{-1}\mathbf{N}\right)$ est petit [Ciarlet, 1982]. L'étude des méthodes itératives se résume donc à la recherche de la matrice $\mathbf{M}^{-1}\mathbf{N}$ qui a le plus petit rayon spectral.

65. Rayon spectral ρ **:** plus grande des valeurs absolues des valeurs propres de la matrice.

Pour trouver la solution à l'équation $\mathbf{A}\mathtt{X} = \mathtt{B}$, on résout donc les systèmes linéaires successifs $\mathbf{M}\mathtt{X}_{k+1} = \mathbf{N}\mathtt{X}_k + \mathtt{B}$. Pour les méthodes de Jacobi et de Gauss-Seidel, la décomposition de $\mathbf{A}$ en deux matrices $\mathbf{M}$ et $\mathbf{N}$ est telle que les coefficients de la matrice $\mathbf{A}$ se répartissent entre $\mathbf{M}$ et $\mathbf{N}$ ($\mathbf{M}[i][j] = \mathbf{A}[i][j]$ ou 0 et ($\mathbf{N}[i][j] = -\mathbf{A}[i][j]$ ou 0). Cela signifie qu'il n'y a pas de "chevauchement"de $\mathbf{M}$ et de $\mathbf{N}$ (figure 56). De façon imagée, les méthodes itératives tentent de n'inverser que la partie de la matrice $\mathbf{A}$ qui se retrouve dans $\mathbf{M}$. On cherche donc une matrice $\mathbf{M}$ facilement inversible (diagonale, triangulaire, ...). Mais d'un autre côté, et de façon intuitive, on peut supposer que, plus la matrice $\mathbf{M}$ contient d'éléments de $\mathbf{A}$, plus la convergence sera rapide ; le cas extrême étant celui où $\mathbf{M} = \mathbf{A}$ et $\mathbf{N} = 0$ et où la convergence est obtenue en une itération.

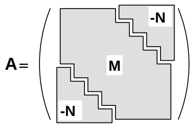


FIG. 56 – Décomposition de la matrice $\mathbf{A}$ pour les méthodes de Jacobi et Gauss-Seidel.

Méthode de Jacobi. Notons $\mathbf{D}$ la matrice composée uniquement des éléments diagonaux de $\mathbf{A}$ (et complétée par des 0 ailleurs), $\mathbf{E}$ la matrice composée des éléments de $\mathbf{A}$ situés sous la diagonale, et $\mathbf{F}$ celle composée des éléments de $\mathbf{A}$ restants, situés au-dessus de la diagonale (figure 57). On suppose de plus que les coefficients de la diagonale de $\mathbf{A}$ sont non nuls, ce qui assure que $\mathbf{D}$ est inversible.

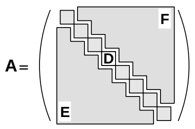


FIG. 57 – $\mathbf{A} = \mathbf{E} + \mathbf{D} + \mathbf{F}$.

La *méthode de Jacobi* est la méthode itérative pour laquelle la décomposition de $\mathbf{A}$ est la plus simple : $\mathbf{M} = \mathbf{D}$ et $\mathbf{N} = -(\mathbf{E} + \mathbf{F})$. Les systèmes linéaires à résoudre successivement sont donc $\mathbf{D}\mathtt{X}_{k+1} = -(\mathbf{E} + \mathbf{F})\,\mathtt{X}_k + \mathtt{B}$. Les calculs nécessaires à la résolution de ces systèmes, illustrés figure 58 peuvent se résumer ainsi ($k^{\text{ème}}$ itération et $j^{\text{ème}}$ ligne) :

$$\begin{aligned}\mathbf{A}[j][j] \times \mathtt{X}_{k+1}[j] = \quad & -\mathbf{A}[j][1] \times \mathtt{X}_k[1] - \cdots - \mathbf{A}[j][j-1] \times \mathtt{X}_k[j-1] \\ & -\mathbf{A}[j][j+1] \times \mathtt{X}_k[j+1] - \cdots - \mathbf{A}[j][n] \times \mathtt{X}_k[n]\end{aligned}$$

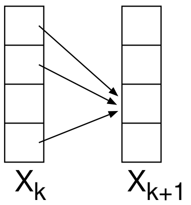


FIG. 58 – Itération de Jacobi.

Méthode de Gauss-Seidel. Dans le calcul précédent, pour fabriquer $\mathtt{X}_{k+1}[j]$, on utilise tous les $\mathtt{X}_k[i]$, pour i allant de 1 à n sauf j, car la diagonale est nulle dans la matrice $(\mathbf{E} + \mathbf{F})$. On peut cependant remarquer que, puisqu'il s'agit de la $j^{\text{ème}}$ composante, les $j - 1$ premières composantes du nouveau vecteur $\mathtt{X}_k$ sont déjà disponibles. La *méthode de Gauss-Seidel* consiste à utiliser dans le calcul du vecteur $\mathtt{X}_{k+1}$ les coefficients déjà calculés de ce même vecteur (figure 59) :

$$\begin{aligned}\mathbf{A}[j][j] \times \mathtt{X}_{k+1}[j] = \quad & -\mathbf{A}[j][1] \times \mathtt{X}_{k+1}[1] - \cdots - \mathbf{A}[j][j-1] \times \mathtt{X}_{k+1}[j-1] \\ & -\mathbf{A}[j][j+1] \times \mathtt{X}_k[j+1] - \cdots - \mathbf{A}[j][n] \times \mathtt{X}_k[n]\end{aligned}$$

La relation de récurrence sur les vecteurs $\mathtt{X}_k$ devient : $\mathbf{D}\mathtt{X}_{k+1} = -\mathbf{E}\mathtt{X}_{k+1} - \mathbf{F}\mathtt{X}_k + \mathtt{B}$, soit $\mathtt{X}_{k+1} = (\mathbf{D} + \mathbf{E})^{-1}(-\mathbf{F})\,\mathtt{X}_k + (\mathbf{D} + \mathbf{E})^{-1}\,\mathtt{B}$. Puisque les éléments de la diagonale de $\mathbf{A}$ sont supposés non nuls, la matrice $(\mathbf{D} + \mathbf{E})$ est bien inversible.

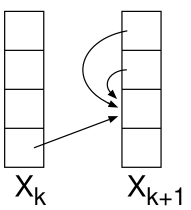


FIG. 59 – Itération de Gauss-Seidel.

Par rapport à la méthode de Jacobi où la matrice $\mathbf{M}$ est diagonale, dans la méthode de Gauss-Seidel $\mathbf{M}$ est triangulaire inférieure ($\mathbf{M} = (\mathbf{D} + \mathbf{E})$ et $\mathbf{N} = -\mathbf{F}$), et contient donc plus d'éléments de $\mathbf{A}$. On peut donc s'attendre, de façon intuitive, comme expliqué précédemment, à ce que cette méthode converge plus rapidement. On montre en effet que

$$\rho\left((\mathbf{E} + \mathbf{F})^{-1} \times (-\mathbf{F})\right) = \left(\rho\left(\mathbf{D}^{-1} \times (-\mathbf{E} - \mathbf{F})\right)\right)^2$$

ce qui signifie que, si l'une des méthodes converge, l'autre aussi, et que la méthode de Gauss-Seidel sera plus rapide.

Conclusion. Dans la méthode de Jacobi, toutes les composantes du vecteur sont calculées avant de passer à l'itération suivante. En quelque sorte, les calculs sont synchronisés entre chaque itération. Au contraire, la méthode de Gauss-Seidel utilise les composantes du vecteur dès que possible, ce qui conduit à remplacer cette synchronisation globale de toutes les composantes du vecteur par une dépendance (et donc synchronisation) de données, composante par composante. La séparation stricte entre les itérations de la méthode de Jacobi est donc levée, et on constate que la méthode gagne en vitesse.

4.2.3 Itérations asynchrones*

Les itérations asynchrones consistent à lever davantage de contraintes que celle de Jacobi, et à ne plus considérer ni synchronisation globale, ni même dépendance de données composante à composante.

Origines. Les premiers travaux parlent de *conduite libre*[66] (S. Schechter, 1962, rapporté dans [Miellou, 1975]). La conduite libre consiste à choisir librement les composantes du vecteur qui sont mises à jour à chaque étape. Dans leurs travaux, M. Charnay, F. Mussy et F. Robert parleront d'*itérations chaotiques série parallèle* (cité dans [Miellou, 1975]). La méthode des *relaxation chaotiques*[67] est introduite par Chazan et Miranker en 1969 (cité par [Baudet, 1978] notamment). Cette méthode revient à une conduite libre, additionnée d'un retard dans l'accès aux autres composantes du vecteur. En effet, chaque composante du vecteur $\mathtt{X}$ est calculée par un processeur différent. Les composantes dont un processeur aurait besoin sont en cours de calcul ou d'acheminement, et le processeur se contente alors de la dernière valeur reçue pour ces composantes.

La méthode de Chazan et Miranker s'applique au cas où l'opérateur F est linéaire sur $\mathbb{R}^n$ et défini par $\mathsf{F}(\mathtt{X}) = \mathbf{A}\mathtt{X} + \mathtt{B}$. Les auteurs montrent qu'il y a convergence des itérations vers le point fixe de F si le rayon spectral de la matrice $\mathbf{A}$ vérifie $\rho(|\mathbf{A}|) < 1$ ($|\mathbf{A}|$ est la matrice des valeurs absolues : $|\mathbf{A}|[i][j] = |\mathbf{A}[i][j]|$).

Miellou. Dans [Miellou, 1975], suivant les travaux de J.P.D. Donnelly (1971) qui parlait de relaxation retardée[68], l'auteur appelle les itérations chaotiques de Chazan et Miranker des *itérations chaotiques à retard*. Ces travaux généralisent ceux de Chazan et Miranker au cas des applications F non linéaires définies sur $\mathbb{E}$, un produit cartésien fini d'espaces de Banach[69] quelconques $\mathbb{E}_i$ de norme $||.||_i$ $(1 \leq i \leq \alpha)$.

Si $||.||$ désigne la norme vectorielle canonique[70] de $\mathbb{E}$, on définit la matrice $\mathbf{T}$ comme étant *majorante linéaire* de l'application F si, pour tous vecteurs $\mathtt{X}$ et $\mathtt{Y}$ de l'ensemble de définition de F, on a $||\mathsf{F}(\mathtt{X} - \mathtt{Y})|| \leq \mathbf{T}||\mathtt{X} - \mathtt{Y}||$. Si, de plus, $\mathbf{T}$ vérifie $\rho(\mathbf{T}) < 1$, alors $\mathbf{T}$ est appelée *matrice contractante* de l'opérateur F, qui est appelé dans ce cas *opérateur contractant*. Miellou montre que si F a un point fixe dans son domaine de définition (qui doit être d'intérieur non vide) et est contractante dans un voisinage de ce point fixe, alors la relaxation chaotique à retard converge, pourvu que les retards ne s'«aggravent» pas : une composante retardée calculée à l'étape j et utilisée à l'étape i doit vérifier **(a)** $j \in [i - \mathbf{s}(i), \ldots, i]$, avec $\mathbf{s}$ une fonction de $\mathbb{N}$ dans $\mathbb{N}$, **(b)** $i - \mathbf{s}(i)$ croît avec i, **(c)** $\mathbf{s}(i) < \min(i, s)$, pour un entier fixé s. L'auteur montre également que la convergence est d'autant plus rapide que le rayon spectral de $\mathbf{T}$ est petit.

Baudet. L'article [Baudet, 1978], s'il n'est pas le premier, semble cependant être fondateur dans le domaine des itérations asynchrones. Il apporte un formalisme nouveau qui synthétise et généralise les travaux précédents, et qui sera repris ensuite par d'autres auteurs.

Baudet s'intéresse au cas particulier du Banach réel $\mathbb{R}^n$, qui convient dans la plupart des applications numériques. Il définit les *itérations asynchrones* de la façon suivante. À partir d'un vecteur de

66. En anglais, *free steering.*

67. En anglais, *chaotic relaxation.*

68. En anglais, *delayed relaxation.*

69. Espace de Banach : espace vectoriel normé et complet, c'est-à-dire sur lequel toute suite de Cauchy (u_n) converge (une suite de Cauchy est définie par : $\forall \varepsilon > 0$, $\exists n_\varepsilon$ tel que si $m > n > n_\varepsilon$, alors $||u_m - u_n|| < \varepsilon$).

70. Norme canonique : si $\mathtt{X}$ est un vecteur de $\mathbb{E}$, $||\mathtt{X}||$ est l'élément de $(\mathbb{R}^+)^\alpha$, de $i^{\text{ème}}$ composante $||\mathtt{X}_i||_i$, où $\mathtt{X}_i$ désigne la composante de $\mathtt{X}$ dans $\mathbb{E}_i$, et où $||.||_i$ est la norme de $\mathbb{E}_i$.

$\mathtt{X}_0 \in \mathbb{R}^n$ donné, on définit une suite de vecteurs $\mathtt{X}_j$ par :

$$\mathtt{X}_j[i] = \begin{cases} \mathtt{X}_{j-1}[i] & \text{si } i \notin J_j \\ \mathsf{F}\left(\mathtt{X}_{\mathtt{S}_j[1]}[1], \ldots, \mathtt{X}_{\mathtt{S}_j[n]}[n]\right) & \text{si } i \in J_j \end{cases} \tag{4.5}$$

où $\mathcal{J} = \{J_j\}$ est une séquence de sous-ensembles non-vides J_j de $[1, \ldots, n]$, qui indique quels sont les indices mis à jour à chaque étape, et $\mathcal{S} = \{\mathtt{S}_j\}$ est une séquence de "vecteurs" de $\mathbb{N}^n$, qui donne les retards pour chacune des coordonnées de $\mathtt{X}_j$, et pour chacune des itérations. Les conditions suivantes doivent en outre être respectées pour tous les indices i : **(a)** $\forall j, \mathtt{S}_j[i] \leq j-1$, **(b)** $\lim_{j\to\infty} \mathtt{S}_j[i] = \infty$ et **(c)** i apparaît infiniment souvent dans J_j (il n'y a pas de «famine»). Les itérations asynchrones sont notées $(\mathsf{F}, \mathtt{X}_0, \mathcal{J}, \mathcal{S})$.

L'utilisation la plus intéressante de cette définition formelle est le cas pratique des itérations exécutées sur un réseau de processeurs gérant chacun une composante du vecteur, et où la règle est de prendre, pour chacune des composantes désirées, la dernière valeur produite et disponible.

On peut remarquer que les méthodes de Jacobi et Gauss-Seidel (*cf.* § 4.2.2) vérifient les conditions **(a)**, **(b)** et **(c)**. En effet, pour la méthode de Jacobi, $J_j = \{1, \ldots, n\}$ et $\mathtt{S}_j[i] = j-1$, ce qui signifie que toutes les composantes du vecteur sont mises à jour à chaque étape, en utilisant, pour chacune, la valeur qui a été calculée à l'étape précédente. La mise à jour de la méthode de Gauss-Seidel se traduit quant à elle par le singleton $J_j = \{1 + (k - 1 \mathsf{mod}\, n)\}$ et $\mathtt{S}_j[i] = j-1$, ce qui est équivalent à dire qu'une seule composante est mise à jour à la fois, en utilisant la dernière valeur produite pour chacune des composantes utilisées dans le calcul.

Le contrôle des retards dans le travail de Miellou revient à $j \in [i-s, \ldots, i]$. La condition **(b)** de Baudet est plus générale. De plus, toutes les relaxations chaotiques sont des itérations asynchrones mais le contraire est faux. En effet, il suffit de considérer l'exemple de l'opérateur $\mathsf{F} : \mathbb{R}^2 \to \mathbb{R}^2$, calculé sur un réseau de deux processeurs, dont le premier calcule deux fois plus lentement que le second. Ainsi $j - \mathtt{s}_2(j)$ augmente sans cesse et cette itération asynchrone n'est pas une relaxation chaotique car elle viole la condition **(b)** de Miellou, tandis qu'elle respecte les conditions **(a)**, **(b)** et **(c)** de Baudet.

L'auteur constate qu'une condition de Lipschitz[71] sur F n'est pas suffisante pour assurer la convergence vers un point fixe, ni même l'existence de ce dernier. Il suffit en effet de considérer sur $\mathbb{R}$ l'opérateur $\mathsf{F}(\mathtt{X}) = \sqrt{(\mathtt{X}^2 + a^2)}$, qui, bien que lipschitzien pour toutes les valeurs de a, n'a pas de point fixe pour $a = 1$, et en a une infinité pour $a = 0$. Baudet définit alors un opérateur *contractant* comme étant lipschitzien sur $D \subset \mathbb{R}^n$ avec une matrice lipschitzienne $\mathbf{L}$ de rayon spectral $\rho(L) < 1$.

71. Condition de Lipschitz : $\mathsf{F} : \mathbb{R}^n \to \mathbb{R}^n$ est lipschitzienne sur $D \subset \mathbb{R}^n$ si $|\mathsf{F}(\mathtt{X}) - \mathsf{F}(\mathtt{Y})| \leq \mathbf{L} \times |\mathtt{X} - \mathtt{Y}|$ pour tout $x, y \in D$, où $\mathbf{L}$ est une matrice carrée $n \times n$ non négative appelée *matrice lipschitzienne* de F et $|\mathtt{V}|$ est le vecteur des valeurs absolues des composantes de $\mathtt{V}$.

Lorsque l'opérateur F est linéaire ($\mathsf{F}(\mathtt{X}) = \mathbf{A}\mathtt{X} + \mathtt{B}$), il est contractant si et seulement si $\rho(|\mathbf{A}|) < 1$. On retrouve là le résultat de Chazan et Miranker. Quand F n'est pas linéaire, des méthodes "superlinéaires" ont été développées, telle que celle de Newton, définie par $\mathtt{X}_{i+1} = \mathsf{F}(\mathtt{X}_i) = \mathtt{X}_i - [\mathsf{F}'(\mathtt{X}_i)]^{-1} \times \mathsf{F}(\mathtt{X}_i)$. Il y a convergence vers la racine ξ de F si le vecteur initial $\mathtt{X}_0$ est suffisamment proche de ξ (ce qui implique de trouver un bon point de départ). Baudet prouve que si F' satisfait une condition de Lipschitz sur un voisinage de ξ, alors il s'agit d'un opérateur contractant. On peut généraliser ce résultat, ce qui finit de montrer que les opérateurs contractants définissent une grande classe d'applications.

Baudet montre le résultat suivant : si F est contractant sur un fermé D de $\mathbb{R}^n$, et si $\mathsf{F}(D) \subset D$, alors toute itération asynchrone $(\mathsf{F}, \mathtt{X}_0, \mathcal{J}, \mathcal{S})$ telle que $\mathtt{X}_0 \subset D$ converge vers l'unique point fixe de $\mathtt{F}$ dans D. En quelque sorte, le domaine se réduit strictement à chaque itération.

Üresin et Dubois. Dans l'article [Üresin et Dubois, 1989], les auteurs ont généralisé les résultats de Baudet, démontrés sur l'ensemble infini et continu $\mathbb{R}^n$, au cas d'un ensemble $\mathbb{S}^n$, où $\mathbb{S}$ est un ensemble fini ou non, dénombrable ou non. Les résultats sont complétés dans [Üresin et Dubois, 1990].

Üresin et Dubois appellent *opérateur contractant asynchrone*[72] tout opérateur F de $\mathbb{S}^n$ dans $\mathbb{S}^n$ défini sur le domaine $D(0) \subset \mathbb{S} = \mathbb{S}_1 \times \cdots \times \mathbb{S}_n$, tel que la séquence de domaines $\{D(k)\}$ vérifie les conditions suivantes : **(a)** $\forall k \in \mathbb{N}$, $D(k) = D_1(k) \times \cdots \times D_n(k)$, **(b)** $\exists m \in \mathbb{N}$ tel que $D(k+1) \subset D(k)$ pour $0 \leq k \leq m$, et $D(k) = \{\xi\}$ pour $k \geq m$, et **(c)** si le vecteur $\mathtt{X}$ est dans le domaine $D(k)$,

72. En anglais, *asynchronously contracting operator (ACO).*

alors $\mathsf{F}(\mathtt{X})$ est dans le domaine $D(k+1)$. Les auteurs montrent que ξ est alors le seul point fixe de l'opérateur F sur son domaine $D(0)$, et que toute itération asynchrone $(\mathsf{F}, \mathtt{X}_0, \mathcal{J}, \mathcal{S})$ convergence vers ce point fixe, pourvu que le vecteur initial $\mathtt{X}_0$ soit bien dans le domaine initial $D(0)$.

Dans [Üresin et Dubois, 1990], les auteurs donnent une condition nécessaire dans le cas où l'ensemble $\mathbb{S}$ est fini : si toute itération asynchrone $(\mathsf{F}, \mathtt{X}_0, \mathcal{J}, \mathcal{S})$, partant de $\mathtt{X}_0 \in \mathbb{S}^n$, converge vers ξ, pour tous les ensembles $\mathcal{J}$ des indices successivement mis à jour, et pour tous les ensembles $\mathcal{S}$ des vecteurs des retards, alors l'opérateur F est contractant asynchrone sur un domaine $D(0) \subseteq \mathbb{S}$, avec $\mathtt{X}_0 \in D(0)$.

Les auteurs donnent ensuite des conditions sur la convergence des itérations asynchrones, qui ne sont que suffisantes mais qui peuvent être plus simples à vérifier dans la pratique. Ainsi, lorsque chaque ensemble $\mathbb{S}_i$ admet une relation d'ordre partiel $\preceq_i$, que l'on étend en une relation d'ordre $\preceq$ sur $\mathbb{S}$, les conditions suivantes assurent la convergence des itérations asynchrones vers le point fixe ξ de F : **(d)** l'opérateur F est clos[73] sur $D(0)$, **(e)** les itérations synchrones $\mathtt{Y}_k = \mathsf{F}(\mathtt{Y}_{k-1})$ convergent en décroissant (par rapport à $\preceq$, *i.e.*, $\mathtt{Y}_{k+1} \preceq \mathtt{Y}_k$), et **(f)** F est croissante sur $D(0)$ (par rapport à $\preceq$: $\mathtt{X}_1 \preceq \mathtt{X}_2 \Rightarrow \mathsf{F}(\mathtt{X}_1) \preceq \mathsf{F}(\mathtt{X}_2)$.

La condition **(e)** peut être remplacée, lorsque l'ensemble initial $D(0)$ est fini, par **(e')** F n'est pas expansif[74] sur $D(0)$.

En remplaçant la condition **(e')** de non expansion par la condition **(e")** $\mathsf{F}(\mathtt{X})[i] \prec_i \mathtt{X}[i]$ si $\mathtt{X}[i] \neq \xi[i]$ et $\mathsf{F}(\mathtt{X})[i] = \xi[i]$ sinon, on peut se passer de la condition de monotonie **(f)**.

Pour finir, les auteurs indiquent que la condition **(a)** du produit cartésien d'ensembles dans la définition des opérateurs contractants asynchrones peut être simplifiée dans certains cas. L'intérêt du produit cartésien d'ensembles réside dans le fait qu'ainsi, on est assuré que toute nouvelle valeur du vecteur $\mathtt{X}$, obtenue après une itération, tombe encore dans le domaine. Lorsqu'une composante peut être calculée par plusieurs processeurs en même temps — ce qui peut arriver quand l'attribution des composantes aux vecteurs est faite dynamiquement (*i.e.*, au cours de l'exécution) —, cette condition est en effet nécessaire. Mais lorsqu'à chaque composante du vecteur est associé un seul processeur, elle n'est plus nécessaire.

Ainsi, si $D(0)$ est fini et si les ensembles J_j des composantes mises à jour à chaque étape sont les singletons $\{1 + (k - 1 \bmod n)\}$, les conditions suivantes sur l'opérateur F sont suffisantes pour assurer la convergence des itérations asynchrones vers son point fixe ξ : **(d)** l'opérateur F est clos et **(e")** $\mathsf{F}(\mathtt{X})[i] \prec_i \mathtt{X}[i]$ si $\mathtt{X}[i] \neq \xi[i]$ et $\mathsf{F}(\mathtt{X})[i] = \xi[i]$ sinon. Ces conditions sont assez facilement vérifiées dans la pratique : elles signifient que le nombre de valeurs qu'il est possible de produire est fini, et que les composantes des vecteurs $\mathtt{X}_k$ sont de plus en plus petites, sauf quand elles ont atteint la solution.

Autres travaux. Dans [El Baz et al., 1996], la méthode des itérations asynchrone est affinée. Les auteurs autorisent des mises à jour partielles des calculs au lieu d'attendre que le processeur ait fini son calcul courant pour communiquer. Ils définissent ainsi les *itérations asynchrones à communications flexibles*.

Intérêt des itérations asynchrones. Les itérations asynchrones illustrent l'intérêt de l'asynchronisme au niveau de l'exécution des programmes. L'élimination des contraintes de dépendances de données ont conduit à supprimer le contrôle du calcul par les synchronisations. Il en résulte un gain de temps significatif.

Ainsi, dans [Baudet, 1978], l'auteur constate, par des expériences sur 6 processeurs du Cmmp[75], que 57% du temps des processeurs est passé en attente des synchronisations pour la méthode de Jacobi.

Dans [Chajakis et Zenios, 1991], les auteurs appliquent les itérations asynchrones au problème du flot dans un réseau. Ils obtiennent une accélération de 7,2 pour la version asynchrone contre 6,2 pour pour la version synchrone sur un Alliant FX/8[76].

Le même problème est étudié dans [El Baz et al., 1996], où les auteurs présentent des travaux assez complets sur les itérations asynchrones, afin d'étudier l'impact des «communications flexibles»

73. Opérateur clos : F est clos sur $D(0)$ si $\forall \mathtt{X} \in D(0)$, $\mathsf{F}(\mathtt{X}) \in D(0)$.

74. Opérateur non-expansif : F est non-expansif sur $D(0)$ si $\forall \mathtt{X} \in D(0)$, $\mathsf{F}(\mathtt{X}) \preceq \mathtt{X}$.

75. Le Cmmp a été développé à l'université Carnegie Mellon en 1971. Il est l'un des ancêtres de l'Illiac IV [de Rumeur, 1994]. Il est composé de 16 processeurs (PE) à base de PDP-11 ayant chacun 8ko de mémoire et un cache. Les PEs sont reliés à 16 bancs mémoires de 2Mo chacun par un réseau crossbar [Sansonnet, 1994].

(mises à jour partielle avant la fin du calcul courant). Les résultats des expériences effectuées sur une machine à mémoire partagée[77] donnent toujours l'avantage aux itérations asynchrones par rapport aux itérations synchrones. En particulier, lorsque le réseau étudié est irrégulier, l'équilibrage de la charge est mauvais. Aussi certains processeurs restent longtemps inactifs en attendant la synchronisation, qui est retardée par les processeurs surchargés. Le gain dû aux itérations asynchrones est dans ce cas important.

Dans [Üresin et Dubois, 1996], les auteurs s'intéressent au temps de convergence des itérations asynchrones indépendamment du temps des synchronisations, dans le cas d'une allocation dynamique des tâches sur les processeurs (ce qui permet d'équilibrer la charge). Leurs résultats sont obtenus par simulation. Ils constatent que le gain des itérations asynchrones apparaît surtout quand le calcul d'une composante du vecteur $\mathtt{X}_k$ fait intervenir peu de composantes du vecteur précédent (*i.e.*, il y a une faible dépendance de données). Le gain est significatif également lorsque les composantes de $\mathtt{X}_{k-1}$ nécessaires au calcul de $\mathtt{X}_k$ ne sont pas prévisibles. En effet, il est dans ce cas impossible de faire des synchronisations partielles, et les barrières de synchronisation globales — qui ralentissent fortement l'application — sont alors inévitables.

En outre, les auteurs remarquent que, même lorsque les itérations asynchrones convergent plus lentement que les itérations synchrones (en cas de forte dépendance de données d'une itération à une autre), la différence entre les deux méthodes s'amenuise à mesure que le nombre de processeurs augmente. Les itérations asynchrones finissent ainsi par être moins de deux fois plus lentes que les itérations synchrones, sans que le temps nécessaire aux synchronisations ait été pris en compte dans ces comparaisons. Or ce temps croît, lui, avec le nombre de processeurs. Les itérations asynchrones finissent donc toujours par être plus rapides que les itérations synchrones, à partir d'un nombre suffisant de processeurs.

En conclusion, l'introduction de l'asynchronisme dans les calculs matriciels améliore sensiblement leurs performances.

4.2.4 Les itérations asynchrones et les réseaux associatifs

Asynchronisme de mise en œuvre. Le principe des itérations asynchrones est le relâchement de contraintes dans un calcul matriciel. Comme nous l'avons indiqué au § 4.2.2, on peut attribuer l'origine de ce principe à la méthode de Gauss-Seidel, qui améliore la vitesse de convergence de la méthode de Jacobi en utilisant les données produites dès que possible. Les travaux présentés ensuite généralisent les relâchements de contraintes. Après Miellou et les relaxations chaotiques, Baudet définit les itérations asynchrones, plus générales, pour les opérateurs de $\mathbb{R}^n$. Üresin et Dubois étendent ensuite les résultats de Baudet au cas des ensembles discrets éventuellement finis, et donnent des conditions simples assurant la convergence des itérations asynchrones vers leurs points fixes pour les ensembles de définition finis et infinis.

Un calcul exprimé sous forme matricielle est conçu en imaginant des phases globales dans lesquelles toutes les composantes du vecteur sont mises à jours en même temps. Cela revient à construire l'algorithme avec des «calculs locaux» (ne concernant qu'une composante du vecteur) de façon synchrone. Mais les itérations asynchrones permettent de traduire cette conception synchrone en exécution asynchrone, plus avantageuse. Nous appelons cet asynchronisme *asynchronisme de mise en œuvre*.

Ces travaux concernent les recherches de performance des calculs parallèles sur PRAM. Ils motivent en quelque sorte la recherche de techniques similaires pour le modèle data-parallèle, dans lequel tous les calculs locaux sont synchronisés, au sein d'une primitive de calcul globale. Cependant, pour désynchroniser les calculs locaux d'une primitive data-parallèle, il faut que cette dernière inclut plusieurs calculs pour le même nœud. En effet, si elle ne comporte qu'un seul calcul par donnée élémentaire, ces calculs locaux seront implicitement synchronisés par la phase de calcul globale que représente la primitive data-parallèle.

En conclusion, une primitive data-parallèle incluant plusieurs calculs locaux par données élémentaires permettrait d'envisager la désynchronisation des calculs locaux, et ainsi de profiter d'un asynchronisme de mise en œuvre. Mais les itérations asynchrones intéressent notre travail à un autre

76. L'Alliant FX/8 a 12 processeurs pour l'interface utilisateur (PI), et 8 processeurs de type pipeline pour le traitement numérique flottant (PE) [Sansonnet, 1994]. La mémoire est partagée ; elle est organisée en bancs reliés aux PI et aux PE par un bus central et des caches. De plus, un réseau crossbar relie les PE et le bus mémoire via des caches.

77. La machine utilisée comporte de 16 à 32 transputers T800, comprenant un processeur, une unité flottante, une mémoire rapide et quatre liens de communications bi-directionnels. Les communications sont faites par accès direct à la mémoire. Plusieurs topologies sont envisageables grâce au réseau crossbar C004 d'Inmos ; les expériences ont été faites avec une topologie en forme de pipeline linéaire bidirectionnel.

titre : elles permettent de lever la contrainte de synchronisation des calculs matriciels, que nous avions évoquée au § 4.1.8. Nous revenons maintenant sur ce point.

Calcul asynchrone de $\mathbf{A}^*$. L'importance de la matrice $\mathbf{A}^*$ (définie par l'équation 4.1 page 58) est capitale en algèbre max-plus, notamment pour la résolution des équations (§ 4.1.5) et pour l'algèbre de chemins (§ 4.1.6). Dans [Üresin et Dubois, 1990], les auteurs mentionnent l'application de leurs travaux sur les itérations asynchrones, en se référant au problème des chemins de poids minimum présenté dans [Aho et al., 1974] (nous avons traité ce problème dans l'exemple du § 4.1.6).

Nous proposons ici une généralisation de l'application des itérations asynchrones au calcul de la matrice $\mathbf{A}^*$ dans le cas d'une demi-algèbre idempotente. Le théorème ci-dessous indique que lorsque la matrice $\mathbf{A}^*$ est calculée sur une machine parallèle de type PRAM, il n'y a pas besoin de synchronisation. Les mêmes avantages que ceux que nous avons mentionnés à propos des itérations asynchrones peuvent alors être espérés.

Théorème 24 *Lorsque la matrice $\mathbf{A}^*$ définie dans une demi-algèbre idempotente existe, son calcul peut être effectué en utilisant une itération asynchrone.*

Preuve. On considère le calcul de la matrice $\mathbf{A}^* = \lim_{k\to+\infty} \mathbf{A}^{(k)} = \mathbf{E}_\otimes \oplus \mathbf{A} \oplus \cdots \oplus \mathbf{A}^k \oplus \cdots$ dans la demi-algèbre idempotente $(\mathbb{M}, \oplus, \otimes, \cdot)$ sur le demi-corps idempotent $(\mathbb{S}, \oplus, \otimes)$ (*cf.* § 4.1.4). Nous nous plaçons dans le cas où $\mathbf{A}^*$ existe, c'est-à-dire qu'il n'y a pas de circuit absorbant dans le graphe de précédence $G[\mathbf{A}]$ associé (*cf.* § 4.1.6). Le calcul de $\mathbf{A}^*$ peut s'écrire à l'aide de l'opérateur matriciel F défini par :

$$\begin{array}{rrcl} \mathsf{F} : & \mathbb{M} & \to & \mathbb{M} \\ & \mathbf{B} & \mapsto & \mathsf{F}(\mathbf{B}) = \mathbf{B} \quad \oplus \quad \mathbf{B} \otimes \mathbf{A} \end{array} \tag{4.6}$$

En effet, on a $\mathsf{F}\left(\mathbf{A}^{(k)}\right) = \mathbf{A}^{(k+1)}$:

$$\begin{aligned} \mathsf{F}\left(\mathbf{A}^{(k)}\right) &= \mathbf{E}_\otimes \oplus \mathbf{A} \oplus \cdots \oplus \mathbf{A}^k \oplus \left(\mathbf{E}_\otimes \oplus \mathbf{A} \oplus \cdots \oplus \mathbf{A}^k\right) \otimes \mathbf{A} \\ &= \mathbf{E}_\otimes \oplus \mathbf{A} \oplus \cdots \oplus \mathbf{A}^k \oplus \mathbf{A}^{k+1} \qquad \text{(idempotence de } \oplus) \\ &= \mathbf{A}^{(k+1)} \end{aligned}$$

et $\mathsf{F}(\mathbf{A}^*) = \mathbf{A}^* \oplus \mathbf{A}^* \otimes \mathbf{A} = \mathbf{A}^*$.

Puisque nous avons supposé qu'il n'y avait pas de circuit absorbant dans $G[\mathbf{A}]$, les poids des chemins resteront supérieurs[(q)] au poids minimum d'un arc, qui est donné par $w_{\mathsf{min}} = \bigoplus_{i,j} \mathbf{A}[i][j]$. L'opérateur F donne les poids des chemins de longueur 1, puis 2, etc. dans le graphe $G[\mathbf{A}]$. Il peut donc être défini sur l'ensemble $D(0)$, éventuellement infini, défini par $D(0) = D_{ij}(0)^{n\times n} = [w_{\mathsf{min}}, \dots, e_\oplus]^{n\times n} \subset \mathbb{S}^{n\times n}$. En effet, F transforme les $n \times n$ scalaires de $D_{ij}(0)$ d'une matrice $\mathbf{B}$ de $\mathbb{M}$ en les $n \times n$ autres scalaires de $D_{ij}(0)$ de $\mathsf{F}(\mathbf{B})$.

F est clos sur $D(0)$, et on a $\mathbf{A} \in D(0)$. C'est la condition **(d)** de Üresin et Dubois présentée dans la section précédente. Il nous reste à vérifier les conditions **(e)** et **(f)** (les conditions plus simples **(e')** et **(e")** ne peuvent s'appliquer ici car $D(0)$ n'est pas forcément fini).

La condition **(e)** concerne la convergence des itérations synchrones. Puisque nous avons supposé qu'il n'y avait pas de circuit absorbant, $\mathbf{A}^*$ existe (§ 4.1.6), ce qui signifie que les itérations synchrones définies par $\mathbf{Y}_{k+1} = \mathsf{F}(\mathbf{Y}_k)$ et $\mathbf{Y}_0 = \mathbf{E}_\oplus$ convergent.

La condition **(f)** concerne la croissance de F. Supposons que $\mathbf{B}_1 \preceq_\oplus \mathbf{B}_2$. On a donc $\mathbf{B}_1 \otimes \mathbf{A} \preceq_\oplus \mathbf{B}_2 \otimes \mathbf{A}$, d'où $\mathbf{B}_1 \oplus \mathbf{B}_1 \otimes \mathbf{A} \preceq_\oplus \mathbf{B}_2 \oplus \mathbf{B}_2 \otimes \mathbf{A}$. Ainsi, si $\mathbf{B}_1 \preceq_\oplus \mathbf{B}_2$, on a

(q) La loi $\oplus$ du demi-corps idempotent $(\mathbb{S}, \oplus, \cdot)$ est associative, commutative et idempotente. Elle définit donc un ordre partiel $\preceq_\oplus$ sur $\mathbb{S}$, qui permet de comparer les scalaires. Cet ordre est étendu aux matrices : il s'agit de la comparaison terme à terme de deux matrices. La notation d'intervalles est abusive dans le cas d'un ordre partiel. Nous la conservons cependant car Üresin et Dubois utilisent le produit cartésien d'intervalles pour préciser que les bornes du domaine de définition sont indépendantes des exécutions. C'est le cas ici.

$\mathsf{F}(\mathbf{B}_1) \preceq_\oplus \mathsf{F}(\mathbf{B}_2)$, et F est croissant par rapport à la relation d'ordre matricielle $\preceq_\oplus$ définie par la comparaison terme à terme des scalaires de la matrice.

L'ensemble de définition $D(0)$ de F est donc un produit cartésien d'ensembles, il est ordonné, et les conditions **(d)** (F est clos sur $D(0)$), **(e)** (les itérations synchrones convergent) et **(f)** (F est croissante) sont vérifiées. Le calcul de $\mathbf{A}^*$ peut donc être effectué par une itération asynchrone. □

Modélisation matricielle d'une `direct-association`. Au § 4.1.8, l'algèbre max-plus a permis de modéliser par un calcul matriciel une `direct-association` dans un réseau associatif supposé synchrone. Pour que la modélisation matricielle reste valable avec un réseau asynchrone, il faudrait que certaines composantes du vecteur $\mathtt{X}_k$, qui note les valeurs des nœuds, puissent être mises à jour indépendamment des autres : les composantes du vecteur évolueraient sans aucune concertation.

Cela correspond à une itération asynchrone. La convergence du calcul distribué asynchrone de la $\oplus$-`direct-association` revient donc à la convergence de l'itération asynchrone correspondante, qui peut s'écrire comme suit, d'après les équations 4.4 page 62 et 4.5 page 66 :

$$\mathtt{X}_j[i] = \begin{cases} \mathtt{X}_{j-1}[i] & \text{si } i \notin J_j \\ \mathsf{F}\left(\mathtt{X}_{\mathsf{S}_j[1]}[1], \dots, \mathtt{X}_{\mathsf{S}_j[n]}[n]\right) = \mathbf{A} \otimes \left(\mathtt{X}_{\mathsf{S}_j[1]}[1], \dots, \mathtt{X}_{\mathsf{S}_j[n]}[n]\right)^t \oplus \mathtt{B} & \text{si } i \in J_j \end{cases} \tag{4.7}$$

Le théorème précédent concerne le calcul optimisé de la matrice $\mathbf{A}^*$, mais il ne peut être appliqué directement à l'étude de la convergence de l'itération asynchrone de l'équation 4.7 (l'opérateur agit ici sur des vecteurs, et non sur les matrices). Nous proposons maintenant un raisonnement similaire au théorème 24 pour montrer cette convergence. Cette proposition abolit la contrainte de synchronisation du calcul global mentionnée dans la proposition 23 page 62. Il montre que la `direct-association` se termine sur tout réseau lorsque l'opérateur utilisé est un s-opérateur. Notons qu'une preuve «directe» est donnée au § 3.3.2 page 40 et qu'une preuve par réduction d'expressions est donnée au § 5.1.3 page 80. Nous faisons également le lien entre ce résultat et les travaux de Tel au § 6.1.4 page 107. Nous utilisons cette proposition aux chapitres 5 et 7.

Proposition 25 *Soit $\oplus$ un* s*-opérateur défini sur $\mathbb{S}$. L'itération asynchrone donnée par l'équation 4.7, qui correspond au calcul distribué asynchrone de la $\oplus$-`direct-association`, converge, quel que soit le réseau sous-jacent.*

Preuve. La preuve est semblable à celle du théorème 24 et étend la proposition 23, dont nous reprenons les notations. L'opérateur F est clos sur $D(0) = D_{ij}(0)^n = [w_{\mathsf{min}}, \dots, e_\oplus]^n \subset \mathbb{S}^n$. La convergence des itérations synchrones est donnée par la proposition 23. L'opérateur F est croissant sur son domaine : soit $\mathtt{X}_1$ et $\mathtt{X}_2$ deux vecteurs de $D(0)$ tels que $\mathtt{X}_1 \prec_\oplus \mathtt{X}_2$. Tous les éléments de $\mathbf{A}$ vérifient $e_\otimes \prec_\oplus \mathbf{A}[i][j]$ et on a : $\mathbf{A} \times \mathtt{X}_1 \prec_\oplus \mathtt{X}_2$, d'où $\mathbf{A} \times \mathtt{X}_1 \oplus \mathtt{B} \prec_\oplus \mathtt{X}_2 \oplus \mathtt{B}$. Les conditions **(d)**, **(e)** et **(f)** de Üresin et Dubois sont donc vérifiées, et l'itération asynchrone converge. □

4.3 Systèmes distribués

Dans cette section, nous proposons une rapide introduction aux systèmes distribués[(r)]. Ce domaine concerne les systèmes calculants et communicants, et leur étude. Aussi la comparaison avec

(r) Certains auteurs font la distinction entre les termes *systèmes distribués* et *systèmes répartis* [Delaët, 1995]. Le premier, de l'anglais *distributed systems*, induiraient l'idée d'une distribution centralisée des tâches, tandis que le second supposerait une coopération des tâches, en vue de la répartition du travail. Nous ne faisons pas cette distinction et employons les deux termes indifféremment.

les réseaux associatifs est naturelle. Elle permet de mieux cerner certains concepts des réseaux associatifs, tels que l'asynchronisme, ou les modes de communications.

Nous utilisons des résultats et des méthodes de ce domaine pour l'étude de certaines propriétés du modèle, relatifs au contrôle du réseau (détection de terminaison, propriétés minimales des liens, pannes, etc.) au chapitre 6. De plus, les réseaux associatifs peuvent être considérés comme un système distribué à part entière. De plus, les r-opérateurs que nous définissons au chapitre 5, et les algorithmes que nous présentons au chapitre 7, peuvent être utilisés dans un contexte distribué, comme un réseau de processeurs reliés par un réseau de communication.

La présentation qui suit est volontairement incomplète ; elle est suffisante pour la suite de l'étude des réseaux associatifs. Elle est basée sur les références [Tel, 1991], [Tel, 1994], [Lamport et Lynch, 1990] et [Lynch, 1996].

4.3.1 Présentation*

Origines. L'étude des systèmes distribués émerge dans les années soixante, suite à deux problèmes complémentaires. Tout d'abord, le développement de l'usage de l'ordinateur accroît le besoin de communication entre les machines. Ensuite, pour augmenter la puissance des ordinateurs, l'idée de les connecter se développe [Tel, 1991]. Le besoin de communication d'une part, et celui de calcul d'autre part, conduit donc à faire coopérer des machines distantes les unes des autres par le biais des communications. C'est le concept de *computer network* [Tel, 1994].

Un système distribué peut être défini comme un réseau d'entités calculantes ayant le même but commun : celui de la réalisation d'une tâche globale à laquelle chaque entité contribue par ses calculs locaux. La notion de système «distribué», c'est-à-dire réparti dans l'espace, est relative ; le concepteur d'une machine pensera qu'elle est effectivement composée d'éléments distribués et coopérant par des communications (au sens large). Mais l'utilisateur de cette même machine pensera qu'il s'agit d'un système non distribué, et que le système distribué est l'assemblage de plusieurs de ces machines. Le but du domaine des systèmes distribués est de donner à l'utilisateur une vue non distribuée d'un environnement composé d'éléments répartis dans l'espace, et communicants [Lamport et Lynch, 1990].

La construction de tels systèmes n'est pas simple, à cause du caractère intrinsèquement distribué de la répartition des informations, et de leurs évolutions concurrentes. En particulier, les concepteurs se trouvent confrontés aux problèmes des pannes, de la détection de la terminaison d'un calcul global, des inter-blocages, des exclusions mutuelles, de la cohérence des données réparties, de la dissémination efficace de l'information, ou, au contraire, de sa centralisation rapide, etc.

Paramètres. On peut classer les systèmes distribués par leurs indéterminés [Lynch, 1996] : nombre de processeurs, topologie, codes uniformes (identiques d'un nœud à un autre) ou non, évolutions concertées des processeurs ou non, *etc.* Au niveau des communications, on se demandera si le temps de délivrance d'un message est incertain ou bien fixé, si l'ordre des messages sur un lien est connu ou aléatoire, etc. Dans [Lamport et Lynch, 1990], les auteurs pointent quatre paramètres essentiels : la topologie du réseau, la synchronisation des nœuds, les pannes tolérées et le mode de stockage des messages.

Un processeur[(s)] est essentiellement caractérisé par son programme, ses variables (et la place mémoire qu'elles requièrent, *i.e.*, nombre d'états ou quantité de bits), et par sa connaissance de l'extérieur (possibilité de distinguer ses voisins ou non, par exemple). Pour une entité calculante, les «messages» (au sens large) sont les seules façons de recevoir de l'information sur l'état du système, sur les autres entités, *etc.* Le réseau est constitué des liens qui véhiculent les messages d'une entité calculante à une autre. Il s'agit là aussi d'un terme générique, qui est bien différent suivant l'implémentation ou le modèle. Pour caractériser le réseau, on utilise des paramètres qualitatifs, tels que la topologie, la connectivité, la synchronisation des envois de messages, les communications uni- ou bi-directionelles, etc. Les paramètres quantitatifs précisent le diamètre, la bande passante, la latence, la taille maximale d'un message, etc. Le réseau est modélisé par un graphe, qui est orienté si

(s) Selon les auteurs, et les contextes, un processeur est appelé *entité calculante*, *processus*, *nœud*, *agent*, etc.

les communications sont unidirectionnelles.

L'accès éventuel à une certaine connaissance globale du système distribué est un paramètre important. Il peut s'agir d'une horloge globale, d'une numérotation des entités, d'un état global périodiquement distribué (synchronisation, terminaison, etc.), de l'identifiant[(t)] d'un nœud particulier (élection d'un *leader*), etc. On distingue ainsi les systèmes dont chaque nœud connaît tous les autres, et les systèmes où les nœuds n'ont, au contraire, qu'une connaissance partielle des autres. L'utilisation d'un *routage* permet d'émuler le premier cas par le second. L'identifiant des nœuds est un paramètre central. Lorsqu'il n'y a pas moyen de distinguer deux nœuds quelconques grâce à un identifiant ou à un comportement (exécution) différent, on parle de système *anonyme*. Il a été montré que, dans les systèmes anonymes, certains problèmes sont insolubles, tel que celui de la recherche des composantes bi-connexes d'un réseau non-orienté [Hohberg, 1990]. En effet, il est impossible de casser la symétrie d'un anneau, par exemple, sans utiliser l'identifiant des nœuds, sauf à avoir recours à des solutions non déterministes, qui ne sont pas efficaces [Tel, 1991].

Le mode de communication est également une caractéristique importante d'un système distribué. On distingue les communications par échange de messages des communications par mémoire partagée, ou registres[(u)]. Il faut aussi préciser si les tampons de réception sont de taille finie ou non, et si les messages peuvent être désordonnés ou non (propriété FIFO[78]). En effet, dans un réseau de télécommunication, si les communications ne sont pas effectuées en mode connecté, alors les messages peuvent arriver par des routes différentes, et dans un ordre différent de celui d'émission.

78. *First In First Out.*

Asynchronisme. Il nous paraît important de résumer les différentes formes de synchronisation — et d'absence de synchronisation — considérées dans les systèmes distribués, bien qu'il semble difficile d'obtenir l'unanimité des auteurs sur ces notions[(v)].

Un système est dit *synchrone* si son fonctionnement est organisé en phases globales. Les nœuds sont synchronisés pour que leurs actions aient lieu simultanément durant une même phase globale. Au contraire, un système est *asynchrone* s'il ne présente pas de telles synchronisations. Malgré l'absence de telles phases globales, une certaine forme de synchronisation peut être induite par les communications. Ainsi, dans les systèmes asynchrones, on distingue les envois de messages synchrones des envois asynchrones (on parle de *canal synchrone* ou de *canal asynchrone*) [Tel, 1991]. Une communication est *synchrone* si l'envoi et la réception sont bloquantes : émetteur et récepteur sont bloqués jusqu'à ce qu'ils puissent faire la communication ensemble ; la communication les a donc synchronisés. Dans une *communication asynchrone*, l'envoi n'est pas bloquant. Le temps de délivrance du message est arbitraire, mais fini. La réception est généralement bloquante car l'entité attend l'arrivée d'un message[(w)].

Mais si on admet que tout message envoyé est reçu avant un délai τ_r (*cf.* hypothèse 11 page 37), alors la communication asynchrone induit tout de même une synchronisation. Dans [Tel, 1994], l'auteur parle de *réseau asynchrone à délai borné*[79], mais un système à communications bornées est généralement assimilé à un système synchrone [Delaët, 1995]. En effet, il suffit de commencer une nouvelle phase globale[80] toutes les τ_r unités de temps. Au contraire, dans un système totalement asynchrone, il n'y a pas de concept de temps réel. Des mécanismes, tels que les *synchroniseurs*, permettent d'émuler un réseau synchrone par un réseau asynchrone (sans faute) [Lynch, 1996]. Pour cela, on peut étiqueter les messages avec le numéro de la phase globale à laquelle ils appartiennent, pour ne pas qu'ils soient confondus par les nœuds.

79. En anglais, *Asynchronous Bounded Delay (ABD) network.*

80. En anglais, *round*, ou *pulse*.

(t) L'identifiant est aussi appelé *identité*. Nous avons parlé d'*index* plutôt que d'identité (hypothèse 10 page 36), pour éviter tout risque de confusion avec *identity*, qui signifie élément neutre en anglais.

(u) Le terme *communication par registres* est habituellement réservé aux systèmes à lecture-écriture atomique.

(v) Il s'agit ici de synchronisation des nœuds entre eux ; nous laissons de côté les synchronisations des nœuds avec les liens de communications.

(w) Notons qu'en programmation par passage de messages, il est généralement possible d'effectuer un calcul pendant l'attente. Il faut pour cela que le calcul n'ait pas besoin tout de suite du message attendu. En systèmes distribués, on suppose les communications très longues par rapport aux calculs, et ces derniers sont généralement considérés comme immédiats. Dans ce cas, le nœud calcule instantanément, à l'arrivée d'un message (changement d'état), et attend ensuite le prochain message.

Calculs. Le calcul réalisé par un nœud est toujours séquentiel. Lorsqu'il est déterministe, le comportement de l'entité est entièrement déterminé par son historique, et la suite des messages qu'elle a reçus. Les systèmes ayant des communications synchrones peuvent être décrits par des modèles de programmation utilisant la notion de *rendez-vous* [Lamport et Lynch, 1990], tels que CSP [Hoare, 1978]. Le comportement des système asynchrones est généralement décrit par un ensemble d'états, et de règles de transitions entre ces états.

Lorsque les actions de lecture et celles d'écriture sont les seules qui soient atomiques[81], on parle de *lecture-écriture atomique*[82]. Lorsqu'une lecture suivie d'une écriture (ou l'inverse) constituent une action atomique, on parle d'*atomicité composite*[83]. Par exemple, un système dans lequel les nœuds réalisent, en une seule action atomique, la lecture des données sur leurs voisins, le calcul d'un nouvel état et l'envoi d'un message, ne vérifie pas les spécifications de la lecture-écriture atomique, mais vérifie celle de l'atomicité composite. Les modélisations des systèmes à lecture-écriture atomiques sont plus générales que celles à atomicité composite, car elles conduisent à des exécutions différentes plus nombreuses. En particulier, un résultat montré pour la lecture-écriture atomique sera également vrai sous l'hypothèse de l'atomicité composite, mais l'inverse est faux. Nous donnons maintenant quelques définitions et notations qui nous serviront par la suite.

81. Action atomique : qui ne peut être interrompue par une autre.

82. En anglais, *read-write atomicity.*

83. En anglais, *composite atomicity.*

Une *configuration* d'un système distribué est une instance des états des entités et des liens qui le composent (la configuration note les états de tous les éléments du système). On note $\mathcal{C}$ l'ensemble des configurations du système. Les actions des entités calculantes changent la configuration globale du système. Une *exécution*[84] e est une séquence de configurations $c_1, c_2, \ldots$ telle que pour tout $i = 1, 2, \ldots$, la configuration c_{i+1} est atteinte depuis la configuration c_i par une seule action d'une seule entité[(x)]. c_1 est appelée *configuration initiale* de e. On suppose que les exécutions sont *équitables*[85] (si une action est continuellement exécutable alors elle finira par être réalisée) et *maximales* (l'exécution est infinie ou alors elle est finie et plus aucune action n'est possible). L'ensemble des exécutions commençant par la configuration initiale $c_1 \in \mathcal{C}$ est noté $\mathcal{E}_{c_1}$. Toutes les exécutions e de $\mathcal{E}_{c_1}$ sont de la forme $c_1, c_2, \ldots$. L'ensemble de toutes les exécutions dont les configurations initiales sont des éléments de $\mathcal{C}_1 \subset \mathcal{C}$ est noté $\mathcal{E}_{\mathcal{C}_1}$. L'ensemble de toutes les exécutions possibles du système est noté $\mathcal{E} = \mathcal{E}_{\mathcal{C}}$.

84. En anglais, *computation* ou *execution.*

85. En anglais, *fair.*

Quand les communications du système se font par registres, on décrit un programme, ou *protocole*, par une liste d'*actions gardées*. Une action gardée est donnée par $\langle$*garde*$\rangle \longrightarrow \langle$*instruction*$\rangle$. Une *garde* est une expression booléenne sur les variables accessibles en lecture par le nœud. Une *instruction* ne peut concerner que la modification des variables pour lesquelles le nœud a un accès en écriture. Toute action dont la garde est vraie est dite *exécutable*. Quand aucune règle n'est faisable dans la présente configuration, celle-ci est appelée *terminale*.

4.3.2 Auto-stabilisation*

Nous décrivons maintenant le domaine de l'auto-stabilisation, que nous utilisons au chapitre 6.

Introduction. L'auto-stabilisation est un concept de résistance aux pannes qui a été introduit dans [Dijkstra, 1974]. La propriété d'auto-stabilisation d'un algorithme garantit que celui-ci convergera vers le comportement désiré, et ce, quelle que soit sa configuration de départ. La plupart des programmes auto-stabilisants ne sont pas censés s'arrêter. C'est le cas, par exemple, des protocoles qui assurent le routage des messages dans les réseaux informatiques : leur exécution n'est pas bornée dans le temps. Lorsqu'une panne temporaire survient, modifiant l'état d'un nœud mais pas son comportement (son programme[(a)]), alors le programme retrouve un comportement normal au bout d'un temps fini après la disparition de la panne. Il n'y a donc pas besoin d'une intervention humaine pour ré-initialiser le système. De plus, la tâche, souvent complexe, d'initialisation du système distribué

(x) Nous décrivons ici uniquement le modèle à lecture-écriture atomique que nous utilisons par la suite.

(a) On peut supposer, par exemple, que le programme auto-stabilisant est réalisé par un circuit électronique spécialisé (comme dans la Maille Associative). Les opérations étant câblées, le «code» du programme ne peut subir d'erreur, ce qui n'est pas le cas des registres, qui peuvent contenir des valeurs erronées.

n'est plus nécessaire puisque même s'il est mal initialisé, le programme distribué finira tout de même par présenter le comportement souhaité.

Définitions. Une *spécification* est un prédicat sur les exécutions d'un système distribué. Le système *vérifie la spécification* si toutes les exécutions possibles la vérifient. Lorsque la spécification porte sur une propriété stable globale (*i.e.*, un prédicat sur les configurations du système qui doit toujours être vérifié), elle peut être donnée sous la forme d'un ensemble de configurations, appelées *configurations légitimes*, noté $\mathcal{L}$. Ainsi, chaque exécution qui vérifie la spécification est une séquence constituée de configurations légitimes. Les autres configurations, de $\mathcal{C} \setminus \mathcal{L}$, sont appelées *configurations illégitimes*. Si, pour toute configuration initiale, toute exécution conduit à une configuration dans laquelle toute action exécutée ne change pas la configuration du système, l'algorithme est dit *silencieux*[86] [Dolev et al., 1996]. Si les actions sont gardées, alors l'algorithme silencieux finit sur une configuration terminale. En effet, les expressions booléennes des gardes ne seront plus vérifiées, et plus aucune action ne sera réalisée dans le système. Ce dernier ne changera donc plus de configuration. La dernière configuration est appelée *configuration terminale*.

86. En anglais, *silent*.

Pour définir formellement l'auto-stabilisation (ce qui est nécessaire pour les preuves), nous introduisons la notion d'*attracteur*. Intuitivement, un attracteur est un ensemble de configurations qui «attirent» les autres, pour toutes les exécutions possibles. Un *attracteur fermé* est tel que, si une exécution arrive dans l'attracteur, alors elle n'en sort plus.

Définition 26 *Soit $\mathcal{C}_a$ et $\mathcal{C}_b$ deux sous-ensembles de configurations de $\mathcal{C}$. $\mathcal{C}_a$ est un attracteur fermé pour $\mathcal{C}_b$ si, et seulement si, pour toute configuration initiale c_1 de $\mathcal{C}_b$, pour toute exécution e de $\mathcal{E}_{c_1}$ ($e = c_1, c_2, \dots$), il existe $i \geq 1$ tel que, pour tout $j \geq i$, $c_j \in \mathcal{C}_a$.*

Définition 27 *Un système est auto-stabilisant si et seulement s'il existe un sous-ensemble non vide $\mathcal{L} \subset \mathcal{C}$ de configurations légitimes tel que $\mathcal{L}$ est un attracteur fermé de $\mathcal{C}$.*

Les programmes auto-stabilisants sont comparés selon des critères de *complexité en temps* de stabilisation (temps nécessaire pour atteindre une configuration légitime, une fois que toutes les pannes ont disparu), ou de place mémoire.

4.3.3 Les systèmes distribués et les réseaux associatifs

Nous rapprochons maintenant le modèle des réseaux associatifs et le formalisme des systèmes distribués. En particulier, nous explicitons le schéma de communication, et nous donnons l'algorithme de la `direct-association` sous forme de règles. Nous montrons ensuite que l'exécution de ces règles peut être considérée atomique, sans que les spécifications de la lecture-écriture atomique soient violées. Une partie des résultats présentés ici a été rédigée dans [Ducourthial et Tixeuil, 1998a] [Ducourthial et Tixeuil, 1998b] .

Nœuds et réseau. Le mode de communication dans un réseau associatif, tel que défini au § 3.2 page 35, correspond à la communication par registres. Un nœud peut uniquement connaître les registres de sortie de ses antécédents directs.

Chaque entité v a un accès en lecture aux registres de sortie $b_{\text{out}}[u]$, de ses antécédents directs $u \in \Gamma_G^{-1}(v)$ dans le réseau G. Le nœud v est le seul à pouvoir écrire dans ses registres entrants $b_{\text{in}}[v]$; il y recopie les variables qu'il aura lues dans les registres de sortie de ses antécédents. Il est également le seul à pouvoir lire ses registres entrants, qui peuvent être considérés comme ses variables privées. Le nœud v est aussi le seul à pouvoir écrire dans son registre de sortie $b_{\text{out}}[v]$ un quelconque résultat. Par contre, il n'est pas le seul à pouvoir lire ce registre, puisque tous ses descendants directs y lisent son résultat[(b)]. Ainsi, les seuls registres partagés, et utilisés pour les communications, sont

(b) Dans le modèle que nous avons défini au chapitre 3, le nœud v ne lit son registre de sortie que si sa boucle (v, v) n'est pas masquée. Pour écrire des règles qui ne sont pas continuellement applicables, nous supposons ici que v peut toujours lire son registre de sortie. Étant donné qu'il s'agit de son propre registre, cette hypothèse est très réaliste.

les registres de sortie b_{out}, qui sont accessibles en lecture-écriture par le propriétaire, et en lecture par tous ses descendants.

On définit les fonctions de lecture et d'écriture comme suit. Un appel à la fonction $\texttt{read}(b_{\text{out}}[u])$ par le nœud v, descendant direct de u, consiste à lire la valeur se trouvant dans le registre $b_{\text{out}}[u]$. Un appel à la fonction $\texttt{write}(b_{\text{out}}[v], x)$ par le nœud v consiste à écrire la valeur x dans le registre partagé $b_{\text{out}}[v]$.

L'état d'un nœud est donné par les valeurs de ses registres privés $b_{\text{in}}[v]$, et d'éventuelles autres variables privées internes nécessaires à l'algorithme. L'état d'un lien (u, v) est donné par la valeur du registre partagé $b_{\text{out}}[u]$. Une action atomique d'un nœud consiste en une action interne suivie soit d'un `read`, soit d'un `write`.

Modélisation de la `direct-association` par des règles. Nous nous intéressons ici au cas d'une `direct-association` dans un réseau ne comportant pas de boucles reliant le registre de sortie $b_{\text{out}}[v]$ d'un nœud v à son registre entrant $b^0_{\text{in}}[v]$. En outre, nous supposons qu'il n'y a pas d'arc masqué **(c)**.

Un calcul local d'une entité calculante v participant à une $\diamond$-`direct-association`(net_G, p) consiste à lire, avec la fonction `read`, les valeurs se trouvant dans les registres de sorties de ses antécédents directs $u \in \Gamma_G^{-1}(v)$, à les copier dans les registres entrants privés $b_{\text{in}}[v]$, à effectuer un calcul avec l'opérateur $\diamond$ en utilisant les variables de ses registres $b_{\text{in}}[v]$, et à écrire le résultat dans son registre de sortie partagé $b_{\text{out}}[v]$. Plus formellement, on définit l'algorithme de la `direct-association`, paramétré par l'opérateur $\diamond$, avec deux règles atomiques : *copie* et *calcul*. Pour que ces règles ne soient pas continuellement applicables, on teste si une modification doit être faite avant de l'exécuter (on utilise une garde).

Algorithme 5 $\diamond$-`direct-association`

▶ *Règle de copie sur le nœud v du réseau G :*

$R_1|\diamond$: $\left\{u \in \Gamma_G^{-1}(v) : \quad (\alpha \leftarrow \texttt{read}(b_{\text{out}}[u])) \quad \wedge \quad b_{\text{in}}^{\mathbf{i}(u)}[v] \neq \alpha\right\} \longrightarrow \left\{b_{\text{in}}^{\mathbf{i}(u)}[v] \leftarrow \alpha\right\}$

▶ *Règle de calcul sur le nœud v du réseau G :*

$R_2|\diamond$: $\left\{\texttt{read}(b_{\text{out}}[v]) \neq \diamond\left(b^0_{\text{in}}[v], \dots, b^{\delta^-_v}_{\text{in}}[v]\right)\right\} \longrightarrow \left\{\texttt{write}\left(b_{\text{out}}[v], \diamond\left(b^0_{\text{in}}[v], \dots, b^{\delta^-_v}_{\text{in}}[v]\right)\right)\right\}$

Règles atomiques. Nous avons décrit l'algorithme de la `direct-association` en supposant que les deux règles de copie et de calcul sont atomiques. Cela rend plus simples les preuves du chapitre 6. Mais il ne s'agit pas d'une restriction due à l'algorithme, car celui-ci a un comportement similaire lorsque seules les actions de lecture et d'écriture sont supposées atomiques. En effet, l'algorithme 5 ne remet pas en cause les spécifications de la lecture-écriture atomique telles qu'indiquées dans [Dolev et al., 1993].

Proposition 28 *L'algorithme 5 de la `direct-association` admet la lecture-écriture atomique.*

Preuve. L'hypothèse de la lecture-écriture atomique signifie que seules les actions de lecture (`read`) et d'écriture (`write`) sont atomiques (*cf.* § 4.3.1 page 71), ce qui, *a priori*, remet en cause l'atomicité des règles de copie et de calcul de l'algorithme 5.

Cependant la règle $R_1|\diamond$ contient une seule instruction `read` et des actions internes ; elle fonctionne donc de la même manière en lecture-écriture atomique. La règle $R_2|\diamond$ contient une instruction `read` et une instruction `write` qui utilisent le même registre $b_{\text{out}}[v]$. Pour suivre les spécifications de la lecture-écriture atomique, cette règle devrait être écrite avec deux sous-règles, dont la première ne contiendrait que le `read`, et la seconde que le `write` ; ces deux sous-règles pourraient donc être atomiques. Nous montrons maintenant que dans toute exécution, le remplacement de la règle $R_2|\diamond$ par ces deux sous-règles indépendantes

(c) Cette restriction ne vient pas d'une impossibilité à prendre en compte les boucles et les masques, mais d'un soucis de simplification. La modélisation présentée reste suffisante pour l'utilisation que nous faisons de cet algorithme au chapitre 6.

ne peut être détectée, par aucun nœud du réseau. On note $w_j \in \Gamma_G^{+1}(v)$ les descendants directs de v ; ils ont un accès en lecture sur le registre de sortie $b_{\text{out}}[v]$ de v. Les exécutions suivantes sont indiscernables pour tous les nœuds du réseau (l'indice sur les actions `read` et `write` indique quel est le nœud qui fait l'action) :

$\ldots, \texttt{read}_v(b_{\text{out}}[v]), \texttt{read}_{w_j}(b_{\text{out}}[v]), \texttt{write}_v(b_{\text{out}}[v]), \ldots$

$\ldots, \texttt{read}_{w_j}(b_{\text{out}}[v]), \texttt{read}_v(b_{\text{out}}[v]), \texttt{write}_v(b_{\text{out}}[v]), \ldots$

Cela signifie que le descendant w_j de v peut intercaler une lecture du registre de sortie de v entre le moment où v exécute l'action `read` de sa règle $R_2|\diamond$, et le moment où il exécute l'action `write` de cette même règle, sans que le comportement du système n'en soit perturbé. La règle $R_2|\diamond$ de l'algorithme 5 peut donc être écrite en deux règles atomiques indépendantes, composées chacune d'un seul `read` ou d'un seul `write`. L'algorithme respecte donc les spécifications de la lecture-écriture atomique. $\square$

La définition des réseaux associatifs présentée au chapitre 3 ne précise aucune contrainte quant à la coordination du réseau et des calculs sur les nœuds. C'est cette hypothèse très faible qui permet d'obtenir un algorithme distribué pour la `direct-association` qui respecte la lecture-écriture atomique.

4.4 Conclusion

Bilan. Dans ce chapitre, nous avons étudié plusieurs modélisations de l'objet informatique que constituent les réseaux associatifs.

L'algèbre max-plus permet de modéliser le calcul d'une `direct-association` avec un s-opérateur, ce que ne permet pas l'«algèbre classique» (non idempotente). La spécificité de l'idempotence des s-opérateurs peut donc être prise en compte. Cette modélisation macroscopique peut sembler intéressante. Cependant, elle induit une synchronisation des nœuds de par le calcul matriciel.

Les itérations asynchrones éliminent ces synchronisations. Elles permettent de modéliser un calcul asynchrone dans un réseau. Nous avons montré que le calcul de la matrice $\mathbf{A}^*$, utilisée pour la résolution des équations, pouvait se faire par une telle itération, sans synchronisation. De même, la proposition 25 page 70 permet d'établir la terminaison d'une $\oplus$-`direct-association`, où $\oplus$ est un s-opérateur, à l'aide d'un point fixe matriciel, malgré son asynchronisme. Nous étendons ce résultat aux r-opérateurs au chapitre 5 ; nous indiquons aussi au chapitre 6 comment adapter le raisonnement sur l'auto-stabilisation au point fixe matriciel.

Enfin, les systèmes distribués permettent de rendre compte avec plus de précisions des échanges de données dans le réseau. Nous précisons les liens entre cette modélisation et celle des itérations asynchrones au chapitre 6, où nous envisageons l'étude du contrôle des réseaux associatifs.

Nécessité d'une modélisation microscopique. Le calcul matriciel n'est pas toujours adapté pour modéliser les réseaux associatifs. Il présente trois inconvénients. Tout d'abord, le calcul d'une `direct-association` est défini par le calcul local, c'est-à-dire par l'opérateur utilisé. Si l'ensemble de ces calculs locaux peut être modélisé par un calcul matriciel, la démarche inverse n'est pas évidente. Un certain nombre d'algorithmes sont définis à l'aide de calculs locaux, par exemple en traitement d'images. Définir ces algorithmes par un calcul matriciel est une toute autre démarche.

Ensuite, le calcul matriciel «stocke», pour chaque nœud, autant d'informations qu'il y a de nœuds dans le réseau. En effet chaque nœud se voit attribuer une ligne de la matrice d'incidence du réseau. Par exemple, dans le cas de la recherche des chemins de poids maximum, la $i^{\text{ème}}$ ligne de la matrice $\mathbf{A}^*$ stocke les poids minimaux des chemins partant de tous les sommets du graphe et arrivant en i. Or le calcul effectué par une `direct-association` ne peut construire plus d'une valeur par nœud (elle peut être une liste cependant). La modélisation matricielle ne reflète donc pas exactement le calcul effectué dans un réseau associatif.

Enfin, la vision matricielle du calcul implique de penser l'algorithme de façon synchrone. Même si les itérations asynchrones permettent de mettre en œuvre l'algorithme matriciel de façon asynchrone, il s'agit d'un processus de «traduction» d'un programme conçu de façon synchrone dans un environnement tirant parti de l'asynchronisme. Or nous expliquons au chapitre 7 que la `direct-association` permet d'utiliser l'asynchronisme dès la conception de l'algorithme. Des avantages en termes d'expressivité et de performance peuvent donc être attendus.

Nous étudions donc les aspects calculatoires du réseau sous l'angle microscopique au chapitre suivant.

Chapitre 5

r-opérateurs

La définition de la `direct-association` a déplacé le problème de l'implémentation qui se pose pour l'`association` vers un problème de terminaison. En effet, la `direct-association` n'est définie que si elle se termine.

Dans le chapitre 4, nous avons envisagé une modélisation macroscopique des aspects calculatoires d'un réseau associatif. Dans ce chapitre, nous étudions le problème au niveau du nœud et de l'opérateur utilisé pour son calcul local.

Nous résumons tout d'abord au § 5.1 le problème à résoudre, les résultats déjà obtenus dans les chapitres précédents, les hypothèses que nous nous fixons et la méthode que nous utilisons. Puis nous développons au § 5.2 de nouveaux opérateurs : les r-opérateurs binaires. Ils présentent d'intéressantes propriétés algébriques et algorithmiques. En particulier, ils généralisent les s-opérateurs tout en assurant encore la terminaison des `direct-associations`.

Au § 5.3, nous étendons ces résultats aux opérateurs n-aires, avant de les généraliser au § 5.4. Nous concluons ensuite sur l'intérêt de ces opérateurs, et sur une comparaison avec la modélisation matricielle du chapitre précédent.

5.1 Problème et méthode

5.1.1 Position du problème

Étude microscopique des propriétés calculatoires. L'`association` (*cf.* § 1.2.1 page 12) est une primitive de calcul de haut niveau, toujours définie, mais parfois difficile à mettre en œuvre, comme expliqué au § 3.3 page 40. La `direct-association` (*cf.* § 3.2 page 35) est une primitive de calcul de bas niveau ; elle est facilement mise en œuvre. Mais elle ne définit un opérateur global de $\mathbb{S}^{|V|}$ dans $\mathbb{S}^{|V|}$ que si l'opérateur $\diamond$, défini sur $\mathbb{S}$ et utilisé par les $|V|$ nœuds du réseau, conduit à la terminaison de la $\diamond$-`direct-association`.

La redéfinition du modèle a donc introduit une nouvelle problématique, qui se traduit par la recherche d'opérateurs assurant la terminaison — et donc la définition — du calcul global effectué par la `direct-association`, et par la détection de la terminaison de ce calcul (que nous traitons au chapitre 6).

Nous avons abordé le problème de la terminaison du calcul sous l'angle matriciel au chapitre précédent. Mais nous avons aussi mentionné les limites de cette modélisation. Dans ce chapitre, nous prenons le parti de la vision microscopique, qui permet la conception asynchrone d'algorithmes (*cf.* chapitre 7 page 123). Nous recherchons des opérateurs $\diamond$ qui assurent la terminaison de la $\diamond$-`direct-association`, et dont les applications algorithmiques sont plus riches que celles des s-opérateurs.

Définition du problème. Le problème consiste à trouver des opérateurs assurant la **terminaison** de la `direct-association` sur un résultat **déterministe** et **reconnaissable**.

Nous avons déjà montré que certaines topologies de réseau permettaient la **terminaison** des `direct-associations` (proposition 1 page 30 notamment). Nous recherchons dans ce chapitre des propriétés qui assurent la terminaison sur toute sorte de topologie, ce qui permettra de les utiliser sans avoir une connaissance préalable du réseau. Néanmoins les conditions auxquelles nous aboutirons seront parfois trop fortes si certaines informations sur la topologie sous-jacente, et sur le placement des opérateurs et des données sur cette topologie, sont connues (*cf.* § 5.4.3 page 100).

Résultat **déterministe** signifie que les mêmes valeurs seront obtenues sur les mêmes nœuds si on recommence le même calcul sur les mêmes données. L'utilisation d'opérateurs déterministes, c'est-à-dire assurant un calcul local déterministe, n'est pas suffisante pour assurer le déterminisme du calcul global. En effet, l'asynchronisme du réseau pourrait conduire à des expressions différentes d'une exécution à une autre, selon l'ordre d'arrivée aléatoire des données sur un nœud. La définition 12 page 40 définit plus formellement la terminaison déterministe.

Résultat **reconnaissable** signifie «utilisable en programmation». Il s'agit d'obtenir un résultat du calcul global sous une forme réduite, à laquelle on peut donner une signification. Il existe des opérateurs qui, bien que conduisant à la terminaison du calcul, ne donnent pas une formule exploitable, permettant de reconnaître le résultat. Nous rencontrerons ce problème avec les r-opérateurs généralisés.

5.1.2 Résultats déjà obtenus

s-opérateurs. Les s-opérateurs sont des opérateurs associatifs, commutatifs, idempotents et admettant un élément neutre sur leur ensemble de définition (*cf.* définition 13 page 40). Ces opérateurs assurent la terminaison déterministe des `direct-associations` sur tout réseau (*cf.* théorème 15 page 41). Ils donnent une seule valeur par composante fortement connexe du réseau. La proposition 25 page 70 donne une preuve matricielle, grâce aux itérations asynchrones.

Condition nécessaire. La proposition 3 page 31 indique que la k-idempotence faible est une condition nécessaire à la terminaison d'une `direct-association`, définie comme l'exécution concurrente de calculs locaux. Notons cependant qu'au § 3.1.2 page 30, nous émettons des réserves quant à l'intérêt de la propriété d'idempotence seule. En effet, d'autres propriétés sont nécessaires pour pouvoir supprimer les doubles dans les expressions à l'aide de l'idempotence. Pour un s-opérateur, c'est l'associativité et la commutativité qui permettent d'exploiter l'idempotence. Les opérateurs que nous développons doivent avoir la propriété d'idempotence, mais aussi les propriétés qui permettent d'isoler, au sein même de l'expression, les doublons afin de les supprimer par idempotence.

5.1.3 Méthode employée

L'étude des propriétés des opérateurs nécessaires à la convergence du réseau associatif aurait pu être menée sous l'angle de la sémantique du réseau, considéré comme un réseau flot de données [Brookes, 1993]. Cette approche, proposée par Kahn, nécessite l'introduction d'un formalisme adéquat [Kahn et Plotkin, 1993]. Nous avons préféré utiliser une méthode plus simple, basée sur la réduction d'expressions correspondant aux calculs effectués par les nœuds. Si cette approche est plus limitée que l'étude sémantique, elle n'en reste pas moins suffisante pour les besoins de notre étude, à savoir la définition d'opérateurs (r-opérateurs) utilisables avec la `direct-association`.

Expressions. Pour étudier la terminaison d'une $\diamond$-`direct-association`, nous considérons les résultats sous leur forme littérale, non simplifiée. Ainsi, un nœud fabrique une expression — et non une valeur — avec les expressions qu'il reçoit. Le résultat de la $\diamond$-`direct-association` est donc un ensemble de $|V|$ expressions, de tailles non bornées, composées de l'opérateur $\diamond$, de parenthèses transcrivant les calculs intermédiaires, et des variables correspondant aux composantes de la pvar b^0_{in} initiale (*i.e.*, $b^0_{\text{in}}[u]$, $b^0_{\text{in}}[v]$, $b^0_{\text{in}}[w]$, *etc.*). Chaque expression contient un nombre borné de variables différentes, car il ne peut y en avoir plus que le nombre de nœuds dans le réseau. Par contre, ces variables peuvent apparaître un nombre quelconque (non borné) de fois dans l'expression, par exemple à cause d'un circuit dans le réseau.

Cette technique permet de considérer le «résultat» d'un calcul global même lorsque le réseau ne se stabilise pas. Au lieu de manipuler des valeurs sans cesse changeantes, on manipule des expressions. Celles-ci peuvent, certes, être de plus en plus longues, mais l'usage de «jockers» (tel que $\cdots$) permet cependant de les écrire et de les manipuler. Par la suite, «expression de taille finie» signifie «qui peut être écrite sans jocker» comme, par exemple, l'expression $x \diamond y$.

Exemple. Considérons le réseau à deux nœuds de la figure 60. Le nœud u fabrique une seule expression au cours de la $\diamond$-`direct-association`: x. On aura donc $(b_{\text{out}}[u])^{t_\infty} = x$, où l'exposant t_∞ indique la date de lecture de l'expression, et t_∞ signifie «toute date suffisamment grande» (*cf.* définition 12 page 40). Tant que le nœud v n'a reçu aucune valeur de u, il construit l'expression $y \diamond e_\diamond \diamond \cdots \diamond e_\diamond$ car son tampon entrant $b^1_{\text{in}}[v]$ contient l'élément neutre à droite $e_\diamond$ par défaut. Puis, lorsqu'il reçoit l'expression de $b_{\text{out}}[u]$, il intègre x à son calcul et obtient l'expression $y \diamond e_\diamond \diamond \cdots \diamond e_\diamond \diamond x$. On a donc $(b_{\text{out}}[v])^{t_\infty} = y \diamond e_\diamond \diamond \cdots \diamond e_\diamond \diamond x \diamond \cdots \diamond x$.

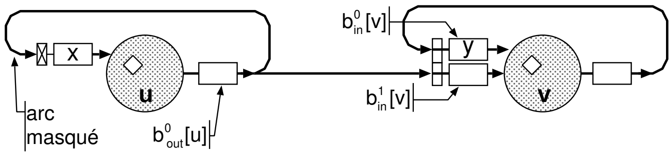


FIG. 60 – *Exemple d'expressions construites avec le réseau :* $(b_{out}[u])^{t_\infty} = x$ *et* $(b_{out}[v])^{t_\infty} = y \diamond e_\diamond \diamond \cdots \diamond e_\diamond \diamond x \diamond \cdots \diamond x$.

Réduction des expressions. Dans l'exemple précédent, le résultat de la $\diamond$-`direct-association` existe si et seulement si certaines propriétés de l'opérateur $\diamond$ permettent de simplifier — ou *réduire* — l'expression $(b_{\text{out}}[v])^{t_\infty} = y \diamond e_\diamond \diamond \cdots \diamond e_\diamond \diamond x \diamond \cdots \diamond x$. Par exemple, les propriétés d'associativité à gauche et d'élément neutre à droite $e_\diamond$ permettent de pratiquer une première réduction et d'obtenir l'expression $(b_{\text{out}}[v])^{t_\infty} = y \diamond x \diamond \cdots \diamond x$. Mais ce n'est pas suffisant pour affirmer que, quelles que soient les valeurs de x et de y utilisées, l'expression sera égale à une expression de taille finie (*i.e.*, sans le jocker $\cdots$). D'autres propriétés de l'opérateur sont nécessaires. Nous définissons la propriété de réduction comme suit.

Définition 29 *Un opérateur $\diamond$ vérifie la propriété de réduction sur $\mathbb{S}$ si toute expression de taille quelconque, composée de parenthèses*[(a)]*, d'opérateurs $\diamond$, et d'un nombre borné d'éléments de $\mathbb{S}$ (éventuellement répétés un nombre quelconque de fois), peut se réduire en une expression de taille finie (sans jocker).*

Clairement, l'expression sur les entiers $1 + \cdots + 1$ ne peut se réduire, tandis que l'expression $0 + \cdots + 0$ peut se réduire en l'expression 0. L'opérateur d'addition $+$ vérifie donc la propriété de réduction sur le sous-ensemble $\{0\}$ de $\mathbb{N}$, mais pas sur le sous-ensemble $\{1\}$. La proposition suivante est une autre formulation, utilisant les réductions d'expressions, de la nécessité de la k-idempotence faible (*cf.* proposition 3 page 31).

Proposition 30 *Si l'opérateur $\diamond$ défini sur $\mathbb{S}$ n'est pas k-idempotent faible sur $\mathbb{S}$ (cf. définition 2 page 31), alors il ne vérifie pas la propriété de réduction sur $\mathbb{S}$.*

Preuve. Il suffit de considérer une valeur $x \in \mathbb{S}$ pour laquelle il n'existe aucun $k \in \mathbb{N}$ tel que $\underbrace{x \diamond \cdots \diamond x}_{k+1} = \underbrace{x \diamond \cdots \diamond x}_{k}$. Dans ce cas, l'expression composée d'un nombre quelconque de x $(x \diamond \cdots \diamond x)$ et de l'opérateur $\diamond$ ne peut être réduite. □

Proposition 31 *Un* s*-opérateur $\oplus$, défini sur $\mathbb{S}$, vérifie la propriété de réduction sur $\mathbb{S}$. De plus, toute expression de taille quelconque X, mais ayant un nombre borné de variables différentes $x_1, \ldots, x_n$, se réduit en l'expression $x_1 \oplus \cdots \oplus x_n$.*

(a) On suppose que toutes les expressions considérées sont bien parenthésées.

Preuve. Soit X une expression de taille quelconque (non bornée), composée d'un nombre quelconque de variables, de s-opérateurs $\oplus$ et de parenthèses, de telle sorte que le nombre n de variables différentes $x_1, \dots, x_n$ soit fini. Le s-opérateur $\oplus$ admet les propriétés d'associativité, de commutativité et d'idempotence. En utilisant l'associativité de $\oplus$, on peut supprimer toutes les parenthèses de X. Puis, en utilisant la commutativité, on peut rassembler les variables x_i identiques. Ensuite, en utilisant de nouveau l'associativité, on peut isoler les variables identiques par des parenthèses : $X = \cdots \oplus (x_i \oplus \cdots \oplus x_i) \oplus \cdots$. Enfin, en utilisant l'idempotence, on peut supprimer les doubles et obtenir l'expression $x_1 \oplus \cdots \oplus x_n$. □

Cette proposition permet de montrer la terminaison d'une $\oplus$-`direct-association` utilisant un s-opérateur. En effet, le fait que toute expression puisse être réduite signifie que sur tout réseau, les expressions des nœuds pourront se ramener à une expression de taille finie puis à une valeur. L'associativité et la commutativité assurent le déterminisme (*cf.* § 3.1 page 29), tandis que le théorème 14 page 40 permet de le reconnaître[(b)].

Plus généralement, pour montrer qu'une $\diamond$-`direct-association` se termine, il est suffisant de montrer qu'il existe une date à partir de laquelle, pour chaque nœud v du réseau, toutes les expressions successivement produites par v se réduisent en une expression fixée X_v de taille finie. Si cette expression est obtenue pour chaque exécution de la $\diamond$-`direct-association` sur le même réseau avec les mêmes données, alors le résultat est déterministe, et il reste à vérifier que l'expression est reconnaissable.

5.1.4 Hypothèses sur les opérateurs

On considère l'ensemble $\mathbb{S}$ des valeurs qu'il est possible de voir apparaître dans le réseau associatif au cours d'une `direct-association`. Dans la suite de ce chapitre, le symbole $\oplus$ désigne un s-opérateur, tandis que $\triangleleft$ désigne un r-opérateur. Le symbole $\diamond$ quant à lui désigne un opérateur quelconque. Par la suite, nous faisons références aux hypothèses suivantes ; les hypothèses 32 et 34 supposées vérifiées par tous les opérateurs traités.

Hypothèse 32 (Loi de composition interne) *L'opérateur binaire $\diamond$ vérifie l'hypothèse de la loi de composition interne sur $\mathbb{S}$ si :*

$$\forall x, y \in \mathbb{S}, \quad \exists! z \in \mathbb{S} \textit{ tel que } x \diamond y = z$$

Hypothèse 33 (Associativité à gauche) *L'opérateur binaire $\diamond$ vérifie l'hypothèse de l'associativité à gauche sur $\mathbb{S}$ si :*

$$\forall x, y, z \in \mathbb{S}, \quad (x \diamond y) \diamond z = x \diamond y \diamond z$$

Hypothèse 34 (Élément neutre à droite) *L'opérateur binaire $\diamond$ vérifie l'hypothèse de l'élément neutre à droite sur $\mathbb{S}$ si :*

$$\exists e_\diamond \in \mathbb{S}, \textit{ tel que } \forall x \in \mathbb{S}, \, x \diamond e_\diamond = x$$

Hypothèse 35 (Régularité faible à gauche) *L'opérateur binaire $\diamond$ vérifie la propriété de régularité faible à gauche sur $\mathbb{S}$ si :*

$$\forall y, z \in \mathbb{S}, (\forall x \in \mathbb{S}, \quad x \diamond y = x \diamond z) \quad \Leftrightarrow \quad (y = z)$$

(b) Le théorème 15 page 41 donne une autre preuve de la terminaison déterministe d'une $\oplus$-`direct-association` sur tout réseau initialisé avec toute pvar, lorsque l'opérateur $\oplus$ est un s-opérateur. La proposition 25 page 70 utilise une méthode matricielle pour ce problème.

Les hypothèses 32 page précédente et 33 page ci-contre sont classiques. La première indique que l'opérateur $\diamond$ est déterministe, c'est-à-dire qu'il n'associe qu'un seul élément à toute paire d'opérande. Par exemple, l'opérateur sur les entiers qui, à (x, y) associe x si la date est paire, et y si la date est impaire, ou qui tire au sort pour savoir s'il retourne x ou y, *etc.* ne définit pas une loi de composition interne sur $\mathbb{N}$; nous dirons qu'il n'est pas déterministe. La seconde est intéressante pour simplifier l'écriture des expressions. L'hypothèse 34 page précédente est nécessaire pour autoriser le masquage des tampons entrants (*cf.* § 3.2.2 page 37). Les s-opérateurs vérifient ces trois hypothèses sur leur ensemble de définition.

L'hypothèse 35 page précédente est, à notre connaissance, originale[(c)]. Il s'agit d'une caractéristique de structure intermédiaire entre la propriété de structure libre — qui permet de simplifier les expressions (les éléments sont tous réguliers : $x \diamond y = x \diamond z \Rightarrow y = z$) — et l'absence de possibilité de simplification. En effet, pour la régularité faible, avant de simplifier l'expression, il faut s'assurer que x peut être simplifié pour *toutes les valeurs* de $\mathbb{S}$. Lorsque l'hypothèse 34 page précédente est admise, cette propriété assure l'unicité de l'élément neutre à droite. En effet, supposons que $x = x \diamond e_\diamond = x \diamond e'_\diamond$ pour tout $x \in \mathbb{S}$. Alors, en utilisant la régularité faible à gauche, on a $e_\diamond = e'_\diamond$.

C'est une hypothèse raisonnable : l'expression $x \diamond y = x \diamond z$ est simplifiée si aucun $x \in \mathbb{S}$ ne la met en défaut. Tous les opérateurs ayant un élément neutre à gauche vérifient cette hypothèse. De même, les opérateurs commutatifs, qui vérifient l'hypothèse 34 page ci-contre, la vérifient (leur élément neutre à droite l'est aussi à gauche). En particulier, les s-opérateurs admettent la régularité faible à gauche. Ce dernier point a son importance. Par exemple, l'opérateur min sur $\mathbb{N}$ ne donne pas une structure libre parce que tous les éléments ne sont pas réguliers. En effet, pour des entiers x , y et z, $\mathsf{min}(x, y) = \mathsf{min}(x, z)$ ne permet pas de conclure sur le fait que $y = z$. Cela est vrai uniquement si x est plus grand que y (ou que z). Par contre, tous les éléments vérifient l'hypothèse de régularité faible à gauche : il suffit de prendre $x = y$ pour obtenir $y \leq z$, puis $x = z$ pour obtenir $y \geq z$. Ainsi, certaines simplifications pourront être opérées sur les expressions, malgré l'absence de régularité du s-opérateur min sur $\mathbb{N}$.

5.2 Les r-opérateurs binaires, une algèbre déformée

Nous avons montré au § 5.1.3 que les s-opérateurs possèdent la propriété de réduction des expressions. En fait, une telle propriété peut être obtenue avec des hypothèses plus faibles que celles qui définissent les s-opérateurs. C'est ce que nous montrons dans cette section. Nous donnons tout d'abord de nouvelles définitions, et développons l'idée de r-algèbre. Puis nous étudions ses propriétés avant de montrer les applications d'une telle construction algébrique pour la `direct-association`. L'intérêt essentiel de cette algèbre déformée réside dans ses applications : réduction des expressions par les r-opérateurs et terminaison déterministe sur un résultat exploitable des `direct-associations`, propriétés autostabilisantes et algorithmiques. Les résultats suivants ont été rédigés dans [Ducourthial, 1997] [Ducourthial, 1998c].

5.2.1 Définitions

Nouvelles propriétés. Nous donnons cinq nouvelles propriétés sur les opérateurs binaires. Les trois premières — r-associativité, r-commutativité et r-idempotence — généralisent les propriétés classiques d'associativité, de commutativité et d'idempotence, en utilisant une application que nous

(c) La propriété de régularité faible à gauche que nous définissons ici ne doit pas être confondue avec la propriété de fidélité (en anglais, *faithful*), qui est définie dans les moduloïdes par le fait que l'application qui associe au vecteur $\mathbf{x}$ le vecteur $\alpha \cdot \mathbf{x}$ est une application injective (avec α un scalaire du corps, et $\cdot$ l'opération externe du moduloïde). Cette confusion nous a conduits à appeler «fidélité» l'hypothèse 35 page ci-contre dans [Ducourthial, 1998c]. Ailleurs, nous l'avons appelée simplification à gauche. Le terme de régularité faible à gauche nous semble finalement plus approprié, car il indique qu'il s'agit d'une version faible de la régularité.

appelons r par commodité (les propriétés gardent le même nom lorsque l'application ne s'appelle pas r). Les deux dernières généralisent les propriétés de commutativité et d'idempotence en utilisant l'ordre des termes dans les expressions.

Définition 36 *L'opérateur binaire $\diamond$ est* r*-associatif sur son ensemble de définition $\mathbb{S}$ s'il existe une application* $\mathsf{r} : \mathbb{S} \to \mathbb{S}$ *telle que pour tout x, y et z dans $\mathbb{S}$, on a :*

$$\begin{aligned}(x \diamond y) \diamond \mathsf{r}(z) &= x \diamond y \diamond \mathsf{r}(z) \qquad \textit{(cf. hypothèse 33)}\\ &= x \diamond (y \diamond z)\end{aligned}$$

Par exemple, l'opérateur $\diamond$ défini sur $\mathbb{N}$ par $x \diamond y \equiv x + 2y$ est r-associatif, avec l'application $\mathsf{r} : \mathbb{N} \to \mathbb{N}$, qui à x associe $\mathsf{r}(x) = 2x$. En effet, $(x \diamond y) \diamond \mathsf{r}(z) = x + 2y + 2\mathsf{r}(z) = x + 2y + 4z$, et $x \diamond (y \diamond z) = x + 2(y + 2z)$. Un opérateur associatif est r-associatif, avec l'application identité $\mathsf{Id} : x \mapsto x$.

Définition 37 *L'opérateur $\diamond$ est* r*-commutatif sur son ensemble de définition $\mathbb{S}$ s'il existe une application* $\mathsf{r} : \mathbb{S} \to \mathbb{S}$ *telle que, pour tout x et y dans $\mathbb{S}$, on a :*

$$\mathsf{r}(x) \diamond y = \mathsf{r}(y) \diamond x$$

Par exemple, la soustraction sur $\mathbb{Z}$ est r-commutative, avec l'application $\mathsf{r} : \mathbb{Z} \to \mathbb{Z}$, qui à x associe $-x$. En effet, $\mathsf{r}(x) \diamond y = -x - y = -y - x = \mathsf{r}(y) \diamond x$. Un opérateur commutatif est r-commutatif avec l'application identité Id. Le lemme suivant est immédiat ; il nous servira pour montrer des résultats ultérieurs.

Lemme 38 *Si $\diamond$ est un opérateur* r*-commutatif sur $\mathbb{S}$ avec l'application* $\mathsf{r} : \mathbb{S} \to \mathbb{S}$ *bijective, alors pour tout x et y de $\mathbb{S}$, les égalités suivantes sont équivalentes :*

$$\begin{aligned}\mathsf{r}(x) \diamond y &= \mathsf{r}(y) \diamond x\\ x \diamond y &= \mathsf{r}(y) \diamond \mathsf{r}^{-1}(x)\\ x \diamond \mathsf{r}^{-1}(y) &= y \diamond \mathsf{r}^{-1}(x)\end{aligned}$$

Définition 39 *L'opérateur binaire $\diamond$ est* r*-idempotent sur son ensemble de définition $\mathbb{S}$ s'il existe une application* $\mathsf{r} : \mathbb{S} \to \mathbb{S}$ *telle que, pour tout x de $\mathbb{S}$, on a :*

$$\mathsf{r}(x) \diamond x = \mathsf{r}(x)$$

Par exemple, l'opérateur max est r-idempotent sur $\mathbb{N}$ avec l'application $\mathsf{r} : \mathbb{N} \to \mathbb{N}$ définie par $\mathsf{r}(x) = x + 1$. En effet, $\mathsf{max}(\mathsf{r}(x), x) = \mathsf{max}(x + 1, x) = x + 1 = \mathsf{r}(x)$. Tout opérateur idempotent est r-idempotent avec l'application identité Id. Comme pour la r-commutativité, on donne le lemme suivant qui simplifiera certaines preuves à venir.

Lemme 40 *Si l'application* $\mathsf{r} : \mathbb{S} \to \mathbb{S}$ *est bijective, alors l'opérateur $\diamond$ est* r*-idempotent si et seulement si, pour tout x de $\mathbb{S}$, on a : $x \diamond \mathsf{r}^{-1}(x) = x$.*

Définition 41 *L'opérateur $\diamond$ défini sur $\mathbb{S}$ est k-commutatif s'il est «commutatif à partir du rang k» dans les expressions : pour tout x, y et z de $\mathbb{S}$, on a*

$$x_1 \diamond \ldots \diamond x_{k-1} \diamond x_k \diamond x_{k+1} = x_1 \diamond \ldots \diamond x_{k-1} \diamond x_{k+1} \diamond x_k$$

Par exemple, l'opérateur défini sur $\mathbb{N}$ par $x \diamond y \equiv x + 2y$ et que nous avons évoqué pour illustrer la r-associativité (définition 36) est 2-commutatif. En effet, $x \diamond y \diamond z = x + 2y \diamond z = x + 2y + 2z = x + 2z + 2y = x \diamond z \diamond y$. Par contre, il n'est pas commutatif (1-commutatif) car $x \diamond y = x + 2y \neq y + 2x$ en général. Un opérateur commutatif et associatif est 2-commutatif car $x \diamond y \diamond z = x \diamond (y \diamond z) = x \diamond (z \diamond y) = x \diamond z \diamond y$.

Définition 42 *L'opérateur $\diamond$ défini sur $\mathbb{S}$ est k-idempotent s'il est «idempotent à partir du rang k» dans les expressions : pour tout x de $\mathbb{S}$, on a*

$$\underbrace{x \diamond \cdots \diamond x}_{k+1} = \underbrace{x \diamond \cdots \diamond x}_{k}$$

Par rapport à la k-idempotence faible (*cf.* définition 2 page 31), la constante[(d)] k de la k-idempotence (forte) est ici la même, pour tous les $x \in \mathbb{S}$. L'idempotence habituelle correspond à la 1-idempotence (forte).

(d) Voir la note **(c)** page 31 au sujet des notations préfixées par une lettre, tel qu'ici k.

Relations. Nous donnons maintenant trois propositions qui précisent certaines relations entre les définitions que nous venons de présenter. La première indique que la 2-commutativité est équivalente à la k-commutativité pour tout $k \geq 2$. Ce résultat n'est pas vrai pour la 1-commutativité, qui est la commutativité habituelle. Afin de simplifier l'exposé, les deux dernières propositions sont énoncées avec une application r bijective[87]. Cependant elles resteraient vraies si r n'était que surjective[88]. En effet, il est suffisant de construire une réciproque en choisissant un élément quelconque dans chaque ensemble d'antécédents $\{x \in \mathbb{S}, \mathsf{r}(x) = y\}$ pour chaque image $y \in \mathbb{S}$.

87. Bijection : $\mathsf{f} : \mathbb{S}_1 \to \mathbb{S}_2$ est une bijection si $\forall y \in \mathbb{S}_2$, $\exists! x \in \mathbb{S}_1$ tel que $\mathsf{f}(x) = y$.

88. Surjection : $\mathsf{f} : \mathbb{S}_1 \to \mathbb{S}_2$ est une surjection si $\forall y \in \mathbb{S}_2$, $\exists x \in \mathbb{S}_1$ tel que $\mathsf{f}(x) = y$.

Proposition 43 *Soit $\diamond$ un opérateur binaire défini sur $\mathbb{S}$ et admettant un élément neutre à droite (hypothèses 34). Alors, s'il est 2-commutatif, il est k-commutatif pour tout $k \geq 2$, et réciproquement.*

Preuve. Supposons que l'opérateur $\diamond$ soit 2-commutatif et considérons l'expression $x_1 \diamond \cdots \diamond x_n$. En utilisant l'hypothèse d'associativité à gauche, on a, pour tout $k \geq 2$, $x_1 \diamond \cdots \diamond x_n = (x_1 \diamond \cdots \diamond x_{k-1}) \diamond x_k \diamond x_{k+1} \diamond \cdots \diamond x_n$. En appliquant la 2-commutativité à cette dernière expression, on obtient $x_1 \diamond \cdots \diamond x_{k-1} \diamond x_k \diamond x_{k+1} \diamond \cdots \diamond x_n = (x_1 \diamond \cdots \diamond x_{k-1}) \diamond x_{k+1} \diamond x_k \diamond \cdots \diamond x_n$. $\diamond$ est donc k-commutatif.

Si maintenant l'opérateur $\diamond$ est k-commutatif, alors il vérifie $x_1 \diamond \ldots \diamond x_{k-1} \diamond x_k \diamond x_{k+1} = x_1 \diamond \ldots \diamond x_{k-1} \diamond x_{k+1} \diamond x_k$ pour tout $x_1, \ldots, x_{k+1} \in \mathbb{S}$. En utilisant l'hypothèse de l'élément neutre à droite, on peut choisir $x_2 = \cdots = x_{k-1} = e_\diamond$, ce qui donne, en réduisant les expressions par la gauche (hypothèse de l'associativité à gauche), $x_1 \diamond x_k \diamond x_{k+1} = x_1 \diamond x_{k+1} \diamond x_k$. L'opérateur $\diamond$ est donc 2-commutatif. $\square$

Proposition 44 *Soit $\diamond$ un opérateur binaire défini sur $\mathbb{S}$ et vérifiant les hypothèses 34 et 35 page 82 et admettant la propriété de r-associativité avec r une bijection de $\mathbb{S}$ dans lui-même. Alors, s'il est 2-commutatif, il est r-commutatif, et réciproquement.*

Preuve. Supposons que l'opérateur $\diamond$ soit r-commutatif sur $\mathbb{S}$. Alors, pour tout $x, y, z \in \mathbb{S}$, puisque $\diamond$ est r-associatif, on a $x \diamond y \diamond z = x \diamond (y \diamond \mathsf{r}^{-1}(z)) = x \diamond (z \diamond \mathsf{r}^{-1}(y))$ (lemme 38 page précédente). Donc $x \diamond y \diamond z = x \diamond z \diamond y$ et $\diamond$ est 2-commutatif.

Maintenant supposons que $\diamond$ soit 2-commutatif. Alors, en utilisant la r-associativité, nous avons, pour tout $x, y, z \in \mathbb{S}$, $x \diamond y \diamond z = x \diamond z \diamond y$, ou encore $x \diamond (y \diamond \mathsf{r}^{-1}(z)) = x \diamond (z \diamond \mathsf{r}^{-1}(y))$. Donc, en utilisant la régularité faible à gauche (hypothèse 35 page 82), nous obtenons $y \diamond \mathsf{r}^{-1}(z) = z \diamond \mathsf{r}^{-1}(y)$ et $\diamond$ est r-commutatif (lemme 38). $\square$

Proposition 45 *Si l'opérateur binaire $\diamond$ défini sur $\mathbb{S}$ admet les propriétés de r-associativité et de r-idempotence, avec r une bijection de $\mathbb{S}$ dans lui-même, alors il vérifie, pour tout $x, y \in \mathbb{S}$, $x \diamond y \diamond y = x \diamond y$, et il est k-idempotent pour tout $k \geq 2$.*

Preuve. Pour tout $x, y \in \mathbb{S}$, on a $x \diamond y \diamond y = x \diamond (y \diamond \mathsf{r}^{-1}(y)) = x \diamond y$, en utilisant la r-associativité, puis la r-idempotence et le lemme 40 page précédente. Pour tout $k \geq 1$, en posant $x = \underbrace{y \diamond \cdots \diamond y}_{k-1}$, on a $x \diamond y \diamond y = \underbrace{y \diamond \cdots \diamond y}_{k+1} = x \diamond y = \underbrace{y \diamond \cdots \diamond y}_{k}$, et $\diamond$ est donc k-idempotent pour tout $k \geq 2$.

$\square$

r-opérateurs. Il est important de noter que les propriétés «classiques» d'associativité, de commutativité et d'idempotence s'obtiennent à partir des propriétés de r-associativité, de r-commutativité et de r-idempotence avec l'application $\mathsf{r} \equiv \mathsf{Id}$. Ainsi les propriétés que nous avons définies généralisent leurs homologues classiques. En faisant varier l'application r, on obtient toutes sortes d'algèbres déformées, que nous appelons r-*algèbres* ; l'«algèbre classique» apparaissant comme une r-algèbre particulière.

Un s-opérateur est défini comme étant un opérateur associatif, commutatif, idempotent et ayant un élément neutre. Il s'agit de la loi d'un demi-groupe abélien idempotent dans l'algèbre classique. Nous définissons maintenant un opérateur similaire dans une r-algèbre.

Définition 46 *L'opérateur $\triangleleft$ est un* r*-opérateur sur $\mathbb{S}$ s'il admet les hypothèses d'élément neutre à droite et de régularité faible à gauche (hypothèses 34 et 35), et s'il existe une bijection* $\mathsf{r} : \mathbb{S} \to \mathbb{S}$ *telle que $\triangleleft$ est* r*-associatif,* r*-commutatif et* r*-idempotent sur $\mathbb{S}$:*

$$\begin{aligned} (x \triangleleft y) \triangleleft \mathsf{r}(z) &= x \triangleleft (y \triangleleft z) \\ \mathsf{r}(x) \triangleleft y &= \mathsf{r}(y) \triangleleft x \\ \mathsf{r}(x) \triangleleft x &= \mathsf{r}(x) \end{aligned}$$

Nous appellerons toujours un tel opérateur «r-opérateur», même si la bijection servant à le définir ne s'appelle plus r. En effet, nous montrons au § 5.2.3 qu'un r-opérateur admet la propriété de réduction des expressions (définition 29 page 81) et le préfixe r signifie «réduction». De la même manière, nous appellerons r*-fonction* la bijection utilisée pour définir le r-opérateur (elle est unique, *cf.* proposition 48).

On peut montrer, en utilisant la r-associativité et la régularité faible à gauche, que si un r-opérateur est défini en utilisant seulement une surjection, alors celle-ci est nécessairement une bijection (théorème 56 page 88). Cependant, pour simplifier notre présentation, nous admettons que l'application r est toujours une bijection. De même, un certain nombre de propriétés des r-opérateurs peuvent être obtenues sans les hypothèses 34 et 35. Cependant, les résultats principaux feront usage de ces hypothèses, et nous les supposerons toujours vraies, pour alléger la présentation[(e)].

Exemple. L'opérateur minc[(f)] que nous définissons maintenant aura d'intéressantes applications en algorithmique (calcul de distance, régions de Voronoï, *etc. cf.* chapitre 7).

Proposition 47 *L'opérateur* minc *défini par* $\mathsf{minc}(x, y) = \mathsf{min}(x, y + 1)$ *est un* r*-opérateur sur* $\overline{\mathbb{N}} = \mathbb{N} \cup \{+\infty\}$.

Preuve. On vérifie facilement que minc admet les hypothèses 32, 33, 34 et 35, avec l'élément neutre à droite $+\infty$. Avec l'application de $\overline{\mathbb{N}}$ dans lui-même définie par $\mathsf{r}(x) = x + 1$, l'opérateur minc est r-associatif : $\mathsf{minc}(\mathsf{minc}(x, y), \mathsf{r}(z)) = \mathsf{min}(x, y+1, z+2) = \mathsf{minc}(x, \mathsf{minc}(y, z))$. Il est aussi r-commutatif : $\mathsf{minc}(\mathsf{r}(x), y) = \mathsf{min}(x + 1, y + 1) = \mathsf{minc}(\mathsf{r}(y), x)$. Enfin, il est r-idempotent : $\mathsf{minc}(\mathsf{r}(x), x) = x + 1 = \mathsf{r}(x)$. Il s'agit donc d'un r-opérateur. □

5.2.2 Propriétés des r-opérateurs

Caractérisation de r. Dans la suite, nous supposons que r est la bijection associée au r-opérateur $\triangleleft$ défini sur $\mathbb{S}$. Nous notons e l'élément neutre à droite de $\triangleleft$; $\mathsf{r}^0(x) = x$, et $\mathsf{r}^{k+1}(x) = \mathsf{r}\left(\mathsf{r}^k(x)\right)$.

Proposition 48 *L'application* r *vérifie* $\mathsf{r}(x) = e \triangleleft x$ *pour tout x dans $\mathbb{S}$.*

Preuve. Pour tous $x, y \in \mathbb{S}$, $x \triangleleft \mathsf{r}(y) = x \triangleleft e \triangleleft \mathsf{r}(y) = x \triangleleft (e \triangleleft y)$ (r-associativité). On a donc $\mathsf{r}(y) = e \triangleleft y$, en utilisant la régularité faible à gauche. □

Proposition 49 *Le noyau*[89] *de* r *est le singleton* $\{e\}$.

89. Noyau de r : $\mathrm{Ker}(r) = \{t \in \mathbb{S}, \mathsf{r}(t) = e\}$.

Preuve. Soit t un élément de $\mathrm{Ker}(r)$. Pour tout $x, y \in \mathbb{S}$, $x \triangleleft y = x \triangleleft y \triangleleft \mathsf{r}(t) = x \triangleleft (y \triangleleft t)$ (r-associativité). Donc, en utilisant la régularité faible à gauche, on a $y = y \triangleleft t$. Puisque $y = y \triangleleft e$, on obtient $e = t$ en utilisant de nouveau la régularité faible à gauche. □

Proposition 50 *La bijection* r *est un isomorphisme*[90] *de* $(\mathbb{S}, \triangleleft)$.

90. Isomorphisme : homomorphisme bijectif.

(e) L'écriture d'une nouvelle définition pose le problème des hypothèses minimalistes. Moins il y a d'hypothèses à vérifier et plus la définition est générale. Par contre l'exposé est plus compliqué car il faut introduire aux bons endroits les bonnes hypothèses. L'autre solution est d'inclure les hypothèses couramment utilisées dans la définition. Elle restreint le champ d'application de la définition mais simplifie l'exposé. C'est cette dernière solution que nous avons choisie ici. Dans [Ducourthial, 1997], un point de vue différent avait guidé notre présentation.

(f) Le terme «minc» vient de minimum et incrément.

Preuve. Pour tout $x, y, u, v \in \mathbb{S}$, $x \lhd y \lhd \mathsf{r}(u) \lhd \mathsf{r}^2(v) = (x \lhd y) \lhd (\mathsf{r}(u) \lhd \mathsf{r}(v))$ (associativité à gauche et r-associativité). Puisque r est surjective, il existe $t \in \mathbb{S}$ tel que $\mathsf{r}(t) = \mathsf{r}(u) \lhd \mathsf{r}(v) \in \mathbb{S}$. Donc $x \lhd y \lhd \mathsf{r}(u) \lhd \mathsf{r}^2(v) = x \lhd y \lhd \mathsf{r}(t) = x \lhd (y \lhd t)$ (associativité à gauche et r-associativité).

D'un autre côté, $x \lhd y \lhd \mathsf{r}(u) \lhd \mathsf{r}^2(v) = x \lhd (y \lhd u) \lhd \mathsf{r}^2(v) = x \lhd ((y \lhd u) \lhd \mathsf{r}(v)) = x \lhd (y \lhd (u \lhd v))$ (r-associativité).

Nous avons donc $x \lhd y \lhd \mathsf{r}(u) \lhd \mathsf{r}^2(v) = x \lhd (y \lhd t) = x \lhd (y \lhd (u \lhd v))$, et en utilisant la régularité faible à gauche, on obtient $t = u \lhd v$. Ainsi $\mathsf{r}(u \lhd v) = \mathsf{r}(t) = \mathsf{r}(u) \lhd \mathsf{r}(v)$ et r est un homomorphisme[91] et donc un isomorphisme de $(\mathbb{S}, \lhd)$ (puisque r est une bijection de $\mathbb{S}$ dans $\mathbb{S}$). □

91. **Homomorphisme :** l'application $\mathsf{f} : \mathbb{S}_1 \to \mathbb{S}_2$ est un homomorphisme de la structure $(\mathbb{S}_1, \diamond_1)$ dans la structure $(\mathbb{S}_2, \diamond_2)$ si elle vérifie, pour tout $a, b \in \mathbb{S}_1$, $\mathsf{f}(a \diamond_1 b) = \mathsf{f}(a) \diamond_2 \mathsf{f}(b)$.

Notons que l'isomorphisme r est déterminé de façon unique (proposition 48 page précédente). L'opérateur minc défini avec $\mathsf{r} : x \mapsto x + 1$ vérifie bien les propositions 48, 49 et 50 page ci-contre : $\mathsf{minc}(+\infty, x) = \mathsf{min}(+\infty, x + 1) = x + 1 = \mathsf{r}(x)$, $\mathsf{r}(x) = x + 1 = +\infty$ n'est vérifié que pour $x = +\infty \in \overline{\mathbb{N}}$ et $\mathsf{r}(\mathsf{minc}(x, y)) = \mathsf{min}(x, y + 1) + 1 = \mathsf{min}(x + 1, y + 2) = \mathsf{minc}(\mathsf{r}(x), \mathsf{r}(y))$.

s-opérateur associé. Au § 5.2.1, nous avons défini les r-opérateurs et la notion d'algèbre déformée par l'application r. En fait, une telle application dissymétrise le s-opérateur et l'algèbre «normale» qui lui est associée. Nous montrons maintenant, en utilisant les propriétés précédentes, que toute algèbre déformée définit tout de même un s-opérateur.

Théorème 51 *Si $\lhd$ est un r-opérateur sur $\mathbb{S}$ défini avec la bijection r de $\mathbb{S}$ dans $\mathbb{S}$, alors l'opérateur $\oplus$ défini par $x \oplus y = x \lhd \mathsf{r}^{-1}(y)$ est un s-opérateur sur $\mathbb{S}$. De plus, il admet le même élément neutre que $\lhd$.*

Preuve. On a d'une part, pour tout $x, y, z \in \mathbb{S}$, $x \oplus (y \oplus z) = x \lhd \mathsf{r}^{-1}(y \oplus z) = x \lhd \mathsf{r}^{-1}(y \lhd \mathsf{r}^{-1}(z)) = x \lhd (\mathsf{r}^{-1}(y) \lhd \mathsf{r}^{-2}(z)) = x \lhd \mathsf{r}^{-1}(y) \lhd \mathsf{r}^{-1}(z)$ (proposition 50 et définition 36). D'un autre côté, $x \lhd \mathsf{r}^{-1}(y) \lhd \mathsf{r}^{-1}(z) = (x \oplus y) \lhd \mathsf{r}^{-1}(z) = (x \oplus y) \oplus z$. Donc $\oplus$ est associatif.

Pour tout $x, y \in \mathbb{S}$, $x \oplus y = x \lhd \mathsf{r}^{-1}(y) = y \lhd \mathsf{r}^{-1}(x)$ (r-commutativité de $\lhd$, lemme 38). Donc $x \oplus y = y \oplus x$ et $\oplus$ est commutatif sur $\mathbb{S}$.

Puisque $\lhd$ est r-idempotent et r est une bijection, on a $x \lhd \mathsf{r}^{-1}(x) = x$ (lemme 40). Donc $x \oplus x = x$, et $\oplus$ est idempotent.

Enfin, $\oplus$ admet un élément neutre, qui est le même que celui de $\lhd$ car $x \oplus e = x \lhd \mathsf{r}^{-1}(e) = x$ (proposition 49).

Ainsi l'opérateur $\oplus$ est un s-opérateur sur $\mathbb{S}$. □

Par la suite, $\oplus$ dénote le s-opérateur associé au r-opérateur $\lhd$ par $x \oplus y = x \lhd \mathsf{r}^{-1}(y)$.

Proposition 52 *L'application r est un isomorphisme de $(\mathbb{S}, \oplus)$.*

Preuve. $\mathsf{r}(x \oplus y) = \mathsf{r}(x \lhd \mathsf{r}^{-1}(y)) = \mathsf{r}(x) \lhd y = \mathsf{r}(x) \oplus \mathsf{r}(y)$ (proposition 50). □

r-opérateurs idempotents. Au § 5.1.2, nous avons rappelé que les opérateurs devaient être k-idempotents faibles pour assurer la terminaison des `direct-associations`. Nous nous penchons maintenant sur les propriétés des r-opérateurs idempotents (*i.e.*, 1-idempotents fort).

Proposition 53 *Si $\lhd$ est un r-opérateur idempotent, son s-opérateur associé $\oplus$ vérifie $x \preceq_\oplus \mathsf{r}^k(x)$ pour tout $k \in \mathbb{N}$, où $\preceq_\oplus$ désigne la relation d'ordre partiel sur $\mathbb{S}$ définie par $x \preceq_\oplus y \equiv x \oplus y = x$ (cf. théorème 14 page 40).*

Preuve. La proposition est vraie pour $k = 0$, même si $\lhd$ n'est pas idempotent : $x \oplus x = x$ puisque $\oplus$ est un s-opérateur. Donc $x \preceq_\oplus x$.

Pour $k > 0$, on montre la proposition par récurrence sur k, en utilisant l'idempotence de $\lhd$. Supposons que $x \preceq \mathsf{r}^{k-1}(x)$ pour un $k > 0$. Alors $x \oplus \mathsf{r}^k(x) = (x \oplus \mathsf{r}^{k-1}(x)) \oplus \mathsf{r}^k(x)$ par hypothèse de récurrence, et $x \oplus \mathsf{r}^k(x) = x \oplus (\mathsf{r}^{k-1}(x) \oplus \mathsf{r}^k(x)) = x \oplus \mathsf{r}^{k-1}(x \oplus \mathsf{r}(x)) = x \oplus \mathsf{r}^{k-1}(x \triangleleft x) = x \oplus \mathsf{r}^{k-1}(x) = x$. On a donc $x \preceq_{\oplus} \mathsf{r}^k(x)$ pour tout $k \in \mathbb{N}$ □

Proposition 54 *Les trois propositions suivantes sont équivalentes : **(a)** $\forall x \in \mathbb{S}, x \preceq_{\oplus} \mathsf{r}(x)$, **(b)** $\oplus$ est (r^{-1})-idempotent et **(c)** $\triangleleft$ est idempotent.*

Preuve. $x \preceq_{\oplus} \mathsf{r}(x) \Leftrightarrow x \oplus \mathsf{r}(x) = x$ (théorème 14 page 40), ce qui signifie que $\oplus$ est (r^{-1})-idempotent (lemme 40). De plus, on a $x \oplus \mathsf{r}(x) = x \Leftrightarrow x \triangleleft x = x$, ce qui signifie que $\triangleleft$ est idempotent. □

Proposition 55 *Si $\triangleleft$ est un r-opérateur idempotent, alors il est (r^{-1})-idempotent.*

Preuve. Suivant la proposition 53 page précédente, on a $x \preceq_{\oplus} \mathsf{r}^2(x)$, c'est-à-dire $x \oplus \mathsf{r}^2(x) = x$ *i.e.*, $x \triangleleft \mathsf{r}(x) = x$ et donc $\mathsf{r}^{-1}(x) \triangleleft x = \mathsf{r}^{-1}(x)$ (proposition 50). Donc $\triangleleft$ est (r^{-1})-idempotent (définition 39). □

Construction des r-opérateurs. Le théorème 51 page précédente explique comment obtenir un s-opérateur depuis un r-opérateur. Nous résumons ici les relations qui existent entre les s- et les r-opérateurs pour faciliter la construction de ces derniers à partir d'opérateurs connus. Le théorème ci-dessous indique, entre autre, que l'hypothèse de la bijection de r utilisée dans la définition des r-opérateurs (définition 46) est trop forte et pourrait être remplacée par celle de la surjection.

Théorème 56 *Si r est une application de $\mathbb{S}$ dans $\mathbb{S}$, et si $\triangleleft$ et $\oplus$ sont des opérateurs binaires sur $\mathbb{S}$ vérifiant, pour tout x et y dans $\mathbb{S}$, $x \triangleleft y = x \oplus \mathsf{r}(y)$, alors les trois propositions suivantes sont équivalentes :*

(a) *r est surjective, $\triangleleft$ admet un élément neutre à droite et la régularité faible à gauche (hypothèses 34 et 35 page 82), et est r-commutatif, r-associatif et r-idempotent sur $\mathbb{S}$,*

(b) *$\triangleleft$ est un r-opérateur défini sur $\mathbb{S}$,*

(c) *$\oplus$ est un s-opérateur sur $\mathbb{S}$ et $\mathsf{r} : \mathbb{S} \to \mathbb{S}$ est un isomorphisme de $(\mathbb{S}, \oplus)$.*

Preuve. (a) ⇒ (b) Il reste à montrer que r est injective[92]. Pour tout $x, y, a, b \in \mathbb{S}$, $\mathsf{r}(x) = \mathsf{r}(y)$ implique $a \triangleleft b \triangleleft \mathsf{r}(x) = a \triangleleft b \triangleleft \mathsf{r}(y)$. Donc $a \triangleleft (b \triangleleft x) = a \triangleleft (b \triangleleft y)$ (r-associativité) et on obtient donc, en utilisant la régularité faible à gauche, $x = y$.

92. Injection : $\mathsf{f} : \mathbb{S}_1 \to \mathbb{S}_2$ est une injection si $\forall x_1, x_2 \in \mathbb{S}_2$, $f(x_1) = f(x_2)$ implique $x_1 = x_2$.

(b) ⇒ (c) Théorème 51 et proposition 50 page 86.

(c) ⇒ (a) Si r est un isomorphisme de $(\mathbb{S}, \oplus)$, c'est une surjection de $\mathbb{S}$ dans $\mathbb{S}$, et pour tout $x, y, z \in \mathbb{S}$: $(x \triangleleft y) \triangleleft \mathsf{r}(z) = x \oplus \mathsf{r}(y) \oplus \mathsf{r}^2(z) = x \oplus (\mathsf{r}(y) \triangleleft \mathsf{r}(z)) = x \triangleleft \mathsf{r}^{-1}(\mathsf{r}(y) \triangleleft \mathsf{r}(z)) = x \triangleleft (y \triangleleft z)$ et donc $\triangleleft$ est r-associatif ; $x \triangleleft \mathsf{r}^{-1}(y) = x \oplus y = y \oplus x = y \triangleleft \mathsf{r}^{-1}(x)$ et $\triangleleft$ est r-commutative (lemme 38) ; $\mathsf{r}(x) \triangleleft x = \mathsf{r}(x) \oplus \mathsf{r}(x) = \mathsf{r}(x)$ et donc $\triangleleft$ est r-idempotent. De plus, si $x \triangleleft y = x \triangleleft z$, $x \oplus \mathsf{r}(y) = x \oplus \mathsf{r}(z)$ et donc $y = z$ ($\oplus$ admet la régularité faible à gauche et r est bijective). Donc $\triangleleft$ admet la régularité faible sur $\mathbb{S}$. Finalement, si $e \in \mathbb{S}$ est l'élément neutre de $\oplus$, $x \triangleleft \mathsf{r}^{-1}(e) = x \oplus e = x$ et $\mathsf{r}^{-1}(e)$ est l'élément neutre de $\triangleleft$. □

Ainsi, pour chaque s-opérateur $\oplus$ sur $\mathbb{S}$, il est possible de construire autant de r-opérateurs $\triangleleft$ qu'il y a d'isomorphismes sur $(\mathbb{S}, \oplus)$. De plus, pour obtenir un r-opérateur idempotent, l'isomorphisme r doit vérifier $x \preceq_{\oplus} \mathsf{r}(x)$ ou, de façon équivalente, $\oplus$ doit être (r^{-1})-idempotent (proposition 54).

Exemple. L'opérateur minc est construit avec le s-opérateur min et la r-fonction $x \mapsto x + 1$. On définit d'autres r-opérateurs au § 7.4 page 137, avec max et $x \mapsto x \times c$, avec l'ordre lexicographique et la concaténation, *etc.*

5.2.3 Application des r-opérateurs à la `direct-association`

Comme indiqué au § 5.1.1 page 79, il s'agit de montrer qu'une $\triangleleft$-`direct-association` se termine sur un résultat déterministe et reconnaissable lorsque $\triangleleft$ est un r-opérateur idempotent.

Réduction des expressions. Nous montrons tout d'abord que les r-opérateurs idempotents admettent la propriété de réduction des expressions. Rappelons que la k-idempotence faible est nécessaire pour obtenir cette propriété (proposition 30 page 81). L'idempotence correspond à la 1-idempotence (forte). Nous appelons s-*expression* une expression écrite à l'aide d'un s-opérateur $\oplus$. De même, une r-expression est écrite en n'utilisant que le r-opérateur $\triangleleft$. En appliquant le théorème 56 page ci-contre, on peut transformer toute s-expression en une r-expression.

Le processus de réduction d'une r-expression pourrait se faire à l'aide d'une traduction de la r-expression en s-expression. Il est intéressant de remarquer que, dans ce cas, l'idempotence du r-opérateur, qui se traduit par $x \preceq_\oplus \mathsf{r}^k(x)$ (proposition 53), n'est utilisée que pour supprimer les différentes puissances de la même variable, qui apparaissent lors de la substitution de $x \oplus \mathsf{r}(y)$ à $x \triangleleft y$, au gré des parenthèses. L'idempotence du s-opérateur $\oplus$ n'est pas suffisante pour cela, mais la suppression des parenthèses dans une s-expression ne donne pas de puissance d'une quelconque application r. Dans la proposition 58 ci-dessous, nous montrons la propriété de réduction des r-opérateurs sans passer par le s-opérateur associé. Le lemme suivant est nécessaire pour cela.

Lemme 57 *Si $\triangleleft$ est un r-opérateur idempotent défini sur $\mathbb{S}$ et si $\oplus$ est son s-opérateur associé, défini par $x \oplus y = x \triangleleft \mathsf{r}^{-1}(y)$, alors pour tout $a, x \in \mathbb{S}$, pour tout $k \in \mathbb{N} \setminus \{0\}$, on a*

$$\begin{array}{llll}
(1) & x \triangleleft x = x & (6) & a \triangleleft x \triangleleft \mathsf{r}^k(x) = a \triangleleft x \\
(2) & x \triangleleft \mathsf{r}^k(x) = x & (7) & x \triangleleft \mathsf{r}^{-k}(x) = \mathsf{r}^{1-k}(x) \\
(3) & \mathsf{r}^k(x) \triangleleft x = \mathsf{r}(x) & (8) & \mathsf{r}^{-k}(x) \triangleleft x = \mathsf{r}^{-k}(x) \\
(4) & a \triangleleft x \triangleleft x = a \triangleleft x & (9) & a \triangleleft \mathsf{r}^{-k}(x) \triangleleft x = a \triangleleft \mathsf{r}^{-k}(x) \\
(5) & a \triangleleft \mathsf{r}^k(x) \triangleleft x = a \triangleleft x & (10) & a \triangleleft x \triangleleft \mathsf{r}^{-k}(x) = a \triangleleft \mathsf{r}^{-k}(x)
\end{array}$$

Preuve. La première équation est due à l'idempotence. La seconde est équivalente à l'inégalité $x \preceq_\oplus \mathsf{r}^{k+1}(x)$ (théorèmes 51 et 14), qui est vérifiée par les r-opérateurs idempotents (proposition 53). La troisième équation est donnée par la r-idempotence, pour $k = 1$. Pour $k \geq 2$, on utilise la r-commutativité, le lemme 38 et la deuxième équation : $\mathsf{r}^k(x) \triangleleft x = \mathsf{r}(\mathsf{r}^{k-1}(x)) \triangleleft x = \mathsf{r}(x) \triangleleft \mathsf{r}^{k-1}(x) = \mathsf{r}(x)$. La quatrième équation est due à la 2-idempotence (proposition 45). On a $a \triangleleft \mathsf{r}^k(x) \triangleleft x = a \triangleleft (\mathsf{r}^k(x) \triangleleft \mathsf{r}^{-1}(x)) = a \triangleleft \mathsf{r}^{-1}(\mathsf{r}^{k+1}(x) \triangleleft x) = a \triangleleft \mathsf{r}^{-1}(\mathsf{r}(x)) = a \triangleleft x$, en utilisant la r-associativité, puis la proposition 50, puis la troisième équation. Cela donne la cinquième équation. La sixième est due à la 2-commutativité (proposition 44) et à la cinquième équation. La septième est montrée comme suit : $\mathsf{r}^k(x \triangleleft \mathsf{r}^{-k}(x)) = \mathsf{r}^k(x) \triangleleft x = \mathsf{r}(x)$ (proposition 50 et troisième équation). D'où $x \triangleleft \mathsf{r}^{-k} = \mathsf{r}^{1-k}(x)$. La huitième est obtenue à partir de la troisième, en utilisant la proposition 50. Pour montrer la neuvième, on utilise la r-associativité, la proposition 50 et la seconde équation : $a \triangleleft \mathsf{r}^{-k}(x) \triangleleft x = a \triangleleft (\mathsf{r}^{-k}(x) \triangleleft \mathsf{r}^{-1}(x)) = a \triangleleft \mathsf{r}^{-k}(x \triangleleft \mathsf{r}^{k-1}(x)) = a \triangleleft \mathsf{r}^{-k}(x)$. Finalement, on obtient la dixième équation à partir de la neuvième, en utilisant la proposition 44. □

Proposition 58 *Un r-opérateur idempotent vérifie la propriété de réduction des expressions sur son ensemble de définition $\mathbb{S}$. De plus, l'expression réduite est déterminée de façon unique à partir de l'expression initiale, et contient une et une seule fois chacune des variables de cette expression.*

Preuve. Nous montrons la proposition en construisant un processus de réduction d'expression, qui se termine sur une expression réduite déterminée de façon unique. Le processus est décrit dans la colonne de gauche et est appliqué à un exemple dans la colonne de droite. Soit une expression composée d'un nombre quelconque (non borné) de r-opérateurs $\triangleleft$ définis sur $\mathbb{S}$, d'un nombre quelconque de parenthèses, d'un nombre borné d'éléments de $\mathbb{S}$ répétés un nombre quelconque (non borné) de fois (définition 29 page 81).

(1) En utilisant l'associativité à gauche, on peut identifier les groupes de variables du type $x \triangleleft y \triangleleft (z \triangleleft t)$ à $(x \triangleleft y) \triangleleft (z \triangleleft t)$.	$a \triangleleft (b \triangleleft (a \triangleleft a) \triangleleft b \triangleleft c) \triangleleft c \triangleleft (b \triangleleft a) =$ $(a \triangleleft (b \triangleleft (a \triangleleft a) \triangleleft b \triangleleft c) \triangleleft c) \triangleleft (b \triangleleft a)$
(2) En utilisant la r-associativité, on peut développer la parenthèse $(z \triangleleft t)$ et obtenir $x \triangleleft y \triangleleft z \triangleleft \mathsf{r}(t)$.	$= a \triangleleft (b \triangleleft (a \triangleleft a) \triangleleft b \triangleleft c) \triangleleft c \triangleleft b \triangleleft \mathsf{r}(a)$
(3) On applique le **(2)** tant qu'il reste des parenthèses qui ne sont pas à gauche, en commençant par les parenthèses les plus internes.	$= a \triangleleft (b \triangleleft a \triangleleft \mathsf{r}(a) \triangleleft b \triangleleft c) \triangleleft c \triangleleft b \triangleleft \mathsf{r}(a) = \ldots =$ $a \triangleleft b \triangleleft \mathsf{r}(a) \triangleleft \mathsf{r}^2(a) \triangleleft \mathsf{r}(b) \triangleleft \mathsf{r}(c) \triangleleft c \triangleleft b \triangleleft \mathsf{r}(a)$
(4) On regroupe les variables identiques. La 2-commutativité est suffisante pour cela.	$= a \triangleleft \mathsf{r}(a) \triangleleft \mathsf{r}^2(a) \triangleleft \mathsf{r}(a) \triangleleft b \triangleleft \mathsf{r}(b) \triangleleft b \triangleleft \mathsf{r}(c) \triangleleft c$
(5) En utilisant les équations 4, 5, 6, 9 et 10 du lemme 57 page précédente, on ne retient que la plus petite puissance de r de toutes les occurrences des variables qui ne sont pas la première de l'expression.	$= a \triangleleft \mathsf{r}(a) \triangleleft \mathsf{r}^2(a) \triangleleft \mathsf{r}(a) \triangleleft b \triangleleft c$
(6) En utilisant les équations 1, 2, 3, 7 et 8 du lemme 57, on retient la puissance de la première occurrence de la première variable de l'expression, si cette puissance est la plus petite de toutes les occurrences, et la plus petite puissance des autres occurrences augmentée de 1 sinon.	$= a \triangleleft b \triangleleft c$

Ce processus de réduction est déterministe et l'expression-résultat est unique. □

Cette proposition indique qu'une $\triangleleft$-`direct-association` se termine lorsque $\triangleleft$ est un r-opérateur idempotent. En effet, chaque nœud recevra un nombre borné de données qui sont les valeurs initiales (b^0_{in}) de ses antécédents. Il ne reste plus qu'à reconnaître le résultat et s'assurer qu'il est bien déterministe.

Reconnaissance et déterminisme du résultat. Si le processus de réduction est, lui, déterministe (*i.e.*, l'expression réduite est uniquement déterminée), l'expression obtenue par un nœud peut varier d'une exécution à une autre. Il nous faut montrer que toutes ces expressions se réduisent en une même expression-résultat, ne dépendant pas des aléas de l'exécution (vitesses de calcul et de transfert non contrôlées), contrairement à l'exemple illustré sur la figure 40 page 39.

Théorème 59 *Si $\triangleleft$ est un r-opérateur idempotent, alors la $\triangleleft$-`direct-association`(net_G, p) se termine sur tout réseau G initialisé avec toute pvar p. De plus, le résultat sur le nœud w est donné par la r-expression*[(g)]

$$b_{out}[w] = b^0_{in}[v] \triangleleft \mathop{\Large\triangleleft}_{v\in\Gamma^-_G(w)} \mathsf{r}^{\mathrm{d}_G(v,w)-1}\left(b^0_{in}[v]\right)$$

qui est équivalente, si $\oplus$ est le s-opérateur associé à $\triangleleft$, à la s-expression

$$b_{out}[w] = \bigoplus_{v\in\overline{\Gamma^-_G}(w)\}} \mathsf{r}^{\mathrm{d}_G(v,w)}\left(b^0_{in}[v]\right)$$

Preuve. L'équivalence des deux expressions est donnée par le théorème 56 page 88. Nous raisonnons ici sur la s-expression qui simplifie le début de la récurrence sur les distances.

(g) C'est la 2-commutativité du r-opérateur qui autorise l'utilisation de l'opérateur $\triangleleft$ appliqué à un ensemble de valeurs non ordonné.

Il existe une date t à partir de laquelle tout nœud w a intégré dans son expression $(b_{\text{out}}[w])^t$ toutes les valeurs initiales $b^0_{\text{in}}[u]$ de ses antécédents u dans le réseau G. Nous montrons par récurrence sur la distance $\mathrm{d}_G(u,w)$ que l'expression $(b_{\text{out}}[w])^t$ écrite sous forme d'une s-expression contient $\mathsf{r}^{\mathrm{d}_G(u,w)}(b^0_{\text{in}}[u])$, et que $b^0_{\text{in}}[u]$ n'apparaît pas sous une plus petite puissance (non nulle) de r dans cette expression.

Cette affirmation est vraie pour $\mathrm{d}_G(u,w) = 0$ (*i.e.*, $u = w$), car l'expression $(b_{\text{out}}[w])^t$ commence par $b^0_{\text{in}}[w]$. Supposons maintenant qu'elle le soit pour toute paire de sommets (u,v) telle que $\mathrm{d}_G(u,v) = k$. Considérons un sommet w qui soit à distance $k+1$ d'un sommet u et à distance 1 d'un sommet v, telle $\mathrm{d}_G(u,v) = k$ (figure 61). D'après l'hypothèse de récurrence, l'expression $(b_{\text{out}}[v])^t$ écrite sous forme d'une s-expression contient le terme $\mathsf{r}^k\left(b^0_{\text{in}}[u]\right)$, et k est la plus petite puissance de toutes les éventuelles occurrences de $\mathsf{r}(b^0_{\text{in}}[u])$. Le nœud w a reçu la valeur $b^0_{\text{in}}[u]$ à travers l'un de ses antécédents directs, tel que v. Il intègre donc le terme $\mathsf{r}^k(b^0_{\text{in}}[u])$ dans sa r-expression, à partir du deuxième rang car la première valeur de l'expression vient du tampon $b^0_{\text{in}}[w]$. De plus, il ne peut recevoir la variable $b^0_{\text{in}}[u]$ sous une plus petite puissance. Lors du processus de transformation de la r-expression en s-expression, ce terme devient $\mathsf{r}^{k+1}(b^0_{\text{in}}[u])$ (théorème 51), ce qui donne le résultat, par récurrence.

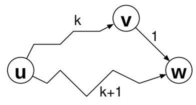


FIG. 61 – Récurrence sur la distance des antécédents.

Les propositions 31 page 81 ou 58 page 89 assurent que les plus grandes puissances des occurrences des variables peuvent être éliminées de la s- ou de la r-expression, ce qui achève la preuve. □

5.2.4 Conclusion.

Partant des s-opérateurs — qui sont idempotents, qui réduisent les expressions et qui assurent la terminaison déterministe sur un résultat reconnaissable des `direct-associations` — nous avons défini les r-opérateurs, plus généraux, en introduisant une application r qui déforme l'algèbre normale. Les r-opérateurs idempotents réduisent aussi les expressions et assurent aussi la terminaison sur un résultat déterministe des `direct-associations`. Nous avons donc étendu la classe des opérateurs qu'il est possible d'utiliser dans un calcul global composé de calculs locaux non coordonnés. On peut remarquer que si $\mathsf{r} \equiv \mathsf{Id}$, le résultat du théorème 59 est équivalent à celui du théorème 15 page 41.

En outre, alors qu'une `direct-association` donne un seul résultat par composante fortement connexe avec un s-opérateur, le résultat est, en général, différent d'un nœud à un autre avec un r-opérateur. En effet, au lieu de structurer les données intégrées au calcul avec l'information relative aux composantes fortement connexes, un r-opérateur conduit à intégrer l'information relative aux distances entre les nœuds. Cette information est beaucoup plus riche que — et recouvre — celle concernant les composantes fortement connexes. Les r-opérateurs exploitent donc mieux l'information codée par le réseau lui-même. La figure 62 illustre l'extension proposée par les r-opérateurs : un calcul local par un r-opérateur revient à un calcul local par un s-opérateur dans lequel les données en provenance des antécédents directs sont intégrées après modification par une application r (qui doit vérifier certaines propriétés).

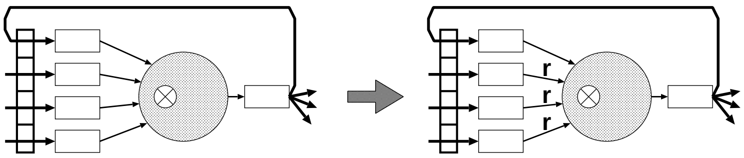


FIG. 62 – *Extension proposée par les* r*-opérateurs binaires.*

Dans cette section consacrée aux r-opérateurs binaires, nous avons montré des résultats et utilisé des raisonnements propres aux r-opérateurs binaires, et qui n'interviennent pas — ou auraient pu ne pas intervenir, *cf.* § 5.3.3 et la preuve du théorème 69 page 96 — dans les applications de ces opérateurs à la `direct-association`. Cependant il nous a semblé que ces opérateurs pouvaient trouver d'autres utilités que celle d'assurer la terminaison déterministe sur un résultat reconnaissable de la `direct-association`. Les applications en algorithmique, en auto-stabilisation, ainsi que la généralisation de l'algèbre classique — qui, grâce aux applications, dépasse le cadre du simple jeu formel — nous confortent dans ce point de vue.

5.3 Les r-opérateurs n-aires

Comme l'indique la figure 62 page précédente, l'usage des r-opérateurs dans les calculs locaux peut être considéré comme la modification des valeurs arrivant des antécédents directs par une application r, avant d'effectuer le calcul par un s-opérateur. Dans cette section, nous étudions le cas de modification par plusieurs applications r_i (*cf.* figure 63). L'usage de plusieurs r-fonctions empêche d'écrire le calcul local comme l'application d'un opérateur binaire sur une liste de variables. Nous définissons donc les r-*opérateurs n-aires*. Dans cette section (et la suivante), le raisonnement est essentiellement basé sur l'utilisation du réseau sous-jacent.

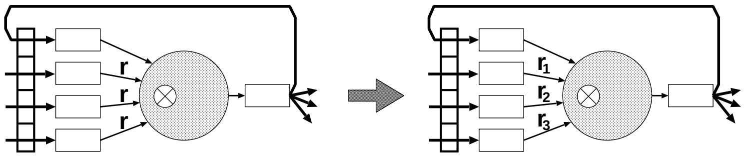


FIG. 63 – *Extension proposée par les* r-*opérateurs n-aires.*

5.3.1 Définitions

Un r-opérateur binaire peut être défini sur un ensemble $\mathbb{S}$ à partir d'un s-opérateur $\oplus$ sur $\mathbb{S}$ et d'un isomorphisme r de $(\mathbb{S}, \oplus)$ par la relation $x \triangleleft y = x \oplus \mathsf{r}(y)$ (théorème 56 page 88). Par exemple, le r-opérateur minc a été défini sur $\overline{\mathbb{N}}$ par $\mathsf{minc}(x, y) = \mathsf{min}(x, y+1)$ (proposition 47 page 86). Suivant cette construction, nous définissons le r-opérateur n-aire minc_w sur $\overline{\mathbb{N}}$ par (w correspond aux poids des arcs, *cf.* § 7.4.1 page 138) :

$$\mathsf{minc}_w\,(x_0, \ldots, x_{n-1}) = \mathsf{min}\,(x_0, x_0 + w_1, \ldots, x_{n-1} + w_{n-1}) \tag{5.1}$$

Il permet de résoudre le problème de la forêt des chemins de poids minimum. Nous définissons au chapitre 7 d'autres r-opérateurs n-aires, avec l'opérateur max et l'addition, ou le minimum (*cf.* tableau de la figure 111 page 151). Plus généralement, on définit un r-opérateur n-aire comme suit.

Définition 60 *Une application ∇ de $\mathbb{S}^n$ dans $\mathbb{S}$ est appelée* r-*opérateur n-aire sur $\mathbb{S}$ s'il existe un* s-*opérateur $\oplus$ sur $\mathbb{S}$ et $n-1$ isomorphismes r_i de $(\mathbb{S}, \oplus)$ tels que, pour tout $x_0, \ldots, x_{n-1}$ dans $\mathbb{S}$, on ait :*

$$\nabla(x_0, \ldots, x_{n-1}) = x_0 \oplus \mathsf{r}_1(x_1) \oplus \ldots \oplus \mathsf{r}_{n-1}(x_{n-1})$$

Ainsi un r-opérateur n-aire est construit avec $(n-1)$ r-opérateurs $\triangleleft_i$ basés sur le même s-opérateur et sur les r-fonctions r_i de telle sorte que $x \triangleleft_i y = x \oplus \mathsf{r}_i(y)$. Par la suite, e désigne l'élément neutre du s-opérateur $\oplus$. C'est l'élément spécial qu'il faut placé dans les tampons masqués (*cf.* § 3.2.2 page 37).

Définition 61 *Si ∇ est un* r-*opérateur n-aire sur $\mathbb{S}$ construit avec le* s-*opérateur $\oplus$ et les $(n-1)$ isomorphismes r_i tels que, pour tout i ($1 \leq i < n$), pour tout $x \in \mathbb{S}$, $x \oplus \mathsf{r}_i(x) = x$, alors ∇ est* r-*opérateur n-aire idempotent.*

Les r-fonctions de l'opérateur minc_w sont définies par $\mathsf{r}_i\ : x \mapsto x + w_i$, avec $w_i \in \overline{\mathbb{N}}$ ($1 \leq i \leq n-1$). Ce sont des isomorphismes de $(\overline{\mathbb{N}}, \min)$ qui vérifient $x \leq \mathsf{r}_i(x)$. minc_w est donc idempotent.

Un réseau associatif est modélisé par un graphe simple (*cf.* § 3.2.1 page 36). Aussi, lorsque chaque nœud v du réseau utilise un r-opérateur n-aire (*cf.* note **(m)** page 37), on peut associer à chaque arc (u, v) du réseau une r-fonction que nous notons en utilisant l'indice de l'arc entrant (*cf.* figure 63 page ci-contre), ou l'antécédent correspondant à cet arc (*e.g.*, r_v^u). De la même façon, on peut associer une application à chaque chemin du graphe (figure 64).

Définition 62 *Soit $P = (v_0, v_1)\dots(v_{k-1}, v_k)$ un chemin dans le graphe G modélisant un réseau associatif dans lequel chaque nœud v utilise un* r*-opérateur δ_v^--aire. On appelle* r*-fonction du chemin P l'application r_P définie par*

$$\mathsf{r}_P(x) = \mathsf{r}_{v_k}^{v_{k-1}} \circ \dots \circ \mathsf{r}_{v_1}^{v_0}(x)$$

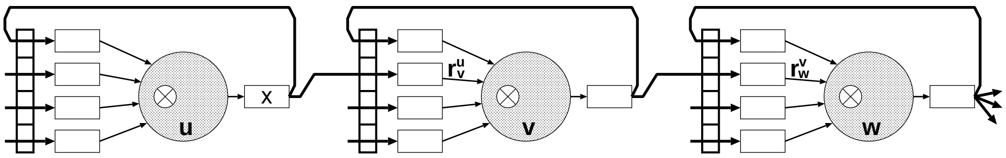


FIG. 64 – *Le nœud w inclut la valeur $\mathsf{r}_w^v \circ \mathsf{r}_v^u(x)$ dans son calcul local ; $\mathsf{r}_w^v \circ \mathsf{r}_v^u$ est une* r*-fonction de chemin.*

5.3.2 Propriétés

Puisqu'un r-opérateur n-aire est construit avec des r-opérateurs binaires, certaines de leurs propriétés sont conservées, moyennant une «transcription» pour prendre en compte l'aspect «n-aire». La recherche de ces propriétés, que nous menons dans cette sous-section, apporte un complément d'information sur les r-opérateurs n-aires.

Propriétés de bases. Nous commençons par étendre les hypothèses 32, 33, 34 et 35 page 82 aux opérateurs n-aires.

Proposition 63 *Si ∇ est un* r*-opérateur n-aire construit avec le* s*-opérateur $\oplus$, et les $(n-1)$* r*-fonctions r_i, alors il vérifie les propriétés suivantes :*

loi interne n-aire : $\forall x_0, \dots, x_{n-1} \in \mathbb{S}, \exists! z \in \mathbb{S}$ *tel que* $\nabla(x_0, \dots, x_{n-1}) = z$

élément neutre n-aire : $\forall x \in \mathbb{S}, \nabla(x, e, \dots, e) = x$

associativité à gauche n-aire : $\forall x_0, \dots, x_{n-1} \in \mathbb{S}, \forall k,\ 0 \leq k \leq n-1,$
$$\nabla\big(\nabla(x_0, \dots, x_k, e, \dots, e), e, \dots, e, x_{k+1}, \dots, x_{n-1}\big) = \nabla(x_0, \dots, x_{n-1})$$

régularité faible à gauche n-aire : $\forall y, z \in \mathbb{S}, \forall k,\ 1 \leq k \leq n-1,$
$$\begin{pmatrix} \forall x_0, \dots, x_{k-1}, x_{k+1}, \dots, x_{n-1} \in \mathbb{S} \\ \nabla(x_0, \dots, x_{k-1}, y, x_{k+1}, \dots, x_{n-1}) = \\ \nabla(x_0, \dots, x_{k-1}, z, x_{k+1}, \dots, x_{n-1}) \end{pmatrix} \Longrightarrow (y = z)$$

Preuve. Les trois premières propriétés sont immédiates. Pour la quatrième, on utilise la régularité faible à gauche du s-opérateur $\oplus$ et la propriété d'isomorphisme de $(\mathbb{S}, \oplus)$ des r-fonctions r_i. □

Ces propriétés n-aires généralisent les propriétés binaires. Par exemple, pour l'opérateur minc, $n = 2$ et on a $\mathsf{minc}(x, y) = x \triangleleft y = z \in \mathbb{S}$, $\mathsf{minc}(x, e) = x \triangleleft e = e$ and $\mathsf{minc}(\mathsf{minc}(x, e), y) = \min(x, e+1, y+1) = \mathsf{minc}(x, y)$. Nous étendons maintenant les trois propriétés qui définissent les r-opérateurs binaires.

Proposition 64 *Si ∇ est un* r*-opérateur n-aire construit avec le* s*-opérateur $\oplus$ et les $n-1$* r*-fonctions* r_i*, alors il vérifie les propriétés suivantes :*

r-*associativité n-aire :* $\forall x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1} \in \mathbb{S}, \quad \forall i \in \{1, \ldots, n-1\}$,

$$\nabla\left(x_0, \ldots, x_{i-1}, \mathsf{r}_i^{-1}(\nabla(y_0, \ldots, y_{n-1})), x_{i+1}, \ldots, x_{n-1}\right) = \nabla\left(x_0 \oplus y_0, \ldots, x_{i-1} \oplus y_{i-1}, y_i, x_{i+1} \oplus y_{i+1}, \ldots, x_{n-1} \oplus y_{n-1}\right)$$

r-*commutativité n-aire :* $\forall x_0, \ldots, x_{n-1} \in \mathbb{S} \quad \forall i, j \in \{1, \ldots, n-1\}$,

$$\nabla\left(x_0, \ldots, x_{i-1}, \mathsf{r}_i^{-1}(x_i), x_{i+1}, \ldots, x_{j-1}, \mathsf{r}_j^{-1}(x_j), x_{j+1}, \ldots, x_{n-1}\right) = \nabla\left(x_0, \ldots, x_{i-1}, \mathsf{r}_i^{-1}(x_j), x_{i+1}, \ldots, x_{j-1}, \mathsf{r}_j^{-1}(x_i), x_{j+1}, \ldots, x_{n-1}\right)$$

et pour $i = 0$:

$$\nabla\left(x_i, \ldots, x_{j-1}, \mathsf{r}_j^{-1}(x_j), x_{j+1}, \ldots, x_{n-1}\right) = \nabla\left(x_j, \ldots, x_{j-1}, \mathsf{r}_j^{-1}(x_i), x_{j+1}, \ldots, x_{n-1}\right)$$

r-*idempotence n-aire :* $\forall x_0, \ldots, x_{n-1} \in \mathbb{S}, \quad \forall i \in \{1, \ldots, n-1\}$,

$$\nabla\left(x_0, \ldots, x_{i-1}, \mathsf{r}_i^{-1}(\mathsf{r}_j(x_j)), x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{n-1}\right) = \nabla\left(x_0, \ldots, x_{i-1}, e, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{n-1}\right) = \nabla\left(x_0, \ldots, x_{i-1}, \mathsf{r}_i^{-1}(\mathsf{r}_j(x_j)), x_{i+1}, \ldots, x_{j-1}, e, x_{j+1}, \ldots, x_{n-1}\right)$$

et pour $i = 0$:

$$\nabla\left(r_j(x_j), \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{n-1}\right) = \nabla\left(e, x_1, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{n-1}\right) = \nabla\left(\mathsf{r}_j(x_j), x_1, \ldots, x_{j-1}, e, x_{j+1}, \ldots, x_{n-1}\right)$$

Comme pour la proposition 63 page précédente, ces propriétés n-aires généralisent les propriétés binaires et sont vérifiées pour $n = 2$. Par exemple, pour l'opérateur minc, pour lequel $\oplus \equiv \min$, on a $\mathsf{minc}\left(x, \mathsf{r}^{-1}(\mathsf{minc}(y, z))\right) = x \oplus (y \oplus \mathsf{r}(z)) = x \oplus y \oplus \mathsf{r}(z) = \mathsf{minc}(x \oplus y, z)$ (r-associativité), $\mathsf{minc}\left(x, \mathsf{r}^{-1}(y)\right) = x \oplus y = y \oplus x = \mathsf{minc}(y, \mathsf{r}^{-1}(x))$ (r-commutativité) et $\mathsf{minc}(\mathsf{r}(x), x) = \mathsf{r}(x) \triangleleft x = \mathsf{r}(x)$ (r-idempotence).

Construction des r-opérateurs n-aires. Nous avons construit les r-opérateurs binaires à partir des propriétés de r-associativité, r-commutativité et r-idempotence (définition 46 page 86). Puis nous avons montré que tout r-opérateur binaire pouvait être construit à partir d'un s-opérateur et d'un isomorphisme (théorème 56 page 88). Pour les r-opérateurs n-aires, nous avons suivi le cheminement inverse (qui est plus simple pour les n-aires) : nous avons construit les r-opérateurs n-aires à partir d'un s-opérateur et d'isomorphismes r_i (définition 60 page 92), puis nous avons montré qu'ils admettaient des propriétés semblables à celles définissant les r-opérateurs binaires (propositions 63 et 64). Nous montrons maintenant que ces propriétés permettent de définir un r-opérateur n-aire ; il est cependant nécessaire d'ajouter la propriété suivante : un r-opérateur n-aire vérifie la propriété de r-*isomorphisme* si chacune de ses r-fonctions r_i vérifie :

$$\mathsf{r}_i\left(\nabla\left(x_0, \mathsf{r}_1^{-1}(x_1), \ldots, \mathsf{r}_{n-1}^{-1}(x_{n-1})\right)\right) = \nabla\left(\mathsf{r}_i(x_0), \mathsf{r}_1^{-1} \circ \mathsf{r}_i(x_1), \ldots, \mathsf{r}_{n-1}^{-1} \circ \mathsf{r}_i(x_{n-1})\right) \tag{5.2}$$

Lemme 65 *Si ∇ est une application de $\mathbb{S}^n$ dans $\mathbb{S}$ telle qu'il existe un opérateur binaire $\oplus$ et $(n-1)$ bijections r_i vérifiant, pour tout $x_0, \ldots, x_{n-1} \in \mathbb{S}$, $\nabla(x_0, \ldots, x_{n-1}) = x_0 \oplus r_1(x_1) \oplus \ldots \oplus r_{n-1}(x_{n-1})$, et si chaque application r_i vérifie l'équation 5.2, alors chaque fonction r_i est un isomorphisme de $(\mathbb{S}, \oplus)$.*

Preuve. Pour tout $x_0, \ldots, x_{n-1} \in \mathbb{S}^n$, on a $\mathsf{r}_i\left(\nabla\left(x_0, \mathsf{r}_1^{-1}(x_1), \ldots, \mathsf{r}_{n-1}^{-1}(x_{n-1})\right)\right) = \mathsf{r}_i(x_0 \oplus \ldots \oplus x_{n-1})$. De plus $\nabla\left(\mathsf{r}_i(x_0), \mathsf{r}_1^{-1} \circ \mathsf{r}_i(x_1), \ldots, \mathsf{r}_{n-1}^{-1} \circ \mathsf{r}_i(x_{n-1})\right) = \mathsf{r}_i(x_0) \oplus \ldots \oplus \mathsf{r}_i(x_{n-1})$. Donc $\mathsf{r}_i(x_0 \oplus \ldots \oplus x_{n-1}) = \mathsf{r}_i(x_0) \oplus \ldots \oplus \mathsf{r}_i(x_{n-1})$ et r_i est un isomorphisme de $(\mathbb{S}, \oplus)$. □

Lemme 66 *Si ∇ est une application de $\mathbb{S}^n$ dans $\mathbb{S}$ telle qu'il existe un opérateur binaire $\oplus$ et $(n-1)$ bijections r_i vérifiant, pour tout $x_0, \ldots, x_{n-1} \in \mathbb{S}$, $\nabla(x_0, \ldots, x_{n-1}) = x_0 \oplus \mathsf{r}_1(x_1) \oplus \ldots \oplus \mathsf{r}_{n-1}(x_{n-1})$, et si ∇ vérifie les propriétés de* r*-associativité n-aire, de* r*-commutativité n-aire et de* r*-idempotence n-aire sur $\mathbb{S}$, alors $\oplus$ est un* s*-opérateur sur $\mathbb{S}$.*

Preuve. Pour tout $x, y, z \in \mathbb{S}$, on a
$x \oplus (y \oplus z) = \nabla\left(x, \mathsf{r}_1^{-1}(y \oplus z), e, \dots, e\right) = \nabla\left(x, \mathsf{r}_1^{-1}(\nabla(y, \mathsf{r}_1^{-1}(z), e, \dots, e)), e, \dots, e\right)$,
ce qui donne, en utilisant la r-associativité n-aire, $\nabla(x \oplus y, \mathsf{r}_1^{-1}(z), e, \dots, e) = (x \oplus y) \oplus z$. Donc $\oplus$ est associatif sur $\mathbb{S}$.

Pour tout $x, y \in \mathbb{S}$, on a $x \oplus y = \nabla(x, \mathsf{r}_1^{-1}(y), e, \dots, e) = \nabla(y, \mathsf{r}_1^{-1}(x), e, \dots, e)$, en utilisant la r-commutativité n-aire. Donc $x \oplus y = y \oplus x$ et $\oplus$ est commutatif sur $\mathbb{S}$.

Pour tout $x \in \mathbb{S}$, on a $\mathsf{r}_1(x) \oplus \mathsf{r}_1(x) = \nabla(\mathsf{r}_1(x), x, e, \dots, e) = \nabla(\mathsf{r}_1(x), e, \dots, e)$, en utilisant la r-idempotence n-aire. Donc $\mathsf{r}_1(x) \oplus \mathsf{r}_1(x) = \mathsf{r}_1(x)$ et $\oplus$ est idempotente (r_1 est bijective sur $\mathbb{S}$). □

Théorème 67 *Soit ∇ une application de $\mathbb{S}^n$ dans $\mathbb{S}$ telle qu'il existe $(n-1)$ bijections r_i et un opérateur binaire $\oplus$ qui vérifient, pour tout $x_0, \dots, x_{n-1} \in \mathbb{S}^n$, $\nabla(x_0, \dots, x_n) = x_0 \oplus \mathsf{r}_1(x_1) \oplus \dots \oplus \mathsf{r}_{n-1}(x_{n-1})$. Alors les deux propositions suivantes sont équivalentes :*

(a) *∇ est un r-opérateur n-aire, i.e., $\oplus$ est un s-opérateur sur $\mathbb{S}$ et chaque fonction r_i est un isomorphisme de $(\mathbb{S}, \oplus)$.*

(b) *∇ vérifie les propriétés de r-associativité n-aire, r-commutativité n-aire, r-idempotence n-aire (définies dans la proposition 64) et de r-isomorphisme (définie par l'équation 5.2 page précédente).*

Preuve. **(b)** $\Rightarrow$ **(a)** est donné par les lemmes 65 page ci-contre et 66 page précédente.

(a) $\Rightarrow$ **(b)** est donné par la proposition 64 et par le fait que si chaque r_i est un isomorphisme de $(\mathbb{S}, \oplus)$, alors ∇ vérifie la propriété de r-isomorphisme : $\mathsf{r}_i(x_0 \oplus \dots \oplus x_{n-1}) = \mathsf{r}_i(x_0) \oplus \dots \oplus \mathsf{r}_i(x_{n-1})$ implique $\mathsf{r}_i(\nabla(x_0, \mathsf{r}_1^{-1}(x_1), \dots, \mathsf{r}_{n-1}^{-1}(x_{n-1}))) = \nabla(\mathsf{r}_i(x_0), \mathsf{r}_1^{-1} \circ \mathsf{r}_i(x_1), \dots, \mathsf{r}_{n-1}^{-1} \circ \mathsf{r}_i(x_{n-1}))$. □

Ce théorème est comparable au théorème 56 page 88, qui donnait des règles pour construire les r-opérateurs binaires à partir des propriétés de base, ou à partir d'un s-opérateur. On constate donc que la généralisation de la construction des r-opérateurs binaires aux n-aires nécessite une nouvelle propriété, que nous avons appelée propriété de r-isomorphisme.

5.3.3 Application des r-opérateurs n-aires à la `direct-association`

En règle générale, un r-opérateur n-aire idempotent ne vérifie pas la réduction des expressions. En effet, il faut, pour cela, supposer la commutativité de la composition des r-fonctions : $\mathsf{r}_i \circ \mathsf{r}_j = \mathsf{r}_j \circ \mathsf{r}_i$. Si cette hypothèse est vérifiée, en appliquant la définition des r-opérateurs, n-aires. on transforme la r-expression en s-expression, et on «développe tous les isomorphismes» ($\mathsf{r}(x \oplus y) \to \mathsf{r}(x) \oplus \mathsf{r}(y)$). On obtient alors des compositions de r-fonctions. Puisque ces dernières commutent, on peut les ordonner selon leurs indices. En utilisant alors l'idempotence du s-opérateur et des r-opérateurs binaires (qui composent le r-opérateur n-aire), on simplifie l'expression pour finalement obtenir une expression de taille finie (*cf.* propositions 31 page 81 et 58 page 89).

L'hypothèse de commutativité des r-fonctions est assez forte ; elle réduirait les applications des r-opérateurs n-aires. C'est pourquoi nous ne l'avons pas intégrée à la définition de ces opérateurs. La propriété de r-isomorphisme — que nous avons utilisée pour obtenir une construction des r-opérateurs n-aires similaire à celle des binaires — est plus faible que la propriété de commutativité de la composition des r-fonctions. Nous montrons cependant que les r-opérateurs n-aires idempotents assurent la terminaison déterministe des `direct-associations` sur tout type de réseau. Nous utilisons un raisonnement sur le réseau sous-jacent, car celui-ci limite les combinaisons des r-fonctions : une combinaison de r-fonctions — par exemple $\mathsf{r}_v^u \circ \mathsf{r}_w^v$ — correspond à un chemin — ici $(u, v), (v, w)$ —, c'est-à-dire à une r-fonction de chemin.

Lemme 68 *Soit P un chemin dans le graphe $G(V, E)$ modélisant un réseau associatif dans lequel chaque nœud v utilise un r-opérateur $(\delta_v^- + 1)$-aire idempotent, basé sur le même s-opérateur $\oplus$. Alors la r-fonction de chemin r_P vérifie $x \preceq_\oplus \mathsf{r}_P(x)$.*

Preuve. L'idempotence des r-opérateurs n-aires construits sur le même s-opérateur $\oplus$ se traduit par l'idempotence de chacun des r-opérateurs binaires les composant (définition 61 page 92), ce qui se traduit par $x \preceq_\oplus \mathsf{r}_v^u(x)$ (proposition 53 page 87) pour tout arc $(u,v) \in E$ et tout $x \in \mathbb{S}$. Donc la r-fonction de chemin r_P, qui est obtenue par composition de plusieurs r-fonctions, vérifie $x \preceq_\oplus \mathsf{r}_P(x)$. □

Théorème 69 *Si l'opérateur de réseau ∇_G est composée, au niveau de chaque nœud, d'un* r*-opérateur n-aire idempotent défini sur $\mathbb{S}$ et basé sur le même* s*-opérateur $\oplus$, alors la ∇_G-`direct-association` se termine sur tout réseau, initialisée avec toute pvar de $\mathbb{S}^{|V|}$. De plus, si $\mathsf{r}_{P_{u\to v}}$ désigne la* r*-fonction de chemin du chemin élémentaire $P_{u\to v}$ de u à v, alors le résultat sur le nœud v est donné par*

$$b_{out}[v] = \bigoplus_{u \in \overline{\Gamma_G^-}(v)} \mathsf{r}_{P_{u\to v}}\left(b_{in}^0[u]\right)$$

Preuve. Il existe une date t à partir de laquelle tout nœud w a reçu toutes les valeurs initiales $b_{\text{in}}^0[v]$ de ses antécédents v dans le réseau G, à travers tous les chemins possibles. En effet, le réseau ou les calculs locaux ne sont pas stables sinon. Puisqu'aucune boucle n'est masquée dans le réseau, chaque nœud a gardé toutes les valeurs reçues dans son expression $(b_{\text{out}}[w])^t$. La r-expression de $(b_{\text{out}}[w])^t$ peut être transformée en s-expression, dont les r-fonctions sont entièrement développées, en appliquant la définition 60 page 92 des r-opérateurs n-aires (les r-fonctions sont des isomorphismes de $(\mathbb{S}, \oplus)$).

Ce faisant, la s-expression contient toutes les r-fonctions de chemins correspondant aux divers chemins reliant les antécédents de w à w lui-même. Si le réseau ne contient pas de circuit, alors ce nombre de chemins est fini, et l'expression est donc elle-même finie. Par contre, si le réseau présente un circuit C contenant les sommets v et w, alors le sommet w a, dans son s-expression, les valeurs $\mathsf{r}_{P_{v\to w}}(b_{\text{in}}^0[v])$, $\mathsf{r}_C(\mathsf{r}_{P_{v\to w}}(b_{\text{in}}^0[v]))$, $\mathsf{r}_C^2(\mathsf{r}_{P_{v\to w}}(b_{\text{in}}^0[v]))$, etc.

Mais d'après le lemme 68 page précédente, on a

$$\mathsf{r}_{P_{u\to v}}\left(b_{\text{in}}^0[v]\right) \preceq_\oplus \mathsf{r}_C\left(\mathsf{r}_{P_{u\to v}}(b_{\text{in}}^0[v])\right) \preceq_\oplus \cdots \preceq_\oplus \mathsf{r}_C^k\left(\mathsf{r}_{P_{u\to v}}(b_{\text{in}}^0[v])\right)$$

pour tout $k \in \mathbb{N}$. L'expression, que le circuit a rendu de taille quelconque non bornée, peut donc être simplifiée car $\mathsf{r}_{P_{u\to v}}\left(b_{\text{in}}^0[v]\right) \oplus \mathsf{r}_C^k\left(\mathsf{r}_{P_{u\to v}}(b_{\text{in}}^0[v])\right) = \mathsf{r}_{P_{u\to v}}\left(b_{\text{in}}^0[v]\right)$ pour tout $k \geq 1$. La s-expression $(b_{\text{out}}[w])^t$, une fois simplifiée, ne contient plus que des r-fonctions de chemins élémentaires[(h)], ce qui montre le théorème. □

5.3.4 Conclusion

Les r-opérateurs n-aires généralisent les r-opérateurs binaires. Ils assurent la terminaison sur un résultat déterministe et reconnaissable des `direct-association`. On peut remarquer que le résultat du théorème 69 généralise celui du théorème 59 page 90. En effet, si toutes les r-fonctions sont identiques (cas des r-opérateurs binaires), on a $\mathsf{r}_{P_{u\to v}} \equiv \mathsf{r}^{\mathrm{d}(u,v)}$. Les r-opérateurs n-aires ont d'intéressantes propriétés algorithmiques (*cf.* chapitre 7).

Cependant, pour obtenir une construction à partir des propriétés de r-associativité, r-commutativité et de r-idempotence (préalablement adaptées aux opérateurs n-aires), il faut introduire la propriété de r-isomorphisme. De plus, pour obtenir la propriété de réduction des expressions, il faut ajouter l'hypothèse de commutativité des r-fonctions du réseau. Malgré des hypothèses de définition qui peuvent donc apparaître faibles, les r-opérateurs n-aires limitent le type de calcul local qu'il est possible de considérer. Nous proposons une seconde généralisation dans la section suivante.

(h) On peut aussi formuler le théorème sans préciser que les chemins sont élémentaires dans la formule résultat : la valeur de l'expression sera la même. Cependant, pour montrer la terminaison de la `direct-association` par la méthode de la réduction des expressions, il faut s'assurer que l'expression peut toujours être ramenée à une expression de taille finie. Le fait que les chemins élémentaires sont suffisants pour exprimer le résultat conduit à une expression de taille finie car il y a un nombre fini de chemins élémentaires dans un graphe ayant un nombre fini d'arcs.

5.4 Les r-opérateurs généralisés

Les r-opérateurs binaires permettent de modifier les valeurs reçues à l'aide d'une r-fonction, avant de les intégrer dans le calcul effectué par un s-opérateur (*cf.* figure 62 page 91). Les r-opérateurs n-aires permettent d'utiliser différentes r-fonctions pour cela. Cependant ces r-fonctions sont toujours des applications de $\mathbb{S}$ dans $\mathbb{S}$. Nous proposons dans cette section d'utiliser des r-fonctions de plusieurs variables (figure 65), et ainsi de considérer de nouveaux opérateurs, que nous appellerons r-opérateurs généralisés. Ces opérateurs auront des applications algorithmiques importantes. Par exemple, l'opérateur *OuvRec* est utilisé en analyse d'images pour supprimer des défauts (morphologie mathématique, *cf.* § 7.6.3 page 148). C'est un r-opérateur généralisé (idempotent) défini comme suit :

$$\textit{OuvRec}\,(x_0,\ldots,x_8) = \mathsf{max}\,(x_0, \mathsf{min}\,(x_0,\ldots,x_8)) \tag{5.3}$$

Néanmoins l'étude de ces opérateurs impose un formalisme plus lourd et les résultats de la `direct-association` ne sont pas toujours reconnaissables. Nous finissons par une discussion sur les hypothèses nécessaires à la définition d'opérateurs assurant la terminaison sur un résultat déterministe et reconnaissable des `direct-associations`. Elle achèvera notre étude des opérateurs admettant cette propriété.

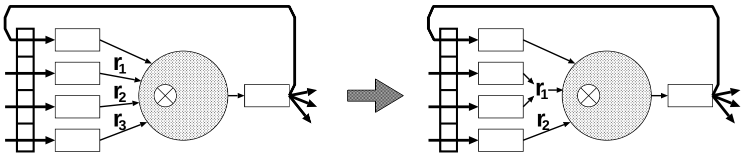


FIG. 65 – *Extension proposée par les r-opérateurs généralisés.*

5.4.1 Définitions

La définition ci-dessous englobe tout opérateur n-aire basé sur un s-opérateur qui puisse être utilisé par un nœud du réseau pour ses calculs locaux.

Définition 70 *L'opérateur $\diamond_u$ utilisé par le nœud u pour ses calculs locaux est une* rs*-fonction s'il s'agit d'une application de $\mathbb{S}^{(\delta_u^-+1)}$ dans $\mathbb{S}$ telle que*

(a) *il existe un* s*-opérateur $\oplus$ sur $\mathbb{S}$,*

(b) *il existe un entier $l > 0$ et $l+1$ entiers $0 < k_i \leq \delta_u^-$, $(0 \leq i \leq l)$,*

(c) *il existe l fonctions r_i de $\mathbb{S}^{k_i}$ dans $\mathbb{S}$, $(0 \leq i \leq l)$*

telle que pour tout $x_0,\ldots,x_{\delta_u^-} \in \mathbb{S}$, $\diamond_u$ puisse être écrit sous la forme :

$$\diamond_u\left(x_0,\ldots,x_{\delta_u^-}\right) \quad = \quad \mathsf{r}_0\left(x_0^0,\ldots,x_0^{k_0}\right) \oplus \mathsf{r}_1\left(x_1^0,\ldots,x_1^{k_1}\right) \oplus \cdots \oplus \mathsf{r}_l\left(x_l^0,\ldots,x_l^{k_l}\right)$$

où, pour tout p dans $\{0,\ldots,l\}$ et tout q dans $\{0,\ldots,k_p\}$, x_p^q est l'une des variables $x_0,\ldots,x_{\delta_u^-}$.

Par exemple, si le degré du nœud v est 3, et si $\mathbb{S} \equiv \mathbb{N}$, alors l'opérateur $\diamond_v$ suivant est une rs-fonction pour le nœud v : $\diamond_v(x_0,x_1,x_2,x_3) = \mathsf{min}(x_0 - 2\times x_1, (x_2+2)\times(x_3+2), 4\times(x_1+x_2+x_3))$.

Lorsque l'une des fonctions r_i de la définition ci-dessus est l'identité Id, et qu'elle prend en argument la première variable x_0, la rs-fonction ne modifie pas la valeur initiale du nœud sur laquelle elle est employée. On dit aussi qu'elle *préserve la première variable*. Dans la suite, toutes les fonctions r_i de la définition 70 sauf celle-ci (identité appliquée à x_0) sont appelées r*-fonctions*. L'opérateur $\diamond_v$ considéré dans l'exemple précédent ne préserve pas la valeur initiale. Par contre, l'opérateur $\diamond_w(x_0,x_1,x_2,x_3) = \mathsf{min}(x_0, (x_2+2)\times(x_3+2), 4\times(x_1+x_2+x_3))$ est une rs-fonction du nœud w qui préserve la première variable.

Définition 71 *Une* r*-fonction* $\mathsf{r} : \mathbb{S}^k \to \mathbb{S}$ *est un homomorphisme n-aire pour l'opérateur $\diamond$ si, pour tout $x_0, \ldots, x_{k-1}, y \in \mathbb{S}$ et pour tout i, $1 \le i \le k$, on a :*

$$\mathsf{r}\,(x_1, \ldots, x_i \diamond y, \ldots, x_k) = \mathsf{r}\,(x_1, \ldots, x_i, \ldots, x_k) \oplus \mathsf{r}\,(x_1, \ldots, y, \ldots, x_k)$$

Par exemple, les r-fonctions $\mathsf{r}_1 : (x, y) \mapsto (x + 2) \times (y + 2)$ et $\mathsf{r}_2 : (x, y, z) \mapsto 4 \times (x + y + z)$ de la rs-fonction $\Diamond_w$ sont des homomorphismes n-aires pour l'opérateur min sur $\mathbb{N}$.

Définition 72 *Un* r*-opérateur généralisé est une* rs*-fonction basée sur le* s*-opérateur $\oplus$ qui préserve la première variable et dont les* r*-fonctions sont des homomorphismes n-aires.*

Ainsi, la rs-fonction $\Diamond_w : (x_0, x_1, x_2, x_3) = \mathsf{min}(x_0, (x_2 + 2) \times (x_3 + 2), 4 \times (x_1 + x_2 + x_3))$ est un r-opérateur généralisé pour le nœud w de degré entrant $\delta_w^- = 3$. Par contre, la rs-fonction $\Diamond_v(x_0, x_1, x_2, x_3) = \mathsf{min}(x_0 - 2 \times x_1, (x_2 + 2) \times (x_3 + 2), 4 \times (x_1 + x_2 + x_3))$ n'en est pas un.

L'idempotence d'un r-opérateur binaire $\triangleleft$ basé sur le s-opérateur $\oplus$ se traduit par $x \preceq_\oplus \mathsf{r}(x)$. Nous généralisons donc cette propriété aux r-fonctions de plusieurs variables pour définir un r-opérateur généralisé.

Définition 73 *Une* r*-fonction* $\mathsf{r} : \mathbb{S}^k \to \mathbb{S}$ *est expansive si pour tout $x_0, \ldots, x_{k-1} \in \mathbb{S}$, on a :*

$$\forall i, 0 \le i < k, \quad x_i \preceq_\oplus r\,(x_0, \ldots, x_{k-1})$$

Par exemple, les r-fonctions r_1 et r_2 de la rs-fonction $\Diamond_w$ définie sur $\mathbb{N}$ sont expansives.

Définition 74 *Un* r*-opérateur généralisé est idempotent si toutes ses* r*-fonctions sont expansives.*

Le r-opérateur généralisé $\Diamond_w$ est donc idempotent. Puisqu'un r-opérateur généralisé idempotent préserve sa première variable et présente des r-fonctions expansives, seule la première variable est préservée. De plus, cette variable n'apparaît dans aucune r-fonction expansive, parce que cela rendrait inutile cette r-fonction dans la définition de l'opérateur. En effet, si r est cette r-fonction, on aurait $x_0 \oplus \mathsf{r}(\ldots, x_0, \ldots) = x_0$.

5.4.2 Application des r-opérateurs généralisés à la `direct-association`

Terminaison. Les r-fonctions des r-opérateurs n-aires que nous avons définis au § 5.3 page 92 permettaient d'associer une r-fonction à chaque arc du réseau, et ainsi de définir des r-fonctions de chemins (figure 64 page 93). Avec un r-opérateur généralisé, une r-fonction de chemin est moins simple à définir. On ne peut associer à un arc (u, v) qu'une fonction partiellement définie, dont la définition exacte nécessite les valeurs véhiculées sur les autres arcs incidents du nœud v. Nous l'appelons r*-fonction paramétrée*. Par exemple, si le nœud w de degré entrant $\delta_w^- = 3$ utilise le r-opérateur généralisé $\Diamond_w : (x_0, x_1, x_2, x_3) = \mathsf{min}(x_0, (x_2 + 2) \times (x_3 + 2), 4 \times (x_1 + x_2 + x_3))$, alors la r-fonction paramétrée r_2 de son deuxième arc entrant dépend des valeurs x_1 et x_3 reçues sur les autres arcs : $\mathsf{r}_2|_{x_1, x_3} : x \mapsto \mathsf{min}((x + 2) \times (x_3 + 2), 4 \times (x_1 + x + x_3))$. Plus généralement, on note la r-fonction paramétrée de l'arc (u, v) par $\mathsf{r}_v^u|_{\ldots}$. La proposition suivante montre que la croissance des variables dans les r-fonctions entraîne la croissance des variables sur les arcs, et réciproquement.

Proposition 75 *Soit v un nœud utilisant, pour ses calculs locaux, la* rs*-fonction $\Diamond_v$ basée sur le* s*-opérateur $\oplus$. Les deux propositions suivantes sont équivalentes :*

- ***(a)*** *chaque* r*-fonction de la* rs*-fonction $\Diamond_v$ est expansive,*
- ***(b)*** *pour chaque antécédent direct u de v, la* r*-fonction paramétrée $\mathsf{r}_v^u|_{\ldots}$ est expansive*[(i)].

Preuve. (a) $\Rightarrow$ (b) Soit $\mathsf{r}_v^u|_{\ldots}(x_u) = \bigoplus_{1 \le i \le \delta_v^-} \left\{ \mathsf{r}_i \left(x_i^1, \ldots, x_i^{k_i} \right), \exists j, 1 \le j \le k_i, x_i^j = x_u \right\}$ la r-fonction paramétrée d'un arc (u, v) incident à v, obtenue à partir des r-fonctions r_i de la rs-fonction $\Diamond_v$. Puisque x_u est une variable utilisée par toutes les r-fonctions r_i composant

(i) L'expansion d'une r-fonction paramétrée (basée sur le s-opérateur $\oplus$) $\mathsf{r}|_\alpha$ se traduit par $x \preceq_\oplus \mathsf{r}|\alpha$ pour tout $x \in \mathbb{S}$ et tout paramètre α.

$\mathsf{r}_v^u|_{\ldots}$, on a $x_u \preceq_\oplus \mathsf{r}_i\left(x_i^1, \ldots, x_u, \ldots, x_i^{k_i}\right)$ pour chacune de ces r-fonctions, par hypothèse. On a donc : $x_u \preceq_\oplus \bigoplus_{1\le i\le \delta_v^-}\left\{\mathsf{r}_i\left(x_i^1, \ldots, x_i^{k_i}\right), \exists j, 1 \le j \le k_i, x_i^j = x_u\right\}$, ce qui signifie que la r-fonction paramétrée $\mathsf{r}_v^u|_{\ldots}$ est expansive.

(b) $\Rightarrow$ **(a)** Soit x_u une variable d'une r-fonction r_i de la rs-fonction $\diamond_v$, telle que x_u est arrivée en v par le lien (u, v). Dans ce cas, la r-fonction r_i est utilisée dans la définition de la r-fonction paramétrée $\mathsf{r}_v^u|_{\ldots}$ correspondant à l'arc (u, v). Par hypothèse, on a $x_u \preceq_\oplus \mathsf{r}_v^u|_{\ldots}(x_u)$. On a donc $x_u \preceq_\oplus \mathsf{r}_i(\ldots, x_u, \ldots)$, ce qui montre la réciproque. □

On définit de même une r-fonction de chemin paramétrée $\mathsf{r}_P|_{\ldots}$, pour chaque chemin P dans le réseau. Le lemme 68 page 95 de croissance des r-fonctions de chemin reste valable pour ces fonctions paramétrées. Ainsi, dans un réseau associatif dont tous les nœuds utilisent un r-opérateur généralisé idempotent basé sur le même s-opérateur, les r-fonctions de chemin sont toutes expansives. Cette propriété permet d'obtenir le théorème suivant, dont la preuve est identique à celle du théorème 69 page 96, relatif à la terminaison des `direct-associations` utilisant des r-opérateurs n-aires.

Théorème 76 *Si l'opérateur de réseau ∇_G est composée, au niveau de chaque nœud, d'un r-opérateur généralisé idempotent défini sur $\mathbb{S}$ et basé sur le même s-opérateur $\oplus$, alors la ∇_G-`direct-association` se termine sur tout réseau, initialisée avec toute pvar de $\mathbb{S}^{|V|}$.*

Reconnaissance du résultat. Lorsque les r-fonctions de l'opérateur généralisé sont des fonctions de $\mathbb{S}$ dans $\mathbb{S}$, alors nous retrouvons le résultat des r-opérateurs n-aires. Mais les r-opérateurs généralisés restent une extension des n-aires, même dans ce cas, car les r-fonctions n'ont pas besoin d'être bijectives. En effet, nous n'avons pas utilisé cette propriété dans le raisonnement sur les chemins du graphe de la preuve du théorème 69 page 96 : seules les propriétés d'homomorphisme et d'expansion des r-fonctions étaient nécessaires. Si les conditions de définition des r-opérateurs n-aires ont permis d'étendre des propriétés intéressantes des r-opérateurs binaires, elles ont aussi limitées les opérateurs concernés. Les r-opérateurs généralisés lèvent cette restriction.

Par exemple, la rs-fonction $(x_0, x_1, x_2, x_3) \mapsto \mathsf{min}(x_0, 2\times x_1, 4\times x_2)$ n'est pas un r-opérateur n-aire sur $\mathbb{N}$, parce que ses r-fonctions ne sont pas bijectives[(j)] sur $\mathbb{N}$. Cependant, c'est un r-opérateur généralisé idempotent, dont les r-fonctions sont unaires, et qui conduit donc à la terminaison déterministe de la `direct-association`, sur le résultat reconnaissable donné par le théorème 69 page 96 ($b_{\text{out}}[v] = \bigoplus_{u\in\overline{\Gamma_G^-}(v)} \mathsf{r}_{P_{u\to v}}\left(b_{\text{in}}^0[u]\right)$).

Outre ces cas simples, basés sur des r-fonctions unaires, il est difficile d'exprimer le résultat d'un nœud sous forme d'une expression synthétique avec les r-opérateurs généralisés, contrairement aux r-opérateurs binaires ou n-aires. Cela est dû à la généralisation des r-fonctions, qui complexifient l'expression du résultat. On peut certes obtenir une formulation récursive de l'expression-résultat, mais elle n'est pas utilisable en pratique.

Malgré ce problème, les r-opérateurs généralisés ont des propriétés intéressantes, notamment en morphologie mathématique (*cf.* § 7.6.3 page 148). Dans [Robin, 1997], l'auteur étudie un circuit asynchrone pour le traitement d'images. À cette fin, il étudie la convergence de l'opérateur ∇ sur $\mathbb{N}$:

$$\textit{FerRec}\,(x_0, \ldots, x_{n-1}) \quad = \quad \mathsf{min}\,(x_0, \mathsf{max}\,(x_1, \ldots, x_{n-1})) \tag{5.4}$$

Cet opérateur est une rs-fonction basée sur le s-opérateur min. Il n'a qu'une seule r-fonction n-aire ($(n-1)$-aire) r, donnée par $\mathsf{r}(x_1, \ldots, x_{n-1}) = \mathsf{max}\,(x_1, \ldots, x_{n-1})$. ∇ préserve la valeur initiale et sa r-fonction est un homomorphisme expansif de $(\mathbb{N}, \mathsf{min})$. Il s'agit donc d'un r-opérateur généralisé. Robin montre que le calcul asynchrone de cet opérateur est équivalent au calcul synchrone. Mis à part ce type de preuves, valables pour un opérateur précis, nous ne disposons pas de solution générale

(j) Pour cet exemple simple, il suffit cependant d'étendre l'ensemble de définition de l'opérateur à $\mathbb{R}^+$.

pour résoudre ce type de problème[k], [l], [m].

5.4.3 Discussion sur les hypothèses

Aux § 5.2.3, 5.3.3 et 5.4.2, nous avons montré que les conditions définissant les r-opérateurs sont suffisantes pour assurer la terminaison déterministe des `direct-associations`, et la reconnaissance du résultat pour les r-opérateurs binaires et n-aires. Nous discutons maintenant des conditions nécessaires.

Idempotence. Tout d'abord, rappelons que la k-idempotence faible des opérateurs est nécessaire (proposition 3 page 31). Nous avons supposé l'idempotence (1-idempotence forte) des r-opérateurs, ce qui s'est traduit par la propriété d'expansion ($x \preceq_{\oplus} \mathsf{r}(\ldots, x, \ldots)$) des r-fonctions et des r-fonctions de chemin. Il est clair que l'hypothèse de la 1-idempotence est trop forte ; les mêmes raisonnements sur la réduction des expressions peuvent être menés en utilisant la k-idempotence faible : au lieu de ne tolérer qu'une seule occurrence de chaque variable x dans les expressions, on en accepte k_x, où k_x est la constante de la k-idempotence faible pour la variable x (*cf.* définition 2 page 31).

En outre, nous voulions trouver des opérateurs qui assurent la terminaison sur tous les réseaux, quelle que soit la pvar utilisée. En fait, des propriétés plus faibles sur les opérateurs peuvent être obtenues si on admet des conditions portant à la fois sur les opérateurs et sur les réseaux. Ainsi, il est inutile de supposer la propriété d'expansion des r-fonctions de chemins r_P sur les chemins P qui ne sont pas des circuits. En effet, cette croissance ne sert qu'à supprimer les occurrences multiples de $\mathsf{r}_P(\cdot)$ dans la s-expression, et il n'y a des occurrences multiples que si P est un circuit (*cf.* preuve du théorème 59). Notons aussi que la proposition 1 page 30 indique que sur un réseau sans circuit, la `direct-association` se termine avec tout opérateur (mais il faut encore s'assurer que le résultat est déterministe et utilisable).

Préservation de la première variable. La préservation de la première variable est une condition nécessaire. En effet, considérons le réseau de la figure 66. Si le s-opérateur $\oplus$ est le minimum sur $\mathbb{S} = \mathbb{Z} \cup \{+\infty\}$, et la r-fonction r est définie par $\mathsf{r} : x \mapsto x - 1$, alors le calcul ne se stabilise pas.

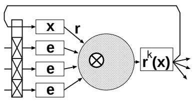


FIG. 66 – La préservation de la première variable est nécessaire.

Homomorphisme. La propriété d'homomorphisme n'est pas nécessaire. Elle est indispensable pour la réduction des expressions, mais il s'agit d'une contrainte liée à la méthode utilisée plutôt qu'une contrainte inhérente au problème à résoudre. La réduction des expressions nous a permis de montrer la terminaison des calculs globaux, et aussi de reconnaître le résultat pour les r-opérateurs binaires et n-aires. De même, l'étude des r-opérateurs généralisés nous a permis de constater que les r-fonctions n'ont pas besoin d'être bijectives. La propriété de bijection que nous avions supposée pour les r-opérateurs binaires et n-aires nous a uniquement servi à mieux caractériser ces opérateurs.

s-opérateur. Nous avons toujours supposé les opérateurs bâtis sur un s-opérateur. Il semble difficile de conclure sur la terminaison, le déterminisme et l'exploitation du résultat lorsque des opérateurs plus généraux sont considérés.

Le manque de coordination des calculs et des communications conduit à «oublier» des calculs intermédiaires, car ceux-ci ne se sont peut-être pas tous produits. Par exemple, un nœud peut avoir une vitesse de calcul très lente, et il n'a pas pu exploiter toutes les valeurs qu'il a reçues avant que celles-ci ne soient écrasées dans ses tampons entrants par de nouvelles valeurs. Cependant, l'usage

---

(k) Nous explicitons les liens entre les r-opérateurs et le calcul matriciel au § 5.5 page suivante ; cette technique ne s'applique pas aux opérateurs généralisés, et ne résout donc pas le problème.

(l) On a $\min(x_0, \max(x_1, \ldots, x_{n-1})) = \max(\min(x_1, x_0), \ldots, \min(x_{n-1}, x_0))$. Mais même en supposant que x_0 est fixé (ce qui permet d'obtenir des r-fonctions unaires), on n'obtient pas un r-opérateur (n-aire ou généralisé) car la première variable n'est pas préservée.

(m) Dans [Cochet-Terrasson et al., 1998], les auteurs étudient les fonctions composées des opérateurs min et max sur $\mathbb{R}^n$. Ces fonctions sont appelées fonctions topiques (en anglais, *topical functions*). Les auteurs étudient les problèmes de points fixes de telles fonctions. L'adaptation de leurs travaux à notre étude pourrait peut-être apporter une solution.

d'un s-opérateur assure que l'oubli de ces calculs ne perturbe pas le résultat de la `direct-association`. En effet, les calculs effectués après ces oublis conduisent à des valeurs plus petites (au sens du s-opérateur) que celles qu'auraient produites les calculs oubliés ; il est donc légitime de ne pas les prendre en compte (*cf.* § 6.1 page 103). En outre, les s-opérateurs et ses dérivés préviennent de toute oscillation dans le réseau. La figure 67 donne un exemple de réseau et d'opérateur qui peut conduire à des oscillations. Retenir une valeur plus petite que celles qui sont arrivées assurent la terminaison du calcul global, et, par là, l'absence de telles oscillations.

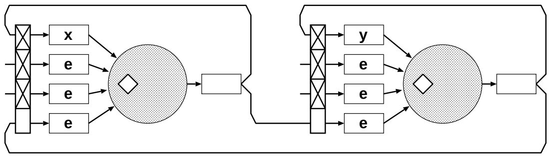


FIG. 67 – *Des oscillations peuvent apparaître si l'opérateur $\diamond$ choisit toujours l'opérande de droite sauf lorsque celui-ci est égal à e (i.e., $x \diamond y = y$ si $y \neq e$, $x \diamond y = x$ sinon).*

Outre la tolérance au manque de coordination du réseau, les s-opérateurs assurent des propriétés intéressantes de résistance aux pannes, que nous détaillons dans le chapitre suivant.

Conclusion. Bien que nous n'ayons pas obtenu les conditions nécessaires et suffisantes sur les opérateurs utilisables avec la `direct-association`, nous avons cependant cerné le problème et largement étendu la classe des s-opérateurs.

5.5 Conclusion

Résumé. Nous avons commencé ce chapitre en justifiant l'étude des aspects calculatoires des réseaux associatifs sur le plan microscopique du nœud. Nous avons ensuite défini le problème, à savoir trouver des opérateurs aux propriétés algorithmiques plus nombreuses que les s-opérateurs, et qui assurent la terminaison de la `direct-association` sur tout réseau, en donnant un résultat déterministe, et reconnaissable (*i.e.*, exploitable en programmation). Nous avons présenté une méthode basée sur la propriété de réduction des opérateurs pour résoudre ce problème ; nous avons montré que les s-opérateurs vérifient cette propriété de réduction des expressions.

Puis nous avons proposé des extensions des propriétés classiques d'associativité, de commutativité et d'idempotence, de manière à généraliser les s-opérateurs. Nous avons ainsi défini et étudié une algèbre déformée (r-algèbre). En particulier, les r-opérateurs idempotents réduisent les expressions, et assurent la terminaison du calcul global asynchrone de la `direct-association` (figure 68 page suivante). Les r-opérateurs ont des propriétés intéressantes, et de nombreuses applications algorithmiques (*cf.* chapitre 7). Sous certaines conditions supplémentaires, ils ont aussi d'intéressantes propriétés de résistance aux pannes (*cf.* chapitre 6).

Nous avons ensuite étendu les propriétés de ces opérateurs binaires aux opérateurs n-aires, puis généralisés. Ces extensions accroissent les applications algorithmiques des opérateurs, en exploitant mieux l'information codée par le sous-réseau lui-même. Ainsi, les s-opérateurs n'utilisent que l'information de la forte connexité du réseau pour construire leur résultat (*cf.* théorème 15 page 41). Ils permettent, par exemple, de calculer le maximum des antécédents. Les r-opérateurs binaires incluent la notion de distance (*cf.* théorème 59 page 90), qui est plus riche (*i.e.*, elle permet de déduire une information sur la forte connexité). Les r-opérateurs n-aires incluent une information relative aux chemins du réseau (*cf.* théorème 69 page 96), plus riche encore. Ils permettent, par exemple, de distinguer deux chemins différents de même longueur.

Les r-opérateurs généralisés incluent une information encore plus complexe, mais dont nous ne disposons pas sous une forme explicite, sauf pour certains cas particuliers, où une étude supplémentaire spécifique a été menée. Le problème de la reconnaissance du résultat des r-opérateurs généralisés n'est pas résolu dans le cas général. Enfin, nous avons fini par une discussion sur les hypothèses utilisées, ce qui cerne d'un peu plus près les conditions nécessaires à la résolution du problème traité dans ce chapitre.

Comme nous le signalions au § 5.1.3 page 80, l'étude aurait pu être menée avec un formalisme plus complet, à la manière de Kahn et Plotkin [Brookes, 1993] [Kahn et Plotkin, 1993]. Le comportement du réseau est décrit par une équation fonctionnelle pour chaque lien. On recherche alors un point fixe. Le rapport entre le comportement du réseau et ce point fixe est appelé *principe de Kahn*. La méthode que nous avons utilisée, plus limitée mais plus simple, était suffisante pour les besoins de notre étude ; elle a permi de définir et d'étudier les r-algèbres. Nous exploitons ces résultats au chapitre 7. Nous les comparons avec les travaux de Tel (convergence des vagues pour les s-opérateurs) au chapitre 6.

Connexions. Un r-opérateur peut être décrit avec les deux lois d'un demi-anneau idempotent (ce peut être le demi-anneau de fonctions $(\mathbb{H}, \oplus, \otimes)$ du § 4.1.7 page 60). Mais les besoins de l'algorithmique des réseaux associatifs, et de ses implémentations (Maille Associative, bibliothèque de calcul, ...), nécessitent de disposer d'un seul opérateur pour coder le calcul d'un nœud. De plus, il nous fallait ici composer avec l'asynchronisme, le déterminisme et la reconnaissance du résultat ; c'est pour cela que nous avons utilisé la réduction des expressions.

Un calcul global avec un r-opérateur binaire ou n-aire peut se mettre sous forme de matrice. Lorsqu'on connaît l'expression de la r-fonction (par exemple, $\mathsf{r}(x) = x + 1$, comme pour l'opérateur minc), alors on obtient une matrice telle que celle utilisée pour l'algèbre de chemins (*cf.* § 4.1.6 page 59). Mais lorsqu'on ne connaît pas son expression, la matrice est semblable à celle de l'algèbre de chemins généralisée (*cf.* § 4.1.7 page 60). Le problème aurait pu, dans une certaine mesure, être ramené à un problème matriciel, grâce aux itérations asynchrones (*cf.* 4.2.4 page 68). La proposition 25 page 70 reste valable avec un r-opérateur idempotent, en remarquant que cette hypothèse correspond à l'absence de circuit absorbant dans le graphe. Le raisonnement de la preuve du théorème 69 page 96 ressemble d'ailleurs à celui de la proposition 3 de [Gondran, 1975].

Mais comme nous l'expliquions en début de chapitre, l'étude des aspects calculatoires du réseau au niveau des nœuds ouvre des possibilités algorithmiques plus larges. En effet, de nombreux algorithmes sont décrits par le comportement d'un nœud, et non par le comportement global donné sous forme matricielle. C'est cet aspect «distribué» de la conception des algorithmes qui nous permet d'en obtenir de nouveaux, au chapitre 7. Par contre, un certain nombre de ces algorithmes conçus avec un r-opérateur pourront être transformés en problème matriciel d'algèbre de chemins (tel que l'arbre en profondeur ou la recherche du diamètre). L'intérêt de cette vision microscopique de l'algorithmique est qu'elle permet de penser l'algorithme de façon asynchrone, plutôt que de vouloir l'adapter à un support asynchrone après coup, comme c'est le cas avec les itérations asynchrones. Enfin, le dernier avantage de cette approche qu'il nous semble important de signaler, est qu'elle permet de raisonner avec les méthodes des systèmes distribués (même si nous proposons d'adapter ce raisonnement au calcul matriciel). Nous abordons cet aspect des choses dans le chapitre suivant.

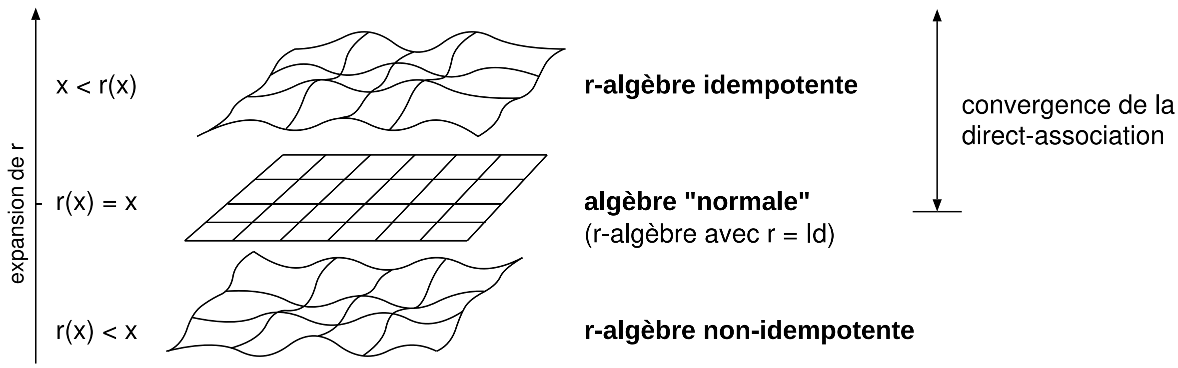


FIG. 68 – *Une* r*-algèbre est une algèbre déformée par une application* r*. La* `direct-association` *converge sur tout réseau lorsque* $x \preceq_{\oplus} \mathsf{r}(x)$.

Chapitre 6

Contrôle

Dans ce chapitre, nous étudions diverses propriétés du modèle, ayant essentiellement trait au contrôle du réseau : pertes de messages, pannes, détection de la terminaison du calcul, *etc.* La mise en œuvre d'un réseau associatif dans une bibliothèque de programmation ou dans un circuit intégré offre des outils pour contrôler le réseau. Mais nous nous situons ici dans un cadre général indépendamment de toute implémentation. Même si ce n'est pas sa vocation première, le réseau associatif pourrait être composé d'ordinateurs, reliés par un réseau de télécommunication. Les résultats généraux que nous détaillons dans ce chapitre sont applicables à toutes les mises en œuvres.

Nous commençons au § 6.1 par étudier les propriétés des liens, la complexité de la `direct-association` et la détection de sa terminaison. Nous montrons ensuite au § 6.2 page 109 que, sous certaines conditions, les r-opérateurs conduisent à des `direct-associations` auto-stabilisantes. Outre l'intérêt pour la résistance aux pannes de notre modèle, cela permet de mieux caractériser les r-opérateurs et leurs applications.

6.1 Propriétés du calcul de la `direct-association`

Dans cette section, nous étudions la `direct-association` sous l'angle des propriétés des liens, de la complexité et de la détection de sa terminaison.

6.1.1 Propriétés des liens

Nous précisons quelques propriétés concernant les liens de communication dans un réseau associatif. La nature de ces liens n'a pas été précisée dans la définition du modèle, laissant plusieurs possibilités à l'implémentation. Au § 4.3.3 page 74, nous avons fait le rapprochement entre les spécifications peu contraignantes de coordination calcul-communication du modèle des réseaux associatifs et le mode de communication par registres à lecture-écriture atomique. Nous examinons ici le cas des communications par passage de messages, plus réaliste lors de la programmation sur un réseau d'ordinateur. Ce type de communications concerne, par exemple, la mise en œuvre du modèle utilisant PVM[93] ou MPI[94][Calvin et Colombet, 1994].

Nous commençons par indiquer quelles sont les hypothèses de communication pour un opérateur quelconque (par exemple, l'addition sur un DAG). Puis nous expliquons que des hypothèses plus faibles peuvent être supposées lorsque l'opérateur est un s- ou un r-opérateur.

93. *Parallel Virtuel Machine.*
94. *Message Passing Interface.*

Hypothèses de communication. Nous supposons que toute donnée envoyée doit être reçue sans modification, et qu'il n'y a pas création spontanée (ni duplication) d'une donnée dans le réseau. De plus, sans autre information sur l'opérateur utilisé, il est nécessaire que les données envoyées sur un lien du réseau soient recopiées dans le tampon entrant correspondant dans le même ordre que celui de l'envoi. Il faut donc admettre la propriété FIFO pour les liens.

Perte de message. Cependant, puisque nous admettons que toute donnée envoyée est reçue au bout d'un temps fini (pas forcément borné, sauf si nous admettons l'hypothèse 11 page 37) et que les calculs ne sont pas subordonnés aux communications, la perte d'un message est admise. En effet, si la donnée x qui a été perdue sur le lien (u, v), est encore dans le tampon $b_{\text{out}}[u]$, elle finira par être retransmise, car toute donnée doit arriver en temps fini. Si elle ne l'est plus, elle a été écrasée dans le tampon $b_{\text{out}}[u]$ par une nouvelle donnée x'. Il n'y a donc plus moyen de recopier x dans $b_{\text{in}}^{\mathtt{i}(u)}[v]$, car cette donnée a disparu du réseau et du nœud qui l'avait produite. Néanmoins, le modèle autorise que la donnée du tampon $b_{\text{in}}^{\mathtt{i}(u)}[v]$ soit écrasée par une nouvelle valeur avant d'avoir été consommée par l'entité v dans un quelconque calcul local.

Duplication et s-opérateurs. Lorsque le calcul local de chaque nœud fournit une s-expression, on peut admettre la duplication (finie) des données sur les liens. En effet, la duplication d'une valeur passe inaperçue si le nœud v qui en fait la réception a masqué son tampon $b_{\text{in}}^{0}[v]$ (boucle en provenance de son tampon de sortie $b_{\text{out}}[v]$), car les calculs et les communications sont indépendants, et le nœud peut effectuer plusieurs calculs locaux sans qu'il y ait eu de communication entre temps. La valeur dupliquée passe également inaperçue lorsque la boucle n'est pas masquée car le double est intégré dans la s-expression de v, puis supprimé grâce à l'idempotence[(a)].

Désordonnancement et s-opérateurs. Si aucune boucle (u, u) du réseau n'est masquée, et que le calcul local de chaque nœud fournit une s-expression, alors le désordonnancement des messages ne modifie pas le résultat de la $\oplus$-`direct-association`. En effet, dans ce cas toute donnée x' produite après la donnée x sur le nœud u est au plus égale à x, d'après le théorème 14 page 40, en remarquant que le calcul qui a produit x' inclut la donnée x. On a donc $x \oplus x' = x'$.

Supposons maintenant que la donnée x soit reçue en premier par le nœud v, descendant direct de u dans le réseau. Quand x' arrive en v, une nouvelle expression est calculée par le nœud v. Cette expression inclut soit x' tout seul (x a été écrasé avant d'être consommé par v, ou perdu en cours de route), soit à la fois x' et x : x' a été lu dans le tampon $b_{\text{in}}^{\mathtt{i}(u)}[v]$, et x était inclu dans la s-expression stockée dans $b_{\text{out}}[v]$ et aussi dans $b_{\text{in}}^{0}[v]$ (boucle non masquée). Puisque $x \oplus x' = x'$, on peut supprimer x de l'expression-résultat sur v, obtenue lorsqu'il n'y a pas eu de déséquencement des données envoyées sur le lien (u, v) (figure 69 gauche).

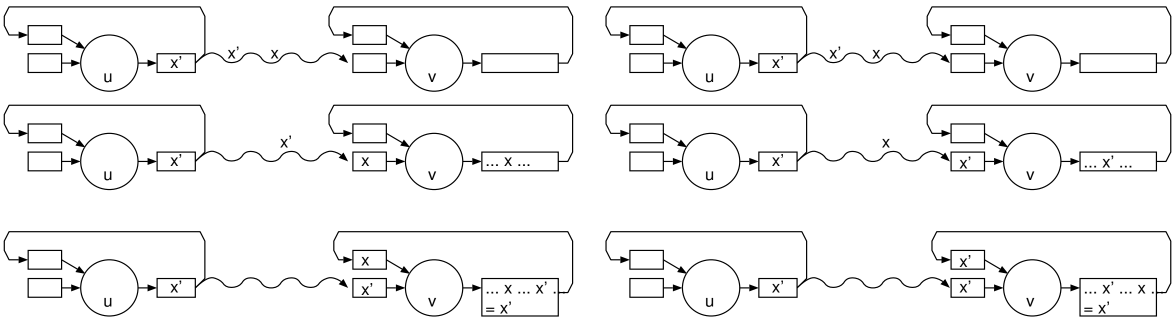


FIG. 69 – *Dès que x' arrive en v, l'expression de $b_{out}[v]$ peut être écrite sans x (à gauche, x arrive avant x', à droite x' arrive avant x).*

Supposons maintenant que x' soit reçu en premier par v, et que x arrive par la suite. De la même façon que précédemment, l'expression de $b_{\text{out}}[v]$ contiendra à la fois x et x', et elle pourra donc être écrite sans x (figure 69 droite). Ainsi, le désordonnancement de deux valeurs x et x' est sans conséquence dans le cas d'un s-opérateur appliqué sur un réseau dont les boucles ne sont pas masquées.

(a) L'absence d'influence de la duplication des données rend ainsi inutile les liens multiples entre deux nœuds du réseau pour les s- et les r-opérateurs.

Conclusion. Avec un opérateur quelconque, il est nécessaire de supposer qu'il n'y a ni modification, ni duplication, ni désordonnancement des valeurs envoyées. Mais l'hypothèse de non coordination des calculs et des communications permet d'admettre la perte de messages (sous réserve que son remplaçant dans le tampon à l'origine du lien fautif, arrive en temps fini dans le tampon situé à l'autre extrémité du lien).

En outre, lorsque le nœud calcule une s-expression, la duplication des données est autorisée. Enfin, s'il n'a pas masqué sa boucle, alors il supporte le désordonnancement des valeurs reçues. Ces propriétés sont aussi vérifiées avec les r-opérateurs car les nœuds calculent encore des s-expressions.

En terme de mise en œuvre avec une bibliothèque de programmation à passage de messages, cela signifie que les tampons d'arrivée peuvent avoir une contenance d'un seul message, et que les envois peuvent être non bloquants, dans le cas des s- et des r-opérateurs.

6.1.2 Complexité en temps.

Hypothèses. Pour évaluer la complexité en temps d'une `direct-association`, nous admettons l'hypothèse du délai borné (hypothèse 11 page 37) sans laquelle aucune évaluation en temps n'est possible. Toute recopie d'un message d'une extrémité à une autre d'un lien nécessite donc un temps borné par τ_r. Nous supposons aussi que le temps de production d'un résultat par une entité calculante est borné par τ_c.

Pour étudier le temps de stabilisation d'une `direct-association`, nous distinguons deux phases : le *temps de construction* et le *temps de diffusion*. Le temps de construction est le délai nécessaire pour que chaque résultat de chaque nœud apparaisse quelque part dans le réseau. Le temps de diffusion est le délai nécessaire à l'acheminement de tous les résultats sur leur nœud respectif. Le temps de stabilisation d'une `direct-association` est la somme du temps de construction et du temps de diffusion.

Temps de diffusion. Le temps de diffusion bénéficie directement des propriétés de réseau calculant du modèle. En effet, le mélange des calculs et des communications donne un temps de diffusion faible. La proposition suivante donne le temps de diffusion sur un réseau dans le pire cas.

Proposition 77 *Le temps de diffusion d'un calcul global déterministe sur le réseau G est borné par* $\mathrm{Diam}\,(G) \times (\tau_c + \tau_r)$.

Preuve. Si une seule donnée devait être diffusée à tout le réseau, la proposition serait vérifiée. Elle le serait encore si toutes les données pouvaient rejoindre leur destination sans se gêner sur les liens de communication.

On constate qu'aucune gêne n'apparaît en raison de l'absence de conflit sur les liens de communication. Supposons que la donnée $d(u)$, qui est le résultat final de u, soit produite sur le nœud u'. De même, supposons que la donnée $d(v)$, qui est le résultat final de v, soit produite sur le nœud v' (figure 70). Supposons aussi que les chemins suivis par $d(u)$ et par $d(v)$ pour rejoindre u et v partagent un segment commun, qui débute sur le nœud w' du réseau.

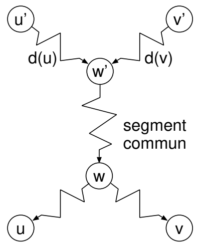


FIG. 70 – Absence de conflit sur un lien de communication.

Puisque le calcul global est déterministe, le même résultat est obtenu si $d(u)$ arrive la première en w' et traverse en premier le segment commun, ou si c'est $d(v)$ qui est la première en w' et qui traverse le segment avant $d(u)$.

Puisque $d(u)$ est le résultat final de u, il ne sera pas modifié durant sa traversée du segment commun. Tous les nœuds du segment vont donc réaliser un calcul local dont le résultat est $d(u)$. De même, $d(v)$ ne sera pas modifiée durant sa traversée du segment commun, et tous les nœuds effectueront à son passage un calcul local dont le résultat sera $d(v)$.

Maintenant supposons que $d(u)$ et $d(v)$ arrivent en même temps sur le nœud w'. Puisque le calcul global est déterministe, cette configuration doit aboutir au même résultat que les autres, et, en particulier, les constations précédentes sont encore valables. Mais alors le résultat du calcul local sur le nœud w' est à la fois $d(u)$ et $d(v)$. Donc $d(u) = d(v)$ et les données ne se gênent pas sur le segment commun. □

Cette preuve utilise le fait que le résultat final de u et de v est basé sur le résultat final du nœud w, qui est construit à l'aide des données reçues, et en particulier avec $d(u)$ et $d(v)$. Le mélange des calculs et des communications assure donc qu'il n'y a pas de conflit dans le réseau.

Temps de construction. Le temps de construction ne peut pas être évalué dans le cas général car il est intrinsèquement lié à l'opérateur utilisé. Considérons, par exemple, l'opérateur $(x, y) \mapsto \mathsf{min}(a, x + 1, y + 1)$ avec $a > 0$, appliqué sur un réseau en anneau, initialisé à 0 partout (figure 71). Le calcul global est déterministe et se termine en temps fini. Le résultat est facilement reconnu : tous

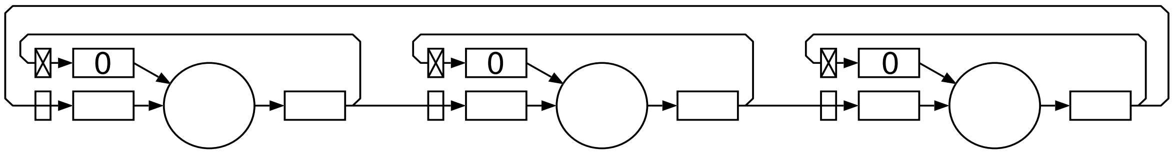


FIG. 71 – *Le temps de construction est intimement lié à l'opérateur utilisé : exemple de l'opérateur* $(x, y) \mapsto \mathsf{min}(a, x + 1, y + 1)$, $a > 0$.

les nœuds vont finir avec la valeur a. Le temps de construction du résultat de chaque nœud, c'est-à-dire le temps d'apparition de la valeur a est : $a \times \mathrm{Diam}\,(G) \times (\tau_r + \tau_c)$. Il est donc lié à l'opérateur lui-même. Par exemple, si a vaut 100, il faudra attendre 100 «tours» pour que cette valeur apparaisse.

Complexité en temps pour un s-opérateur. Lorsque l'ordre $\preceq_\oplus$ induit par le s-opérateur $\oplus$ est un ordre total, le résultat de chaque nœud est l'une des valeurs de ses antécédents ou sa propre valeur. Dans ce cas, le temps de construction est immédiat. La complexité en temps est donc égale au temps de diffusion, qui est $\mathrm{Diam}\,(G) \times (\tau_r + \tau_c)$ (proposition 77 page précédente).

Lorsque l'ordre induit n'est que partiel, l'analyse est différente mais conduit à la même complexité. En effet, dans ce cas, de nouvelles valeurs peuvent apparaître dans le réseau (car $x \oplus y$ peut être différent de x et de y). Le résultat d'un nœud peut donc être construit sur le nœud lui-même. Dans le pire des cas, il faudra que toutes les données soient parvenues sur le nœud pour que celui-ci puisse construire son résultat. Mais si le temps de construction est au pire $\mathrm{Diam}\,(G) \times (\tau_r + \tau_c)$, le temps de diffusion est nul. Ainsi, suivant les nœuds, la proportion du temps passé dans la diffusion et dans la construction peut varier, contrairement à ce qui se passe quand l'ordre est total. Mais il restera toujours borné par $\mathrm{Diam}\,(G) \times (\tau_r + \tau_c)$, car, au bout de ce délai, toutes les données ou leurs combinaisons seront parvenues sur le nœud, qui aura son résultat.

Complexité en temps pour un r-opérateur. Le résultat d'un nœud est, en règle générale, construit sur le nœud lui-même[(b)] et le temps de diffusion est alors nul. Le temps de construction est plus long que celui nécessaire à un s-opérateur. En effet, si on peut considérer les plus courts chemins lors de la propagation d'une donnée pendant une $\oplus$-`direct-association` (où $\oplus$ est un s-opérateur), il n'en est pas de même avec un r-opérateur : le chemin ayant la r-fonction de chemin qui augmente le moins la valeur considérée n'est pas forcément le plus court. La complexité maximale en temps est donc liée à la longueur du plus long chemin entre toute paire de sommets.

Signalons cependant qu'en pratique, la complexité moyenne est meilleure que cette borne supérieure. Ainsi, dans le cas d'une minc-`direct-association`, la complexité en temps est liée à la plus grande distance séparant un nœud détenant une valeur minimale d'un autre (hauteur de la forêt couvrante de poids minimum et non longueur du plus long chemin, *cf.* § 7.4.1 page 138). L'opérateur minc est défini par $(x, y) \mapsto \mathsf{min}(x, y + 1)$; il admet de nombreuses applications algorithmiques (*cf.* chapitre 7).

(b) Ce n'est pas toujours le cas ; il suffit de considérer la diffusion de $+\infty$ avec le r-opérateur minc.

6.1.3 Autres complexités

Complexité en messages. La complexité en messages est une mesure classique en systèmes distribués. Elle consiste à calculer le nombre de messages émis par le système pour exécuter l'algorithme, ici la `direct-association`. Si cette mesure peut trouver des applications dans l'évaluation d'un protocole de contrôle qui s'ajouterait au trafic du réseau, elle ne semble cependant pas adaptée à notre travail. En effet, cette complexité ne prend pas en compte les déséquilibres entre un nœud qui émettrait beaucoup de messages et un autre qui en émettrait peu. La topologie n'est pas non plus prise en compte, ni le routage du réseau (s'il y en a un). Enfin, lorsque le réseau est dédié, comme par exemple dans la Maille Associative, la limitation du nombre de messages n'a pas de raison d'être. Les mises en œuvres du modèle cherchent généralement à améliorer la vitesse de convergence, parfois au détriment de la complexité en message.

L'évaluation du nombre de messages émis est délicate en ce sens qu'elle est liée au temps de convergence du calcul, et que celui-ci ne peut être déterminé dans le cas général (*cf.* figure 71 page ci-contre).

Complexité en place mémoire. La complexité en place mémoire mesure la quantité de mémoire nécessaire à l'exécution de l'algorithme. Pour la `direct-association` cette quantité est directement liée à la pvar mise en jeu. Le nombre de tampons est quant à lui fixé ; il est donné par le degré de chaque nœud dans le réseau. Notons aussi, que certains opérateurs nécessitent l'utilisation d'une pvar-paramètre, telle que celle utilisée pour stocker les poids des arcs avec l'opérateur $\mathtt{minc}_w$ (*cf.* § 7.4.1 page 138).

6.1.4 Détection de la terminaison

La `direct-association` est un calcul global asynchrone sur un réseau d'entités calculantes et communicantes. Pour enchaîner plusieurs primitives de calcul dans un programme, il est nécessaire de déterminer la fin du calcul distribué sur le réseau. Après un rapide tour d'horizon des mécanismes utilisés pour détecter la terminaison, nous discutons de la solution qui nous semble la plus appropriée pour une utilisation de la primitive en calcul intensif. Dans un second temps, nous proposons un rapprochement entre la `direct-association` et les *vagues*, afin d'obtenir une version distribuée du calcul de la `direct-association` qui inclut sa propre détection de la terminaison.

Mécanismes de détection.* Le problème de la détection de la terminaison a été étudié en systèmes distribués et en parallélisme, où, dans les deux cas, un calcul est réparti sur des processeurs, qui doivent donc s'entendre sur la fin de celui-ci. Les hypothèses admises pour résoudre ce problème en calcul parallèle sont cependant plus fortes qu'en systèmes distribués. En calcul parallèle, le problème de la terminaison se pose uniquement si le calcul est asynchrone. C'est le cas dans les PRAM programmées par passage de messages, où la détection de la terminaison est basée sur la notion de synchronisation : lorsque tous les processeurs ont fini le présent calcul, alors chacun peut entamer le suivant. La synchronisation peut être mise en œuvre par un processeur «central», des sémaphores logiciels, ou un mécanisme matériel [Dinning, 1989] [Hoare et al., 1996].

Le problème de la détection de la terminaison est d'une tout autre difficulté en systèmes distribués. Ainsi, dans [Hazari et Zedan, 1987], les auteurs proposent d'associer un nœud de contrôle à chaque entité calculante. Ces nœuds sont reliés par un anneau unidirectionnel, et émettent un compteur si leur entité a terminé son travail. Le compteur est incrémenté à chaque passage sur un nœud de contrôle dont l'entité calculante a terminé, ou supprimé si l'entité n'a pas fini. Lorsque le compteur a atteint le nombre d'entités, le calcul est fini. Mais cette solution n'est pas satisfaisante car une entité ayant terminé son travail peut être réactivée par l'arrivée d'un message [Tel et Van Leeuwen, 1987]. Dans [Arora et al., 1986], les auteurs proposent un algorithme similaire, mais le message circulant sur l'anneau contient l'identifiant de l'émetteur. De plus, il n'est envoyé que si l'entité calculante et toutes ses voisines ont terminé. Un nœud recevant son message sur l'anneau sait que toutes les autres entités ont fini. Mais là encore, une entité qui n'a pas terminé pourrait réactiver une entité qui a fini, en court-circuitant les messages du circuit hamiltonien grâce à une corde [Tan et al., 1986]. Pour utiliser un anneau hamiltonien[(c)], il faut un jeton coloré [Beauquier et Bérard, 1991], ou un jeton en forme de vecteur d'autant de composantes qu'il y a de nœuds dans le réseau [Mattern, 1987]. Un tel vecteur est aussi maintenu par chaque nœud. Ces vecteurs comptent le nombre de messages en transit ; à chaque fois que le jeton arrive sur un nœud, on lui ajoute le vecteur du nœud. Lorsque le jeton est un vecteur vide, le calcul est terminé.

Mécanisme pour la `direct-association`. La mise en œuvre de la `direct-association` dans un but de calcul intensif nécessite un mécanisme de détection de la terminaison efficace. Il semble qu'une solution centralisée soit la plus appropriée. La Maille Associative dispose d'un arbre de détecteur de stabilité locale, dont la racine est le contrôleur de la machine (*cf.* § 1.2.2 page 13 et figure 18 page 15). Dans ANET, un processus maître distribue les «morceaux» du calcul parallèle. Il contrôle donc la terminaison du calcul (*cf.* chapitre 8).

Nous proposons maintenant d'inclure la détection de la terminaison de la `direct-association` dans son algorithme. Cette étude a été présentée dans [Ducourthial, 1998a]. Elle indique que la `direct-association` peut être adaptée dans un contexte distribué. Nous commençons par comparer la `direct-association` aux algorithmes en *vagues*[95].

95. En anglais, *wave.*

Vagues.* Une vague est un algorithme distribué, lancé par un (ou plusieurs) nœud(s) du réseau, qu'on appelle *initiateur*[96]. Après une initialisation interne, ces nœuds envoient des messages à leurs voisins. À la réception d'un message, ceux-ci effectuent un calcul local, et émettent à leur tour des messages (après une initialisation interne lors de la réception du premier message). Cette cascade de calculs locaux définit une *demi-vague*. Si la demi-vague se réfléchit depuis les dernières entités atteintes jusqu'aux initiateurs, on parle de *vague* [Raynal et Hélary, 1988].

96. En anglais, *initiator* ou *starter.*

Les vagues et demi-vagues ont été largement étudiées et utilisées en algorithmique distribuée, et ce, sous différentes appellations et variantes. Dans [Dijkstra et Scholten, 1980], les auteurs proposent une technique générale de détection de terminaison pour ces «calculs diffus»[97]. Dans [Chang, 1982], l'idée d'un retour vers l'instigateur (unique) est proposée pour s'assurer de la terminaison de la vague, et la technique est appelée algorithme à écho[98] . Dans [Segall, 1983], l'auteur étudie les protocoles PI[99] (demi-vague) et PIF[100] (vague), dans les réseaux non orientés, et l'étude est étendue aux réseaux orientés dans [Tel, 1987]. Diverses applications sont, en outre, développées. Dans [Raynal et Hélary, 1988], les auteurs étudient les algorithmes en vague sur diverses topologies, et utilisent la topologie (anneau, arbre) pour détecter la fin de la vague.

97. En anglais, *diffusing computations.*

98. En anglais, *echo algorithm.*

99. *Propagation of Information.*

100. *Propagation of Information with Feedback.*

Dans [Tel, 1994], l'auteur propose un formalisme plus général pour décrire les vagues (appelées algorithmes totaux[101] dans [Tel, 1991]) : une vague est un algorithme qui se termine et qui contient au moins un événement «décision» sur un nœud, qui survient après qu'un événement antérieur[(d)] ait eu lieu sur chacun des autres nœuds. Il montre ensuite que les PIFs sont des vagues ainsi définies, et que, réciproquement, toute PIF peut être utilisée pour calculer une vague. Il utilise aussi ces vagues pour calculer l'*infimum* d'une variable d'état des nœuds.

101. En anglais, *total algorithms.*

Un infimum $\oplus$ est un s-opérateur, et la vague calculant cet infimum ressemble à une $\oplus$-`direct-association`. Nous avons présenté cette comparaison dans [Ducourthial, 1998d]. Une demi-vague utilisant un r-opérateur permet, par exemple, de calculer une distance, ou bien d'obtenir un arbre en profondeur, etc. (*cf.* § 7.4 page 137).

Détection de la terminaison de la demi-vague utilisant un r-opérateur. Une demi-vague basée sur un r-opérateur ne peut pas s'exécuter sur une topologie particulière (telle qu'un anneau ou une arborescence couvrante du réseau), ce qui aurait permis de détecter sa terminaison. En effet, un r-opérateur inclut dans son résultat une information sur le réseau plus riche que le s-opérateur (distances

(c) Nous nous limitons ici aux mécanismes utilisant un circuit hamiltonien du réseau.

(d) Il s'agit de l'antériorité par rapport à la causalité des événements du système : cette relation est donné par l'enchaînement temporel sur chaque nœud (temps local au nœud) ; l'émission d'un message est un événement qui précède sa réception ; on obtient alors une relation de causalité entre les événements des nœuds émetteurs et récepteurs.

ou chemins au lieu de connexité), et le résultat est différent lorsque la primitive est exécutée sur un sous-réseau.

Nous proposons donc une modification de cette demi-vague pour qu'elle inclut une détection de sa terminaison, dans le cas où il n'y a qu'un seul initiateur et où l'opérateur est un r-opérateur *strictement idempotent* (cela exclut les s-opérateurs) :

Définition 78 *Si, pour tout $x \in \mathbb{S}$, le* r*-opérateur binaire $\triangleleft$, construit avec le $\oplus$-opérateur $\oplus$ et la* r*-fonction* r*, vérifie $x \prec_\oplus \mathsf{r}(x)$, où le relation $\prec_\oplus$ est définie par $x \prec_\oplus y \equiv (x \oplus y = x \text{ et } x \neq y)$, alors il est strictement idempotent.*

On suppose que des messages de contrôle peuvent circuler en sens inverse sur les arcs de G, et qu'une entité peut lire les valeurs des registres de sortie de ses voisines (antécédentes et descendantes). Une entité v du réseau est dite *cohérente à la date t* si elle vérifie[(e)] pour toute antécédente directe u, $b^t_{\text{out}}[v] \preceq_\oplus \mathsf{r}\left(b^t_{\text{out}}[u]\right) \neq e_\oplus$ et pour toute descendante directe w, $b^t_{\text{out}}[w] \preceq_\oplus \mathsf{r}\left(b^t_{\text{out}}[v]\right) \neq e_\oplus$. On appelle D^t le graphe défini à la date t par les arcs (u, v) de G tels que $b^t_{\text{out}}[u] = \mathsf{r}\left(b^t_{\text{out}}[v]\right) \neq e_\oplus$. D^t est, pour toute date t, un DAG, qui évolue en fonction des mises à jour de la vague.

La détection de la terminaison est faite ainsi : toute entité du DAG envoie le message de contrôle "je suis cohérente" à ses antécédentes directes si et seulement si elle est cohérente et elle a reçu un tel message de toutes ses descendantes directes dans le DAG. La demi-vague est achevée si et seulement si l'initiateur a reçu le message "je suis cohérente"de toutes ses filles.

En effet, si la demi-vague est terminée, alors toutes les entités du DAG (qui couvre dans ce cas tout le réseau) sont cohérentes. Pour la réciproque, constatons d'abord qu'une feuille du DAG ne peut être cohérente que lorsque le DAG ne peut plus s'étendre dans le réseau à partir d'elle, et que si une feuille n'est pas cohérente, alors l'initiateur (la racine du DAG) ne peut pas l'être. Considérons donc un DAG couvrant, et supposons la demi-vague non terminée. Montrons que dans ce cas l'initiateur n'est pas cohérent. Si la demi-vague n'est pas terminée, il existe un arc (u, v) dans le réseau tel que v n'est pas cohérente, et la demi-vague doit mettre à jour v et toutes ses descendantes. Si v se met à jour, elle ne devient cohérente immédiatement que si elle n'a pas de descendante. Sinon, elle devra attendre que ses descendantes directes soient elles-mêmes cohérentes (elles ne l'étaient plus suite à la mise à jour de v), qui elles-mêmes attendront après leurs descendantes directes, et ainsi de suite. Autrement dit, v ne sera cohérente que lorsque toutes ses descendantes dans le DAG le seront.

6.2 Propriétés auto-stabilisantes

La propriété auto-stabilisante d'un algorithme assure que celui-ci retrouvera «automatiquement» un comportement normal, après une panne temporaire capable de modifier l'ensemble des mémoires du système, dès que celle-ci aura disparu (*cf.* § 4.3.2 page 73). Nous montrons dans cette section que, sous certaines conditions, la $\triangleleft$-`direct-association` est auto-stabilisante sur toute topologie lorsque $\triangleleft$ est un r-opérateur, alors que ce n'est pas le cas lorsqu'il s'agit d'un s-opérateur (sauf dans certains cas particuliers, *e.g.*, DAGs). Nous commençons par présenter les hypothèses que nous utilisons. Ensuite, nous montrons que ces hypothèses sont nécessaires et suffisantes.

6.2.1 Hypothèses

Algorithme. Nous utilisons l'algorithme 5 de la `direct-association` donné au § 4.3.3 page 74. Nous supposons que l'algorithme ne prend jamais fin, c'est-à-dire que, bien qu'il y ait convergence, la `direct-association` n'est pas arrêtée (algorithme silencieux, *cf.* § 4.3.1 page 71). Ainsi,

(e) Les dates correspondent au temps global ; elles sont données par un observateur extérieur ayant une vue d'ensemble du système. La date est donné par l'exposant t dans $b^t_{\text{out}}[v]$.

dès qu'une faute transitoire surgit, l'algorithme redonne le bon résultat après un délai fini. Les algorithmes concernés sont ceux qui ne sont constitués que d'une seule `direct-association`. C'est le cas, par exemple, de la forêt des plus courts chemins (*cf.* § 7.4.1 page 138), de la forêt des fiabilités maximales (*cf.* § 7.4.2 page 139) ou de l'arbre en profondeur (*cf.* § 7.4.4 page 140).

Nous n'utilisons pas l'hypothèse de l'index (hypothèse 10 page 36), mais nous supposons que la pvar initiale, qui est stockée dans les registres b^0_{in} au début de la `direct-association` n'est pas modifiable. Il s'agit, par exemple, d'une valeur stockée en ROM. Bien qu'elle puisse apparaître restrictive, cette hypothèse reste faible. En effet, dans la plupart des algorithmes concernés, il nous suffirait de disposer d'un seul bit sûr, indiquant si le nœud est ou non une source, et c'est cette information que nous noterons dans $b^0_{\text{in}}[v]$, sur chaque nœud v. Ce type d'hypothèse est couramment admis, puisque nous ne disposons pas des identifiants, et que sans elle il n'y a pas de solution[(f)].

Puisque les registres b^0_{in} ne sont pas modifiables, la boucle (u, u) de chaque nœud u n'est pas nécessaire. De même, on suppose qu'il n'y a pas d'arc masqué. Les algorithmes auto-stabilisants que nous obtenons contiennent une seule `direct-association`. L'aspect dynamique du réseau est donc ici inutile.

Opérateur. Nous supposons que l'algorithme de la `direct-association` est instancié par un r-opérateur n-aire ∇, bien qu'un raisonnement similaire — mais au formalisme plus lourd — pourrait être mené avec un r-opérateur généralisé. Le calcul local de chaque nœud est donc donné par l'équation 3.2 page 37. Notons qu'en réalité cet opérateur est différent d'un nœud à un autre, puisqu'il dépend du degré entrant de chaque nœud, et de ses r-fonctions.

Nous restreignons notre étude au cas où ce r-opérateur est construit avec un s-opérateur $\oplus$ définissant un ordre total. Dans ce cas, pour tout ensemble de données $\{x_0, \ldots, x_k\}$, le minimum (au sens de $\prec_\oplus$) $\oplus\{x_0, \ldots, x_k\}$ est l'un des $x_i \in \{x_0, \ldots, x_k\}$, ce qui n'est pas forcément le cas avec un ordre partiel (un exemple d'ordre partiel est donné au § 7.4.5 page 142).

Nous supposons de plus que les r-opérateurs sont strictement idempotents.

Cette propriété est étendue aux opérateurs n-aires de la même façon que l'idempotence l'a été, en considérant chaque r-fonction du r-opérateur n-aire. On appelle *variable* r*-augmentée* une variable x modifiée par une r-fonction (éventuellement de chemin) : $\mathsf{r}(x)$ est la variable x r-augmentée.

On note $\mathcal{L}(b)$ la valeur légitime d'un registre b. D'après le théorème 69 page 96, on a $\mathcal{L}\left(b_{\text{out}}[v]\right) = \oplus\left\{\mathsf{r}_P(b^0_{\text{in}}[v]), u \in \Gamma^{-1}_G(v), P \text{ chemin élémentaire de } u \text{ à } v\right\}$. De même, un registre entrant $b^{\mathbf{i}(u)}_{\text{in}}[v]$ est légitime si $b^{\mathbf{i}(u)}_{\text{in}}[v] = \mathcal{L}\left(b_{\text{out}}[u]\right)$. Puisque le premier registre $b^0_{\text{in}}[v]$ n'est pas modifiable, il est toujours légitime. Un nœud v est *légitime* si son registre de sortie $b_{\text{out}}[v]$ est légitime. Finalement, une *configuration légitime* est telle que tous les nœuds du réseau sont légitimes.

Topologie. Les travaux en auto-stabilisation sur des réseaux généraux ont essentiellement concerné des réseaux bidirectionnels [Dolev et al., 1993] [Afek et al., 1990], ou des réseaux orientés (unidirectionnels) mais fortement connexes [Alstein et al., 1994]. Cela permet en général de construire une topologie virtuelle adéquate, telle qu'un arbre ou un anneau, sur lequel l'algorithme auto-stabilisant peut s'exécuter [Alima et al., 1998]. Ces restrictions sont raisonnables lorsque l'algorithme doit avoir un comportement dynamique. Par exemple pour traverser un graphe, un jeton est toujours en mouvement (ses déplacements sont dictés par l'algorithme), et il doit passer infiniment souvent sur chaque nœud. Dans [Bourgon et al., 1995], les algorithmes silencieux ont été abordés sur des réseaux qui n'étaient pas fortement connexes, mais qui n'avaient pas de circuit (DAGs).

Nous montrons ici que si la `direct-association` est utilisée avec un r-opérateur n-aire strictement idempotent et construit sur un $\oplus$-opérateur définissant un ordre total, alors elle est auto-stabilisante, quelque soit la topologie. Nous commençons par montrer que la stricte idempotence (*cf.*

(f) Voir en particulier § 4.3.1 page 71, à propos des algorithmes complètement anonymes. Notons que dans l'algorithme de l'arbre en profondeur, la pvar p est plus riche, mais elle revient à utiliser la pvar des index (pour obtenir un arbre en profondeur enraciné sur le nœud de plus petit index), qui est stockée dans les registres b^0_{in} au début de la `direct-association`. L'hypothèse reste donc raisonnable.

définition 78 page 109) est une condition nécessaire, avant de montrer qu'elle est suffisante (sous réserve que le s-opérateur définisse un ordre total).

6.2.2 Condition nécessaire

Nous avons vu que la k-idempotence est une condition nécessaire pour assurer la terminaison de la $\triangleleft$-`direct-association` (non-auto-stabilisante), avec $\triangleleft$ un r-opérateur (binaire ou n-aire). Nous montrons maintenant que la stricte idempotence des r-opérateurs est une condition nécessaire pour assurer l'auto-stabilisation du calcul global.

Lemme 79 *Si l'algorithme 5 page 75 de la* `direct-association` *est instancié avec un* r*-opérateur qui n'est pas strictement idempotent, alors il n'est pas auto-stabilisant.*

Preuve. Nous montrons la proposition en exhibant un réseau et une initialisation particulière pour lesquels l'algorithme de la `direct-association` ne se stabilise pas (au sens de l'auto-stabilisation), même si l'opérateur est idempotent.

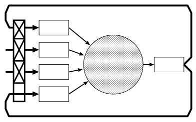


FIG. 72 – La stricte-idempotence est nécessaire pour obtenir l'auto-stabilisation.

Considérons le système distribué composé d'un seul nœud v et d'un seul lien, qui est une boucle de v à v (figure 72). Supposons que v utilise un r-opérateur binaire non strictement idempotent $\triangleleft$, défini avec le s-opérateur $\oplus$ et l'isomorphisme r de $(\mathbb{S}, \oplus)$ (définition 46 page 86). Soit y un élément de $\mathbb{S}$ vérifiant $y = \mathsf{r}(y)$. Considérons le cas où le nœud v est initialisé avec la donnée x plus grande (au sens de $\preceq_\oplus$, l'ordre total induit par $\oplus$) que y : $y \preceq_\oplus b^0_{\text{in}}[v] = x$ (*i.e.*, $x \oplus y = x$). Supposons de plus que $b^1_{\text{in}}[v] = e_\oplus$ au départ. Sans l'apparition d'une faute, le registre $b_{\text{out}}[v]$ stabilise sur la valeur x, et le nœud v est légitime.

Maintenant, supposons qu'il y ait eu une faute et que le registre $b_{\text{out}}[v]$ contienne temporairement une donnée erronée y. Cette donnée sera copiée dans $b^1_{\text{in}}[v]$ après l'application de la règle de lecture $R_1|\triangleleft$ de l'algorithme. Il n'y a alors plus moyen d'obtenir le bon résultat x, puisque la règle de calcul $R_2|\triangleleft$ fournit $x \triangleleft y = x \oplus \mathsf{r}(y) = x \oplus y = x$. Le nœud v ne sera donc jamais légitime, et l'algorithme instancié avec l'opérateur ∇ non strictement idempotent n'est donc pas auto-stabilisant. □

C'est la stricte croissance des r-fonctions des r-opérateurs strictement idempotents qui assure l'auto-stabilisation de l'algorithme de la `direct-association`, même lorsqu'il y a des circuits.

6.2.3 Condition suffisante : algorithme silencieux

Pour montrer que la stricte-idempotence conduit à l'auto-stabilisation, nous commençons par montrer que l'algorithme est silencieux.

Lemme 80 *L'algorithme 5 page 75 de la* `direct-association`*, instancié avec un* r*-opérateur n-aire ∇ strictement idempotent, est silencieux.*

Preuve. Il nous faut montrer que, partant d'une configuration initiale c_1 quelconque, toute exécution équitable est finie en l'absence de nouvelle faute (*i.e.*, c_1 marque le début de la phase de convergence, après la dernière faute survenue dans le réseau). En d'autres termes, il faut montrer qu'après un temps fini, aucune règle de copie $R_1|\nabla$ ni de calcul $R_2|\nabla$ n'est applicable.

Dans tout configuration initiale c_1, il y a un nombre fini de valeurs stockées dans les registres entrants $b_{\text{in}}[v]$, ou dans les registres sortants b_{out}. Sans perte de généralité, on peut considérer que les résultats stockés dans les registres sont des expressions formelles dans lesquelles les r-fonctions ne sont pas calculées (elles sont laissées sous la forme $\mathsf{r}(x)$). Cette notation des résultats sans calcul des r-fonctions permet de reconnaître les déplacements des valeurs initiales dans le réseau, et d'affirmer que, en l'absence de fautes, puisque l'ordre est total, il n'y a pas de création de nouvelles valeurs, mais simplement déplacement des valeurs initiales, r-augmentées à chaque passage d'un arc (des valeurs peuvent progresser dans différentes directions, tandis que d'autres peuvent disparaître).

En effet, toute règle de copie $R_1|\nabla$ sur un nœud v correspond à une progression des données des registres de sortie $b_{\text{out}}[u]$ des antécédents directs u de v vers les registres entrants $b_{\text{in}}^{\mathbf{i}(u)}[v]$. De même, puisque nous avons supposé que la relation d'ordre induite par le s-opérateur $\oplus$ (avec lequel est bâti le r-opérateur ∇) est totale, la règle de calcul $R_1|\nabla$ sur v correspond à la progression de la donnée d'un seul registre entrant $b_{\text{in}}^i[v]$, qui se retrouve dans le registre de sortie $b_{\text{out}}[v]$ de v, après avoir été r-augmentée par la r-fonction r_i de v (qui peut être l'identité si $i = 0$). En effet, on a, pour un seul $i \in \{0, \ldots, \delta_v^-\}$,

$$b_{\text{out}}[v] = \mathsf{r}_i\left(b_{\text{in}}^i[v]\right) = \nabla\left(b_{\text{in}}^0[v], \ldots, b_{\text{in}}^{\delta_v^-}[v]\right) = b_{\text{in}}^0 \oplus \mathsf{r}_1\left(b_{\text{in}}^1[v]\right) \oplus \ldots \oplus \mathsf{r}_{\delta_v^-}\left(b_{\text{in}}^{\delta_v^-}[v]\right)$$

S'il n'y a plus de progression de données à partir d'une configuration c_t, alors cette configuration est terminale, car cela signifie qu'aucune règle n'est applicable.

Supposons que le réseau soit un DAG. Dans ce cas, il n'y a pas de circuit, et la progression des données initiales ne peut être infinie ; l'algorithme est donc silencieux.

Supposons maintenant qu'il y ait un circuit dans le réseau. Dans ce cas, certaines données initiales pourraient sans cesse progresser. Soit x une valeur initiale présente sur le nœud v à la configuration c_i, à l'intérieur de la r-fonction de chemin r_P. Supposons qu'il existe une exécution partant de la configuration c_i telle que x atteigne de nouveau v à la configuration c_j, postérieure à c_i. Dans ce cas, x a été r-augmentée par toutes les r-fonctions des arcs que la donnée aura empruntée. Ces r-fonctions déterminent une r-fonction de chemin, qui correspond au chemin suivi par x entre les configurations c_i et c_j ; nous la notons $\mathsf{r}_{P_{c_i \to c_j}}$. En utilisant l'hypothèse de la stricte idempotence de l'opérateur ∇, toute r-fonction vérifie, pour tout $y \in \mathbb{S}, y \prec_{\oplus} r(y)$. La r-fonction de chemin r_P vérifie donc $y \prec_{\oplus} \mathsf{r}_P(y)$, pour tout $y \in \mathbb{S}$ (*cf.* lemme 68 page 95). Ainsi, dans la configuration c_j, la valeur initiale x présente dans l'un des registres entrants de v est r-augmentée par $\mathsf{r}_{P_{c_i \to c_j}}$ et on a $\mathsf{r}_P(x) \prec_{\oplus} \mathsf{r}_{P_{c_i \to c_j}} \circ \mathsf{r}_P(x)$. Cela signifie que l'expression du registre entrant de v qui contient x est strictement croissante à chaque arrivée de x en v. Supposons maintenant qu'il existe une exécution e_{c_i} dans laquelle x atteigne infiniment souvent v. Dans ce cas, il existe une configuration c_k de e_{c_i} telle que $b_{\text{in}}^0[v] \prec_{\oplus} \mathsf{r}_{P_{c_i \to c_k}} \circ \mathsf{r}_P(x)$. Cela signifie que l'expression contenant x ne peut être «choisie» par la règle de calcul de v, ce qui contredit le fait que la valeur x progresse indéfiniment.

Ainsi, pour toute exécution e_{c_1}, et en l'absence de nouvelles fautes, il existe une configuration terminale, et l'algorithme est silencieux. □

6.2.4 Condition suffisante : configuration terminale légitime

Pour montrer que la stricte idempotence est une condition suffisante à l'auto-stabilisation de la `direct-association`, il nous reste à montrer que la configuration terminale de toute exécution (il en existe toujours une, d'après le lemme précédent) est légitime.

Partition des sommets. Puisque nous avons supposé que l'ordre induit par le s-opérateur $\oplus$ (sur lequel est bâti le r-opérateur $\diamond$ strictement idempotent) est total, nous pouvons trier, pour toutes les configurations, les valeurs des registres entrants des nœuds du système. Cela nous permet, pour une configuration donnée, de classer les nœuds du réseau en plusieurs classes distinctes.

Définition 81 *Soit c une configuration ; on note Λ l'ensemble de toutes les variables d'entrées légitimes dans c. En utilisant la relation d'ordre total $\prec_{\oplus}$ induite par $\oplus$, on pose $\Lambda = \{\lambda_0, \lambda_1, \ldots, \lambda_k\}$ avec $\lambda_i \prec_{\oplus} \lambda_j$ pour $i < j$.*

Définition 82 *Soit c une configuration. On note Ψ_n l'ensemble des nœuds du réseau $G(V, E)$ tel que le plus petit registre entrant légitime* r*-augmenté soit égal à λ_n :*

$$\Psi_n = \left\{v \in V \textit{ tel que } \exists i, 0 \le i \le \delta_v^-, b_{in}^i[v] \textit{ est légitime}, \mathsf{r}_i\left(b_{in}^i[v]\right) = \lambda_n,\ \textit{et}\ \ \nexists n' < n \textit{ tel que } v \in \Psi_{n'}\right\}$$

Les lemmes suivants découlent des définitions ; ils nous servent pour la suite de la preuve.

Lemme 83 *Dans toute configuration, les valeurs des registres de sorties b_{out} légitimes ne sont jamais inférieures à λ_0.*

Preuve. Toute variable légitime est égale à une expression construite avec des données initiales augmentées par des r-fonctions de chemin strictement croissantes. Supposons que l'on ait $\mathcal{L}\left(b_{\text{out}}[v]\right) \prec_\oplus \lambda_0$. Dans ce cas, nous avons

$$\oplus\left\{\mathsf{r}_P\left(b^0_{\text{in}}[u]\right), u \in \Gamma^-_G(v), P \text{ chemin élémentaire de } u \text{ à } v\right\} \prec_\oplus \lambda_0$$

et donc il existe $u \in \Gamma^-_G(v)$ et une r-fonction de chemin r_P telle que $b^0_{\text{in}}[u] \prec_\oplus \mathsf{r}_P\left(b^0_{\text{in}}[u]\right) \prec_\oplus \lambda_0$, ce qui est impossible, par définition de λ_0. □

Lemme 84 *Tous les nœuds de Ψ_0 vérifient $\mathcal{L}(b_{out}[v]) = \mathcal{L}(b^0_{in}[v]) = b^0_{in}[v] = \lambda_0$.*

Preuve. On a $\mathcal{L}(b_{\text{out}}[v]) = \oplus\left\{\mathsf{r}_i\left(\mathcal{L}\left(b^i_{\text{in}}[v]\right)\right), 0 \le i \le \delta^-_v\right\}$ et puisque v est dans Ψ_0, $\mathcal{L}(b_{\text{out}}[v]) \prec_\oplus \lambda_0$. Mais suivant le lemme 83, aucune variable légitime n'est plus petite que λ_0, ce qui signifie que $\mathcal{L}(b_{\text{out}}[v]) = \lambda_0$.

Maintenant supposons qu'il existe un entier q, $1 \le q \le \delta^-_v$ tel que $\mathcal{L}(b_{\text{out}}[v]) = \mathsf{r}_q\left(\mathcal{L}(b^q_{\text{in}}[v])\right)$. Puisque les r-fonctions sont strictement croissantes, on a : $\mathcal{L}(b^q_{\text{in}}[v]) \prec_\oplus \mathsf{r}_q\left(\mathcal{L}(b^q_{\text{in}}[v])\right)$, soit $\mathcal{L}(b^q_{\text{in}}[v]) \prec_\oplus \lambda_0$, ce qui est impossible, par définition de λ_0. □

Méthode. Pour une configuration donnée, on considère l'assertion P^k_n qui signifie qu'il existe un chemin de longueur k de nœuds distincts illégitimes :

$$\begin{aligned} P^k_n \;:\; & \exists v \in G, \exists n_v \ge n \text{ tel que } v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma^{-1}_G(v), \ldots, u_k \in \Gamma^{-k}_G(v) \text{ tel que } u_i \in \Psi_{n_i} \text{ avec } n_i \ge n \quad (1 \le i \le k), \\ & (u_k, u_{k-1}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E, \text{ et } b_{\text{out}}[u_k] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \prec_\oplus \lambda_n \end{aligned} \tag{6.1}$$

En effet, $v \in \Psi_n$ signifie que le plus petit registre entrant r-augmenté de v est λ_n. Si le registre de sortie de v est plus petit que λ_n, alors son contenu provient d'un registre entrant r-augmenté qui est lui-même plus petit que λ_n. Ce registre ne peut être légitime, et v ne peut donc pas être légitime. Il en est de même pour les sommets $u_i \in \Psi_{n_i}$, dont les registres de sorties contiennent des valeurs plus petites que λ_{n_i}.

Pour montrer que, dans une configuration terminale c_t, tous les nœuds du réseau sont légitimes, nous procédons par contradiction et récurrence multiple sur P^k_n. **(a)** Nous montrons tout d'abord que si un nœud de Ψ_0 n'est pas légitime, alors P^1_0 est vérifié (lemme 88 page 115). **(b)** Puis nous montrons par récurrence sur k que si un nœud de Ψ_0 est non légitime, alors P^k_0 est vérifié pour tout $k \in \mathbb{N}^*$ (lemme 86 page suivante). **(c)** Nous concluons alors sur le fait que tous les nœuds de Ψ_0 sont légitimes (lemme 87 page 115). **(d)** Ensuite, nous montrons que si tous les nœuds de $\Psi_0 \cup \ldots \cup \Psi_{n-1}$ sont légitimes, et un nœud de Ψ_n ne l'est pas, alors P^1_n est vérifié (lemme 88 page 115). **(e)** Par récurrence sur k, nous montrons alors que si tous les nœuds de $\Psi_0 \cup \ldots \cup \Psi_{n-1}$ sont légitimes et aucun nœud de Ψ_n ne l'est, alors P^k_n est vérifié pour tout $k \in \mathbb{N}^*$ (lemme 89 page 116). **(f)** Nous concluons alors sur le fait que tous les nœuds de G sont légitimes (lemme 90 page 117).

Développement. Le raisonnement utilisé dans les lemmes suivants peut se résumer ainsi[(g)]. Si l'assertion P^k_n est vérifiée, alors le premier nœud u_1 du chemin est illégitime. Mais, puisqu'on se situe dans une configuration terminale, la règle de calcul de ce nœud n'est pas exécutable, ce qui

(g) Malgré les ressemblances entre les lemmes, il nous a semblé plus complexe de factoriser le raisonnement dans un lemme générique, dont les notations auraient été encore plus lourdes que de développer chaque preuve.

signifie qu'il a correctement effectué son calcul. Dans ce cas, et puisque l'ordre est total, la valeur de son registre de sortie $b_{\text{out}}[u_1]$ est égal à l'un de ses registres entrants $b^j_{\text{in}}[u]$, r-augmenté par la r-fonction correspondante. Puisque le r-opérateur est strictement idempotent, cette r-fonction est strictement expansive ($b^j_{\text{in}}[u]x \prec_\oplus \mathsf{r}(b^j_{\text{in}}[u])$), et ce registre est donc illégitime (sinon $b_{\text{out}}[u]$ le serait). Il ne peut donc s'agir du registre $b^0_{\text{in}}[u]$, qui est toujours légitime, par hypothèse. Il s'agit donc d'un registre entrant relié à un antécédent. Puisque la configuration est terminale, la règle de copie de u_1 n'est pas exécutable, ce qui signifie que u_1 a correctement recopié les valeurs de ses antécédents. L'antécédent qui correspond au $j^{\text{ème}}$ registre entrant de u_1 n'est donc pas légitime, et le chemin des sommets illégitimes s'agrandit.

Lemme 85 *Dans toute configuration terminale où il existe un nœud $v \in \Psi_0$ non légitime, P^1_0 est vérifiée.*

Preuve. D'après l'équation 6.1 page précédente, on a :

$$\begin{aligned} P^1_0 :\ & \exists v \in G, \exists n_v \geq 0 \text{ tel que } v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma^{-1}_G(v) \text{ tel que } u_1 \in \Psi_{n_1} \text{ avec } n_1 \geq 0, \\ & (u_1, v) \in E \text{ et } b_{\text{out}}[u_1] \prec_\oplus \lambda_0 \end{aligned}$$

Considérons un nœud $v \in \Psi_0$. Suivant le lemme 84, nous avons $\mathcal{L}(b_{\text{out}}[v]) = b^0_{\text{in}}[v] = \lambda_0$. Supposons maintenant qu'il existe une configuration terminale c_t telle que le nœud v ne soit pas légitime : $b_{\text{out}}[v] \neq \mathcal{L}\left(b_{\text{out}}[v]\right)$. Puisque la configuration c_t est terminale, aucune règle n'est applicable, et en particulier la règle de calcul $R| \diamond$, ce qui signifie que :

$$b_{\text{out}}[v] = \oplus\left\{\mathsf{r}_i\left(b^i_{\text{in}}[v]\right), 0 \leq i \leq \delta^-_v\right\} \prec_\oplus b^0_{\text{in}}[v] \prec_\oplus \lambda_0$$

Ainsi le nœud v a au moins un registre entrant de valeur r-augmentée plus petite que λ_0 :

$$\exists u_1 \in \Gamma^{-1}_G(v), \mathsf{r}^{u_1}_v\left(b^{\mathbf{i}(u_1)}_{\text{in}}[v]\right) \prec_\oplus \lambda_0$$

Puisque toutes les r-fonctions sont strictement expansives, nous avons :

$$\exists u_1 \in \Gamma^{-1}_G(v), b^{\mathbf{i}(u_1)}_{\text{in}}[v] \prec_\oplus \mathsf{r}^{u_1}_v\left(b^{\mathbf{i}(u_1)}_{\text{in}}[v]\right) \prec_\oplus \lambda_0$$

Maintenant, puisque la configuration considérée est terminale, la règle de copie $R| \diamond$ n'est pas exécutable, et toutes les variables entrantes du nœud v ont été correctement copiées depuis les antécédents directs de v. Formellement, nous avons :

$$\exists u_1 \in \Gamma^{-1}_G(v), b_{\text{out}}[u_1] = b^{\mathbf{i}(u_1)}_{\text{in}}[v] \prec_\oplus \mathsf{r}^{u_1}_v(b^{u_1}_{\text{in}}[v]) \prec_\oplus \lambda_0$$

On remarque alors que (u_1, v) est un chemin de longueur 1. De plus, le nœud u_1 est non-légitime (lemme 84 page précédente). Nous avons donc montré que, s'il existe une configuration terminale c_t dans laquelle un nœud v n'est pas légitime, alors l'assertion P^1_0 est vérifiée. □

Lemme 86 *Dans toute configuration terminale pour laquelle un nœud $v \in \Psi_0$ n'est pas légitime, l'assertion P^k_0 est vérifiée pour tout $k \in \mathbb{N}$.*

Preuve. Nous utilisons une récurrence sur k, dont le premier pas est donné par le lemme 85.

Supposons qu'il existe une configuration terminale c_t et un entier $k > 1$ telle que P^{k-1}_0 soit vérifiée :

$$\begin{aligned} P^{k-1}_0 :\ & \exists v \in G, \exists n_v \geq 0 \text{ tel que } v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma^{-1}_G(v), \ldots, u_{k-1} \in \Gamma^{-(k-1)}_G(v) \text{ tel que } u_i \in \Psi_{n_i} \text{ avec } n_i \geq n \quad (1 \leq i \leq k) \\ & (u_{k-1}, u_{k-2}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E \text{ et } b_{\text{out}}[u_{k-1}] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \\ & \prec_\oplus \lambda_0 \end{aligned}$$

Puisque c_t est terminale, aucune règle n'est exécutable. Le registre $b_{\text{out}}[u_{k-1}]$ de chaque nœud u_{k-1} de $\Gamma_G^{-(k-1)}(v)$ (antécédents à distance $k-1$ de v) contient donc, soit la valeur de $b_{\text{in}}^0[u_{k-1}]$, soit la valeur d'un autre registre entrant $b_{\text{in}}^i[u_{k-1}]$, r-augmentée par la r-fonction correspondante : $\mathsf{r}_{u_{k-1}}^i\left(b_{\text{in}}^i[u_{k-1}]\right)$, $1 \leq i \leq \delta_{u_{k-1}}^-$. Cependant, par définition, il n'y a pas de valeur initiale plus petite que λ_0. On a donc :

$$\begin{aligned}&\exists u_k \in \Gamma_G^{-k}(v) \text{ tel que : } (u_k, u_{k-1}) \in E \text{ et}\\&\mathsf{r}_{u_{k-1}}^{u_k}\left(b_{\text{in}}^{\mathbf{i}(u_k)}[u_{k-1}]\right) = b_{\text{out}}[u_{k-1}] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \prec_\oplus \lambda_0\end{aligned}$$

Puisque c_t est terminale, chaque valeur des registres entrants des nœuds de $\Gamma_G^{-(k-1)}(v)$ ont été correctement copiés depuis les registres de sortie des nœuds de $\Gamma_G^{-k}(v)$. On a donc :

$$\begin{aligned}&\exists u_k \in \Gamma_G^{-k}(v) \text{ tel que } (u_k, u_{k-1}) \in E \text{ et } b_{\text{out}}[u_k] = b_{\text{in}}^{\mathbf{i}(u_k)}[u_{k-1}] \text{ et}\\&\mathsf{r}_{u_{k-1}}^{u_k}\left(b_{\text{in}}^{\mathbf{i}(u_k)}[u_{k-1}]\right) = b_{\text{out}}[u_{k-1}] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \prec_\oplus \lambda_0\end{aligned}$$

Les r-fonctions sont strictement expansives, donc :

$$\begin{aligned}&\exists u_k \in \Gamma_G^{-k}(v) \text{ tel que } (u_k, u_{k-1}) \in E \text{ et}\\&b_{\text{out}}[u_k] \prec_\oplus b_{\text{out}}[u_{k-1}] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \prec_\oplus \lambda_0\end{aligned}$$

Nous obtenons donc l'assertion P_0^k. Nous avons donc montré par récurrence que l'assertion P_0^k est vérifiée pour tout $k \in \mathbb{N} \setminus \{0\}$. □

Lemme 87 *Dans toute configuration terminale, aucun nœud de Ψ_0 est illégitime.*

Preuve. Si c'était le cas, d'après le lemme 86 page précédente, P_0^k serait vérifiée, pour tout $k \in \mathbb{N} \setminus \{0\}$. Puisque P_0^k ne peut être vérifiée pour k plus grand que le nombre de nœuds du réseau, aucun nœud de Ψ_0 ne peut rester illégitime. □

Lemme 88 *S'il existe une configuration terminale c_t telle que chaque nœud $u \in \Psi_0 \cup \ldots \cup \Psi_{n-1}$ est légitime et s'il existe un nœud v de Ψ_n qui ne l'est pas, alors P_n^1 est vérifiée pour $n \in \mathbb{N}$.*

Preuve. Suivant l'équation 6.1 page 113, on a :

$$\begin{aligned}P_n^1 \ : \ &\exists v \in G, \exists n_v \geq n \text{ tel que } v \in \Psi_{n_v},\\&\exists u_1 \in \Gamma_G^{-1}(v) \text{ tel que } u_1 \in \Psi_{n_1}, \text{ avec } n_1 \geq n,\\&(u_1, v) \in E \text{ et } b_{\text{out}}[u_1] \prec_\oplus \lambda_n\end{aligned}$$

Considérons un nœud $v \in \Psi_n$. Il existe alors un entier q, $1 \leq q \leq \delta_v^-$, tel que $b_{\text{in}}^q[v]$ est légitime et $\mathsf{r}_v^q\left(b_{\text{in}}^q[v]\right) = \lambda_n$. Supposons maintenant qu'il existe une configuration terminale c_t dans laquelle le nœud v est illégitime : $b_{\text{out}}[v] \neq \mathcal{L}\left(b_{\text{out}}[v]\right)$. Puisque la configuration c_t est terminale, aucune règle n'est applicable, ce qui signifie que : $b_{\text{out}}[v] = \oplus\left\{\mathsf{r}_v^i\left(b_{\text{in}}^i[v]\right), 0 \leq i \leq \delta_v^-\right\} \prec_\oplus \lambda_n$ Supposons que $b_{\text{out}}[v] = b_{\text{in}}^0[v]$. Dans ce cas, on a $b_{\text{in}}^0[v] \prec_\oplus \lambda_n$. Mais puisque $b_{\text{in}}^0[v] = \mathcal{L}(b_{\text{in}}^0[v])$, v ne serait pas dans Ψ_n. On a donc :

$$\exists u_1 \in \Gamma_G^{-1}(v), \mathsf{r}_v^{u_1}\left(b_{\text{in}}^{\mathbf{i}(u_1)}[v]\right) = b_{\text{out}}[v] \prec_\oplus \lambda_n$$

Puisque toutes les r-fonctions sont strictement expansives, on a :

$$\exists u_1 \in \Gamma_G^{-1}(v), b_{\text{in}}^{\mathbf{i}(u_1)}[v] \prec_\oplus \mathsf{r}_v^{u_1}\left(b_{\text{in}}^{\mathbf{i}(u_1)}[v]\right) \prec_\oplus \lambda_n$$

Maintenant, puisque la configuration considérée est terminale, toutes les variables entrantes du nœud v ont été bien copiées depuis les antécédents directs de v. Formellement, nous avons :

$$\exists u_1 \in \Gamma_G^{-1}(v), b_{\text{out}}[u_1] = b_{\text{in}}^{\mathbf{i}(u_1)}[v] \prec_\oplus \mathsf{r}_v^{u_1}(\mathbf{i}(u_1)) \prec_\oplus \lambda_n$$

Puisque $b_{\text{in}}^{\mathbf{i}(u_1)}[v] \prec_\oplus \lambda_n$ et $v \in \Psi_n$, $b_{\text{in}}^{Indu_1}[v]$ ne peut être légitime. Donc $b_{\text{out}}[u_1]$ n'est pas légitime et le nœud u_1 est illégitime, ce qui signifie que $u_1 \in \Psi_{n_1}$ avec $n_1 \geq n$.

(u_1, v) est un chemin de longueur 1. Nous avons donc montré que, s'il existe une configuration terminale c_t dans laquelle le nœud v est illégitime, alors l'assertion P_n^1 est vérifiée, ce qui signifie qu'il existe un chemin de longueur 1 dont les nœuds sont illégitimes et distincts. □

Lemme 89 *S'il existe une configuration terminale c_t telle que tous les nœuds $u \in \Psi_0 \cup \ldots \cup \Psi_{n-1}$ sont légitimes et s'il existe un nœud v dans Ψ_n qui est illégitime, alors P_n^k est vérifiée pour tout $n, k \in \mathbb{N}$.*

Preuve. Nous montrons que, s'il existe une configuration terminale c_t dans laquelle le nœud $v \in \Psi_n$ est non-légitime, alors, pour tout $k \in \mathbb{N}$, P_n^k est vérifié. Le pas de base est donné par le lemme 88 page précédente.

Supposons qu'il existe configuration terminale c_t telle que P_n^{k-1} soit vérifiée :

$$\begin{aligned} P_n^{k-1}: \quad & \exists v \in G, \exists n_v \geq n \text{ tel que } v \in \Psi_{n_v}, \\ & \exists u_1 \in \Gamma_G^{-1}(v), \ldots, u_{k-1} \in \Gamma_G^{-(k-1)}(v), \\ & \forall i, 1 \leq i < k, \exists n_i \geq n, u_i \in \Psi_{n_i}, \\ & (u_{k-1}, u_{k-2}) \in E, \ldots, (u_2, u_1) \in E, (u_1, v) \in E \text{ et} \\ & \qquad b_{\text{out}}[u_{k-1}] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \prec_\oplus \lambda_n \end{aligned}$$

Puisque c_t est terminale, le registre de sortie $b_{\text{out}}[u_{k-1}]$ de chaque nœud u_{k-1} de $\Gamma_G^{-(k-1)}(v)$ est égal à sa donnée initiale $b_{\text{in}}^0[u_{k-1}]$, ou à l'une de ses variables d'entrée, r-augmentée par la r-fonction correspondante, $\mathsf{r}_v^i\left(b_{\text{in}}^i[u_{k-1}]\right)$, $1 \leq i \leq \delta_{u_{k-1}}^-$. Aucun registre légitime r-augmenté n'est plus petit que λ_n sur u_{k-1}, sinon $u_{k-1} \in \Psi_{n'}$ avec $n' < n$. En particulier, $b_{\text{in}}^0[u_{k-1}] \not\prec_\oplus \lambda_n$. On a donc :

$$\begin{aligned} & \exists u_k \in \Gamma_G^{-k}(v) \text{ tel que } (u_k, u_{k-1}) \in E \text{ et} \\ & \mathsf{r}_{u_{k-1}}^{u_k}\left(b_{\text{in}}^{\mathbf{i}(u_k)}[u_{k-1}]\right) = b_{\text{out}}[u_{k-1}] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \prec_\oplus \lambda_n \end{aligned}$$

Puisque c_t est terminale, les valeurs des registres entrants des nœuds de $\Gamma_G^{-(k-1)}(v)$ ont été correctement copiées depuis les registres de sortie des nœuds de $\Gamma_G^{-k}(v)$. On a donc :

$$\begin{aligned} & \exists u_k \in \Gamma_G^{-k}(v) \text{ tel que } (u_k, u_{k-1}) \in E \text{ et } b_{\text{out}}[u_k] = b_{\text{in}}^{\mathbf{i}(u_k)}[u_{k-1}] \text{ et} \\ & \mathsf{r}_{u_{k-1}}^{u_k}\left(b_{\text{in}}^{\mathbf{i}(u_k)}[u_{k-1}]\right) = b_{\text{out}}[u_{k-1}] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \prec_\oplus \lambda_n \end{aligned}$$

Les r-fonctions sont strictement expansives, donc

$$\begin{aligned} & \exists u_k \in \Gamma_G^{-k}(v) \text{ tel que } (u_k, u_{k-1}) \in E \text{ et} \\ & b_{\text{out}}[u_k] \prec_\oplus b_{\text{out}}[u_{k-1}] \prec_\oplus \ldots \prec_\oplus b_{\text{out}}[u_1] \prec_\oplus \lambda_n \end{aligned}$$

Finalement, $b_{\text{in}}^{\mathbf{i}(u_k)}[u_{k-1}]$ n'est pas légitime, sinon u_{k-1} ne serait pas dans Ψ_n ($b_{\text{in}}^{\mathbf{i}(u_k)}[u_{k-1}] \prec_\oplus \lambda_n$). Donc $b_{\text{out}}[u_k]$ n'est pas légitime et le nœud u_k ne l'est donc pas non plus, ce qui signifie que $u_k \in \Psi_{n'}$ avec $n' \geq n$. Cela nous donne P_n^k. Nous avons donc montré par récurrence que l'assertion P_n^k est vérifiée pour tout $k \in \mathbb{N} \setminus \{0\}$. □

Lemme 90 *Pour toute configuration terminale c_t, tous les nœuds du réseau sont légitimes.*

Preuve. S'il existe une configuration terminale c_t telle que tous les nœuds $u \in \Psi_0 \cup \ldots \cup \Psi_{n-1}$ sont légitimes et qu'il existe un nœud $v \in \Psi_n$ non légitime, alors P_n^k est vérifié pour $k \in \mathbb{N} \setminus \{0\}$ et $n \in \mathbb{N}$ (lemme 89). Puisque le réseau est fini, P_n^k ne peut être vérifié pour k plus grand que le nombre de nœuds du réseau. Donc tous les nœuds de Ψ_n sont légitimes pour $n \in \mathbb{N}$. Tous les nœuds du réseau sont donc légitimes. □

6.2.5 Conclusion

Les différents résultats partiels obtenus dans cette section permettent finalement de montrer le résultat suivant :

Théorème 91 *Si les registres b_{in}^0 des nœuds du réseau G, dans lesquels sont copiés les valeurs de la pvar p, ne sont pas modifiables (e.g., ROM), et si le* r*-opérateur ∇ est basé sur un* s*-opérateur $\oplus$ induisant un ordre total, alors l'algorithme 5 page 75 de la $\diamond$-*`direct-association`(net_G, p) *est auto-stabilisant si et seulement si ∇ est strictement idempotent.*

Preuve. *Condition nécessaire.* D'après le lemme 79, si l'opérateur ∇ n'est pas strictement idempotent, alors l'algorithme n'est pas auto-stabilisant.

Condition suffisante. D'après le lemme 80, si l'opérateur ∇ est strictement idempotent, l'algorithme est silencieux, c'est-à-dire que pour toute configuration initiale c_1, toutes les exécutions $e = c_1, c_2, \ldots$ aboutissent à une configuration terminale, dans laquelle plus aucune règle n'est applicable. Or, d'après le lemme 90, toutes les configurations terminales sont telles que tous les nœuds du réseau sont légitimes, ce qui signifie que toutes les configurations terminales sont légitimes. Cela donne le théorème. □

6.2.6 Méthode matricielle.

Nous avons modélisé le calcul d'une `direct-association` avec une itération asynchrone au § 4.2.4 page 68 [(h)]. Nous proposons ici une solution matricielle à l'étude de l'auto-stabilisation de la `direct-association`.

La modélisation matricielle note par la matrice $\mathbf{A}$ les relations entre les nœuds provenant des r-fonctions. Celles-ci sont inaltérables, par hypothèse, car elles font partie du code. Le vecteur $\mathtt{B}$ (*cf.* équation 4.4 page 62) donne les valeurs des registres b_{in}^0, qui sont tous masqués, et contiennent les valeurs initiales des nœuds. Les valeurs des registres b_{out} sont données par le vecteur $\mathtt{X}_k$. $\mathtt{X}_0$ donne les valeurs initiales de ces registres. Pour montrer, de façon matricielle, que la `direct-association` est auto-stabilisante, il nous faut montrer que l'itération asynchrone 4.7 page 70 converge, quelle que soit l'initialisation $\mathtt{X}_0$ du réseau, c'est-à-dire quelle que soit la configuration initiale.

Proposition 92 *L'itération asynchrone de l'équation 4.7 page 70, qui correspond au calcul global asynchrone ∇-*`direct-association` *sur un réseau quelconque converge pour toute configuration initiale (tout vecteur initial $\mathtt{X}_0$), lorsque le* r*-opérateur ∇ est strictement idempotent et basé sur un* s*-opérateur $\oplus$.*

Preuve. Les conditions de convergence **(d)** et **(f)** de Üresin et Dubois sont toujours respectées (*cf.* § 4.2 page 63) car la matrice $\mathbf{A}$, qui code le réseau quelconque et les r-fonctions strictement expansives, n'a pas changé (*cf.* proposition 25 page 70). Il reste à vérifier la condition **(e)**, qui concerne la convergence de l'itération synchrone $\mathtt{X}_{k+1} = \mathbf{A} \otimes \mathtt{X}_k \ \oplus \ \mathtt{B}$, et ce, pour tout

(h) Voir aussi § 5.5 page 101 pour les r-opérateurs.

vecteur initial $\mathtt{X}_0$. On a :

$$
\begin{array}{llcl}
\text{On a :} & \mathtt{X}_1 & = & \mathbf{A}\otimes\mathtt{X}_0 \;\oplus\; \mathtt{B} \\
 & \mathtt{X}_2 & = & \mathbf{A}\otimes\mathtt{X}_1 \;\oplus\; \mathtt{B} \quad = \quad \mathbf{A}^2\otimes\mathtt{X}_0 \;\oplus\; \mathbf{A}\otimes\mathtt{B} \;\oplus\; \mathtt{B} \\
 & & & \dots \\
\text{soit :} & \mathtt{X}_k & = & \mathbf{A}^k\otimes\mathtt{X}_0 \;\oplus\; \mathbf{A}^{k-1}\otimes\mathtt{B} \;\oplus\; \cdots \;\oplus\; \mathbf{A}\otimes\mathtt{B} \;\oplus\; \mathtt{B} \\
 & & = & \mathbf{A}^k\otimes\mathtt{X}_0 \;\oplus\; \left(\mathbf{A}^{k-1}\oplus\cdots\oplus\mathbf{A}\oplus\mathbf{E}_\otimes\right)\otimes\mathtt{B} \\
 & & = & \mathbf{A}^k\otimes\mathtt{X}_0 \;\oplus\; \mathbf{A}^{(k-1)}\otimes\mathtt{B} \quad (\textit{cf.}\ \S\ 4.1.5\ \text{page } 58)
\end{array}
$$

Appelons h le plus grand élément plus petit que toutes les composantes de la matrice[(i)] $\mathbf{A}$. Puisque toutes les r-fonctions sont strictement expansives, on a :

$$e_\otimes \quad \prec_\oplus \quad \mathsf{h} \stackrel{\text{déf.}}{=} \bigoplus_{1\leq i,j\leq n} \mathbf{A}[i][j]$$

Soit $\mathbf{E}$ et $\mathbf{H}$ les matrices définies par $\mathbf{E}[i][j] = e_\otimes$ et $\mathbf{H}[i][j] = \mathsf{h}$, pour tout i et j dans $\{1,\dots,n\}$. On a :

$$E_\otimes \quad \prec_\oplus \quad \mathbf{H} \quad \preceq_\oplus \quad \mathbf{A}$$

ce qui donne $\mathbf{H}^k \preceq_\oplus \mathbf{A}^k$. Or $\lim_{k\to+\infty}\mathbf{H}^k = \mathbf{E}_\oplus$ (car $e_\otimes \prec_\oplus \mathsf{h}$). Donc $\lim_{k\to+\infty}\mathbf{A}^k = \mathbf{E}_\oplus$. Puisque il n'y a pas de circuit absorbant (grâce à l'idempotence des r-opérateurs), $\mathbf{A}^{(k-1)}$ converge vers $\mathbf{A}^*$ quand k augmente (*cf.* § 4.1.5 page 58). On a donc :

$$
\begin{array}{l}
\forall\, \mathtt{X}_0 \in \mathbb{S}^n, \quad \exists k_0 \in \mathbb{N} \quad \text{tel que :} \\
\mathtt{X}_k \;=\; \mathbf{A}^k\otimes\mathtt{X}_0 \;\oplus\; \mathbf{A}^{(k-1)}\otimes\mathtt{B} \;=\; \mathbf{A}^{(k-1)}\otimes\mathtt{B} \;=\; \mathbf{A}^*\otimes\mathtt{B} \quad \text{pour} \quad k\geq k_0
\end{array}
$$

et le système converge quel que soit le vecteur initial $\mathtt{X}_0$.

En conclusion, les conditions **(d)**, **(e)** et **(f)** de Üresin et Dubois sont respectées pour tout vecteur initial $\mathtt{X}_0$, et l'itération asynchrone converge donc pour toute configuration initiale. □

On peut noter que l'hypothèse de l'ordre total (du s-opérateur sur lequel est basé le r-opérateur) n'a pas été nécessaire dans cette preuve, alors que nous l'avions supposée pour la première preuve de l'auto-stabilisation. Cependant, la modélisation matricielle du calcul distribué a gommé les registres entrants : ceux-ci se sont retrouvés confondus avec les registres de sorties des antécédents. Ainsi, le calcul local correspond à lire toutes les valeurs des registres des antécédents, à effectuer le calcul, et à écrire le résultat. Ce comportement correspond à l'atomicité composite et non à la lecture-écriture atomique (*cf.* § 4.3.1 page 71).

En conclusion, la modélisation matricielle du calcul asynchrone de la `direct-association`, rendue possible grâce à l'algèbre de chemins et aux itérations asynchrones, conduit à une preuve de l'auto-stabilisation plus concise que celle basée sur les règles, et qui ne nécessite pas l'ordre total. Mais elle est valable dans un cadre plus restreint de l'atomicité composite, au lieu du cadre général de la lecture-écriture atomique.

6.3 Conclusion

Dans ce chapitre, nous avons étudié divers aspects du contôle des réseaux associatifs.

(i) Ici, les éléments de la matrice peuvent être des r-fonctions. La relation d'ordre $\preceq_\oplus$ sur $\mathbb{S}$ est étendue en une relation d'ordre partiel sur les fonctions de $\mathbb{S}$ dans $\mathbb{S}$: le minimum (au sens de $\oplus$) des fonctions f et g est la fonction h construite en prenant, en chaque $x\in\mathbb{S}, \mathsf{f}(x)\oplus\mathsf{g}(x)$. En général, $\mathsf{h}\neq\mathsf{f}$ et $\mathsf{h}\neq\mathsf{g}$. L'ordre est aussi étendu aux vecteurs et aux matrices, en considérant le minimum (au sens de $\oplus$) terme à terme. Il s'agit là aussi d'un ordre partiel.

L'hypothèse de non coordination des calculs et des communications dans le réseau associatif permet d'autoriser la perte d'un message. La propriété d'idempotence des opérateurs utilisés sur des réseaux quelconques permet d'autoriser la duplication des messages. Enfin, si les boucles des nœuds ne sont par masquées, la commutativité et l'associativité des s-expressions (obtenues avec les s- et les r-opérateurs) autorise le désordonnancement des messages.

Nous avons ensuite montré que, si le r-opérateur $\triangleleft$ est basé sur un s-opérateur définissant un ordre total, alors il y a équivalence entre la stricte idempotence de l'opérateur et l'auto-stabilisation de la $\triangleleft$-`direct-association`, sur tout réseau.

Outre l'intérêt pour le modèle des réseaux associatifs, ce résultat élargit les applications des r-opérateurs aux systèmes distribués. En effet, ces opérateurs peuvent être utilisés comme un moyen de conception d'algorithmes auto-stabilisants. Nous avons, en outre, obtenu une preuve générique, valable pour tous les r-opérateurs, c'est-à-dire commune à plusieurs algorithmes.

Nous indiquons au chapitre 7 quels sont les algorithmes qui sont auto-stabilisants. Citons notamment la recherche d'un arbre en profondeur, d'une forêt des poids minimum ou des chemins de fiabilité maximale.

Enfin, nous avons indiqué comment le raisonnement matriciel peut s'appliquer à ce type de problème. Bien que réductrice sur certains aspects, cette modélisation nous semble très intéressante. En effet, elle permet de faire le lien entre des domaines assez divers : algèbre max-plus, itérations asynchrones, systèmes distribués. Les r-opérateurs permettent ainsi de faire des liens fructueux entre des domaines différents.

Pour finir, signalons que ce chapitre a aussi permis de mieux caractériser les r-opérateurs; la figure 73 complète la figure 68 page 102.

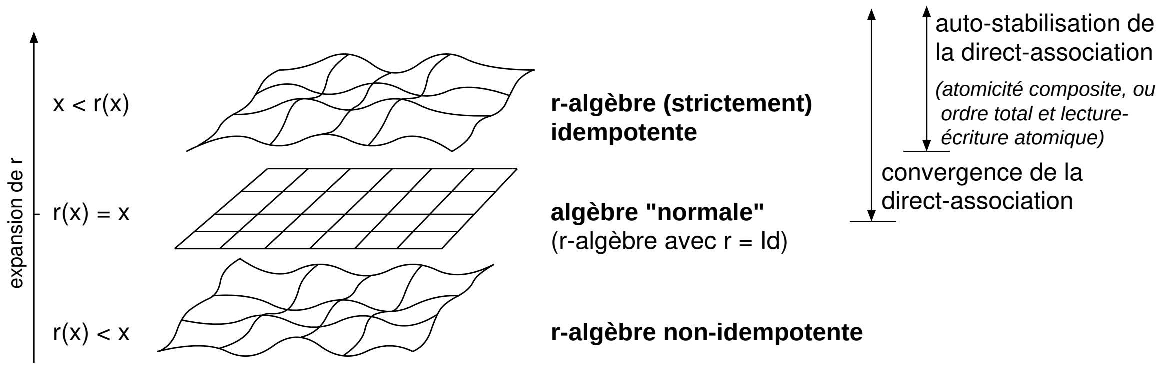


FIG. 73 – *Les* r*-algèbres strictement idempotentes assurent l'auto-stabilisation de la* `direct-association`*.*

Troisième partie

Applications et Mises en œuvres

Chapitre 7

Algorithmique

Dans la première partie, nous avons présenté le modèle des réseaux associatifs, tel que défini à l'origine, et un exemple d'utilisation : la segmentation de Voronoï. Nous avons ensuite introduit une nouvelle définition, au travers d'une nouvelle primitive de calcul asynchrone : la `direct-association`. Dans la seconde partie, nous avons étudié — sous différentes modélisations — les propriétés de la `direct-association` et des r-opérateurs, qui assurent sa terminaison.

L'étude des applications du modèle — que nous abordons dans cette troisième partie — nécessite de définir une algorithmique adaptée à ses spécificités : parallélisme de données irréguliers et calculs asynchrones sur des graphes notamment. Nous développons dans ce chapitre une algorithmique variée, autorisant l'évaluation des applications du modèle. Cette algorithmique est parfois une adaptation d'algorithmes existants, et parfois radicalement différente de l'existant.

Nous commençons par résumer les spécificités de la programmation dans un réseau associatif. Au § 7.2, nous donnons plusieurs algorithmes permettant de détecter la nature du réseau ; celle-ci détermine les opérateurs utilisables par la primitive. Au § 7.3, nous illustrons l'application de la primitive sur un graphe de précédence de tâches d'un algorithme. Au § 7.4, nous utilisons les r-opérateurs pour résoudre divers problèmes d'algorithmique de graphes. Nous étudions ensuite les applications de la `direct-association` en algorithmique géométrique au § 7.5, et en traitement d'images au § 7.6. Enfin, nous concluons sur les possibilités algorithmiques du modèle au § 7.7, où nous donnons une méthodologie de programmation. Nous indiquons aussi les connexions entre les algorithmes développés dans ce chapitre et l'algèbre de chemins matricielle ou les systèmes distribués.

7.1 Introduction

Nous précisons les particularités de la programmation d'un réseau associatif, les primitives et les conventions utilisées.

7.1.1 Modèle de programmation

Spécificités. Le modèle des réseaux associatifs offre au programmeur un parallélisme de données, augmenté de certaines caractéristiques fondamentales. La première concerne l'utilisation des sous-réseaux du graphe de base, et la seconde les calculs asynchrones sur ces graphes.

Le graphe de base est soit une donnée du problème, soit un outil pour le résoudre. Le premier cas correspond, par exemple, à un système distribué. Le réseau est figé, et il faut s'en accommoder pour programmer les protocoles. Le second cas correspond, par exemple, à l'algorithmique d'images. Le graphe de base est choisi par le programmeur, en fonction du problème à traiter. Ce peut être une grille simple, en 4-connexité, ou une grille en 8-connexité, plus précise dans les contours, mais non planaires[(a)], ou bien encore une grille hexagonale, planaire, et dans laquelle les voisins d'un pixel

(a) Un graphe de base non planaire nécessite certaines précautions dans le maniement des graphes virtuels, pour assurer sont tous à la même distance (on passe sans difficulté de l'une à l'autre). Dans ce cas, le graphe de base s'apparente à une machine virtuelle d'instanciation. Toutes les variables parallèles (pvars) ont la forme du graphe de base. Les objets irréguliers sont codés par des sous-réseaux de ce réseau de base. Lorsque le graphe de base est une maille, toutes sortes d'objets peuvent être codées par des sous-réseaux (*cf.* annexe C page 195). Nous illustrons dans ce chapitre les aptitudes du modèle à l'algorithmique de graphe.

La seconde caractéristique fondamentale du modèle concerne les calculs sur les sous-réseaux. La `direct-association` propose un calcul global asynchrone, qui ne se termine que lorsque tout le réseau est stable. Les s-opérateurs fournissent un seul résultat par composante fortement connexe du réseau, mais les r-opérateurs permettent d'obtenir des résultats différents sur des nœuds de la même composante. Nous illustrons dans ce chapitre l'intérêt de la primitive data-parallèle asynchrone, ainsi que des r-opérateurs, dans plusieurs champs de l'algorithmique.

Algorithme de réseau. La programmation d'un réseau associatif peut être vue de deux manières différentes. La première est semblable à celle des modèles de programmation à parallélisme de données classiques : on considère les primitives comme des opérateurs de manipulation des données, essentiellement stockées dans des tableaux.

La seconde nous semble plus riche. Elle consiste à prendre davantage en compte le réseau, composé des nœuds et des liens les connectant. Ce point de vue permet de mieux tirer parti de l'information codée par les relations inter-données, plutôt que de n'utiliser que les valeurs des données.

La philosophie de programmation des réseaux associatifs incite à exploiter le réseau. Nous montrons dans ce chapitre que les algorithmes en tirent un bon parti, notamment grâce aux r-opérateurs. Nous parlerons d'*algorithme de réseau* pour qualifier un algorithme construit dans cette philosophie. L'usage de l'asynchronisme de la `direct-association` n'est intéressant que dans cette optique.

Par exemple, pour calculer les distances depuis une graine donnée dans une image, on peut diffuser l'abscisse et l'ordonnée de la graine. Mais on peut aussi utiliser le fait qu'il suffit d'ajouter 1 à la valeur calculée par un nœud voisin plus proche de la graine ; la distance se calcule ainsi par diffusion dans le réseau (*cf.* § 7.5.1). Pour la construction des régions de Voronoï, chaque pixel doit choisir sa graine la plus proche, et cela nécessite de nombreuses itérations (*cf.* § 2.2.2 page 24). Nous expliquons au § 7.5.1 que la diffusion asynchrone des distances agit comme une relaxation, qui supprime les itérations. Les régions sont obtenues en une seule `direct-association`. L'intérêt du remplacement des itérations par une seule primitive asynchrone (au grain plus gros) est expliqué au chapitre suivant.

Les algorithmes de réseau ont des caractéristiques en commun avec les systèmes distribués, dont nous nous inspirons pour certains algorithmes.

Complexité. La complexité de la `direct-association` est étudiée au § 6.1 page 103. Nous prenons le parti de privilégier la complexité en temps à la complexité en messages dans l'algorithmique que nous présentons. Cependant, dans de nombreux algorithmes, la taille des données véhiculées par le réseau est déterminante dans le nombre d'étapes globales. On retrouve là une caractéristique classique du calcul distribué. C'est le cas, par exemple, pour la détermination des composantes fortement connexes. Nous donnons plusieurs algorithmes pour ce problème, avec différentes tailles de messages (*i.e.*, largeur de la pvar, *cf.* § 3.2.1 page 36).

Nous mesurons les complexités des algorithmes en nombre d'étapes globales, et en nombre d'étapes locales. Une étape globale correspond à une `direct-association`. Les étapes locales ne font pas appel à la stabilisation : elles correspondent aux calculs locaux (tels que des affectations de pvars) et aux `step-associations` et `step-exclusive-associations`.

que deux régions ne se chevauchent pas en utilisant deux diagonales qui se croisent.

7.1.2 Primitives

Primitives de calcul. L'instruction $p = \diamond - \texttt{association}(net_G, q)$ indique que la pvar q est construite en appliquant une `association` sur le sous-réseau codé par la pvar de masques net_G, et initialisé avec la pvar q. Les éléments neutres sont souvent «extrêmes» dans les algorithmes que nous donnons (par exemple, nous utilisons $+\infty$). Mais ces éléments sont aisément codés en pratique, en utilisant une valeur finie (la plus grande admise dans le cas de $+\infty$). Nous utilisons aussi les primitives classiques `Global-and(pvar p)` et `Global-or(pvar p)`, qui renvoient le calcul d'un and ou d'un or sur toutes les valeurs de la pvar p. Ces primitives pourraient cependant être remplacées par une `association` sur le réseau de base (s'il est fortement connexe), en utilisant l'opérateur concerné. Bien qu'il soit possible de traduire les primitives de programmation parallèle classiques (*send*, *broadcast*, *scan*, etc.) dans le modèle [Ducourthial, 1995], nous ne les utilisons pas dans ce chapitre, car elles ne sont pas spécifiques au modèle de calcul.

Manipulation du réseau. Les possibilités algorithmiques du modèle sont plus nombreuses, et plus efficaces, lorsque le graphe de base est orienté symétrique. On peut alors considérer le *sous-réseau inverse* G^{-1}, défini en inversant tous les arcs du sous-réseau G. L'hypothèse du graphe inverse n'est pas toujours vérifiée, suivant que le réseau de base est choisi ou subi (*cf.* 7.1.1).

Hypothèse 93 (Graphe inverse) *À tout sous-réseau G, codé par la pvar de masques* net_G*, correspond le sous-réseau G^{-1}, codé par la pvar de masques* $\mathrm{net}_G{}^{-1}$.

On note $net_G^{-1} = net_{G^{-1}}$ la pvar de masques du sous-réseau inversé. Sa construction peut être réalisée en δ_G^- `step-associations` pour récupérer les masques de chaque nœud voisin. Elle peut aussi être réalisée en une seule `step-association`, en utilisant une pvar de largeur $\delta_G^- \times$ `sizeof(masque)` $= \delta_G^- \cdot \log\left(\delta_G^-\right)$ bits. Le plus efficace consiste cependant à transmettre, une seule fois au début du programme, l'indice $\mathbf{i}_v(u)$ de l'arc (u, v) au sommet u, pour chaque antécédent direct u de v, et pour tout nœud v. Dans ce cas, on peut utiliser un opérateur qui extrait le $i^{\text{ème}}$ bit de la donnée arrivant sur le $i^{\text{ème}}$ arc entrant, et qui concatène tous les bits ainsi obtenus sur les arcs entrants. Une seule `step-association` avec cet opérateur et la pvar des masques permet alors d'obtenir le graphe inverse.

La pvar de masques $net_G \cup net_{G'}$ provient de l'union des deux sous-réseaux codés par les pvars net_G et $net_{G'}$. Elle est obtenue en calculant un or entre les deux masques sur chaque nœud. On peut aussi considérer le complémentaire d'un réseau, comme nous l'avons fait au chapitre 2 avec le graphe frontière, complémentaire du graphe des régions. La pvar de masques du sous-réseau complémentaire est obtenue en inversant les bits du masque sur chaque nœud. Ces manipulations ne demandent pas de communications.

Nous généralisons la primitive `link-with-equals` de la Maille Associative (*cf.* § 1.2.2 page 13) : la primitive `link-with-equals(pvar⟨masque⟩` net_G`, pvar` p_1`, pvar` p_2`)` permet de fabriquer un sous-réseau du réseau codé par la pvar net_G, qui contient l'arc (u, v) si et seulement si $p_1[u] = p_2[v]$.

Notons que dans les algorithmes de ce chapitre, certaines manipulations du réseau sont écrites en toute lettre pour faciliter la compréhension des algorithmes. Le maniement d'un graphe virtuel, et la traduction des `associations` spécifiées sur ce graphe en `associations` sur les graphes des arcs et des sommets le composant, sont expliqués au § 2.2.1 page 22.

7.2 Nature du réseau

Dans cette section, nous proposons plusieurs algorithmes permettant de reconnaître la structure du réseau sous-jacent. Cette structure est primordiale dans le modèle des réseaux associatifs, car elle détermine les opérateurs qu'il est possible d'utiliser. Dans un environnement de programmation tel qu'ANET (*cf.* chapitre 8), des méthodes plus efficaces que les algorithmes de réseau présentés ici pourront cependant être mises en œuvre. L'intérêt de cette section est double. Tout d'abord, on montre que la mise en œuvre des `associations` est possible sans aide extérieure (telle que l'information plus ou moins centralisée d'une bibliothèque de programmation). Ensuite, les algorithmes présentés illustrent diverses techniques de la programmation des réseaux associatifs, ainsi que les aptitudes du modèle à l'algorithmique de graphe.

7.2.1 Recherche des sources

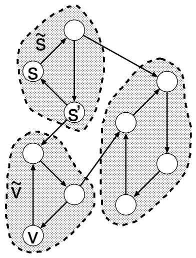


FIG. 74 – Un graphe peut ne pas avoir de source ; mais il a toujours au moins une source logique. Les nœuds du graphe réduit $\widetilde{G}$ (CFC) sont représentés en grisé.

Les sources d'un graphe sont de première importance pour plusieurs algorithmes, tels que les plongements d'arborescences ou les recherches de composantes connexes. Il s'agit de sommets permettant de diffuser une information, ou d'identifier des groupes de sommets. La recherche de ces sources est donc souvent une étape obligée dans l'algorithmique des réseaux associatifs. Une *source* est définie comme un sommet qui n'a pas d'antécédent. Cependant il existe des graphes, tel que celui de la figure 74, qui n'ont pas de source. Cela nous conduit à définir les *sources logiques*. On appelle *graphe numéroté* un graphe $G(V, E)$ muni d'une numérotation injective $\mathsf{n}_G : V \rightarrow \mathbb{N}$ de ses sommets. Lorsque l'hypothèse de l'index (hypothèse 10 page 36) est admise, le réseau de base, ainsi que tous les sous-réseaux, sont numérotés.

Définition 94 *Un sommet u est une source logique du graphe numéroté $G(V, E)$ si sa composante fortement connexe est source dans le graphe réduit $\widetilde{G}$ (cf. définition 17 page 42), et s'il a le plus grand numéro dans cette composante.*

Proposition 95 *Pour tout sommet v d'un graphe numéroté $G(V, E)$, il existe une source logique s dans $\overline{\Gamma^-_G}(v)$, l'ensemble des antécédents de v augmenté de lui même. De plus, s est la seule source logique de $\overline{\Gamma^-_G}(s)$.*

Preuve. Soit v un sommet de G et $\widetilde{v}$ sa composante fortement connexe, qui est un sommet du graphe réduit $\widetilde{G}$ (figure 74). Puisque $\widetilde{G}$ est un DAG (éventuellement réduit à un seul sommet), son sommet $\widetilde{v}$ a au moins une source dans l'ensemble de ses antécédents augmenté de lui-même. Soit $\widetilde{s} \in \overline{\Gamma^-_{\widetilde{G}}}(\widetilde{v})$ cette source. Le sommet $s \in V$ défini par $\mathsf{n}_G(s) = \mathsf{max}\{\mathsf{n}_G(u), \widetilde{u} = \widetilde{s}\}$ est une source logique ; elle fait partie des antécédents de v augmenté de lui-même.

Soit $s \in V$ une source logique de G, et s' une autre source logique telle que $s' \in \overline{\Gamma^-_G}(s)$. Puisque $\widetilde{s}$ n'a pas d'antécédent dans $\widetilde{G}$, on a $\widetilde{s'} = \widetilde{s}$. De plus, on a $\mathsf{n}_G(s') = \mathsf{max}\{\mathsf{n}_G(u), \widetilde{u} = \widetilde{s}\} = \mathsf{n}_G(s)$, d'où, par injection de la numérotation n_G, $s = s'$. □

On peut remarquer que toutes les sources d'un graphe sont aussi des sources logiques, mais l'inverse est faux, comme l'indique la figure 74. Une source logique est une source seulement si elle n'a pas d'antécédents. Les sources logiques d'un DAG sont ses sources.

Proposition 96 *Un graphe numéroté $G(V, E)$ admet une arborescence couvrante orientée de la racine vers les feuilles si et seulement s'il n'admet qu'une seule source logique.*

Preuve. Si G admet une unique source logique s, alors tout sommet $v \neq s$ de G a cette source parmi ses antécédents. En ne retenant, pour chaque sommet v, qu'un seul plus court chemin de s à v, on obtient une arborescence couvrante orientée de la racine s vers les feuilles.

Supposons maintenant que G admette une arborescence couvrante T orientée depuis une racine r vers ses feuilles. Dans ce cas, tout sommet v de G a le sommet r dans ses antécédents.

La composante fortement connexe $\widetilde{r}$ de r est donc le seul sommet du graphe réduit $\widetilde{G}$ qui soit une source dans $\widetilde{G}$. Dans ce cas, il existe un seul sommet s vérifiant $\mathsf{n}_G(s) = \max\{\mathsf{n}_G(v), \widetilde{v} = \widetilde{r}\}$, et s est l'unique source logique de G. □

L'algorithme 6 permet de déterminer les sources logiques d'un réseau dont les nœuds sont numérotés (*cf.* hypothèse 10 page 36). Il détecte tout d'abord des candidats (lignes 1-4) : ce sont les nœuds qui ont le plus grand index de tous leurs antécédents. Mais certains nœuds, qui ont un index plus grand que les sources logiques se trouvant dans leurs antécédents, ont été sélectionnés «par erreur». On supprime donc ces nœuds de la sélection pour obtenir les sources logiques (lignes 5-10).

Algorithme 6 SourcesLogiques(**pvar**⟨ **masque** ⟩ net_G)

▶ net$_G$ *code le réseau ; p est la pvar résultat ; $\imath$ est la pvar des index (cf. hypothèse 10 page 36).*
1 **pvar** $p \leftarrow$ `max-direct-association`$(net_G, \imath)$
2 **partout où** $p \neq \imath$ **faire**
3     $p \leftarrow +\infty$
4 **fin partout où**
5 p = `min-direct-association`(net_G, p)
▶ *Certains sommets v sélectionnés ont une source logique s parmi leurs antécédents ; dans ce cas, $p[s] < p[v]$, et on peut les éliminer.*
6 **partout où** $p \neq \imath$ **faire**
7     $p \leftarrow +\infty$
8 **ailleurs**
9     $p \leftarrow 0$
10 **fin partout où**
11 **retourner** p ▶ *Les sources logiques sont les sommets v tels que $p[v] = 0$.*

Les sources sont quant à elles aisément détectées.

Algorithme 7 Sources(**pvar**⟨ **masque** ⟩ net_G)

pvar $p \leftarrow 0$
partout où $net_G \neq 0$ **faire**
    $p \leftarrow +\infty$ ▶ *Masque non nul ; le nœud n'est pas une source.*
fin partout où
retourner p ▶ *Les sources sont les sommets v tels que $p[v] = 0$.*

7.2.2 Algorithmes sur des DAGs

Les DAGs permettent d'utiliser des opérateurs non-idempotents (*cf.* proposition 1 page 30). Il est donc important de savoir reconnaître ce type de réseau. Cela peut être fait par la recherche des composantes fortement connexes (*cf.* § 7.2.4 page 132) ; nous proposons cependant deux autres algorithmes (algorithmes 9 page suivante et 11 page 129). Le premier illustre la «limitation du résultat d'un opérateur» ; le second utilise des itérations sur un sous-réseau acyclique pour les calculs non-idempotents. Nous commençons par donner un algorithme de tri topologique, qui est utilisé par les deux autres algorithmes. Cet algorithme de tri sert aussi à ordonner les composantes fortement connexes du graphe réduit (qui est un DAG). Cela permet de calculer les `associations` du haut (racines) vers le bas (feuilles) du DAG, et ainsi de ne pas faire de calculs locaux inutiles, dans des composantes dont les antécédentes n'ont pas encore une valeur définitive (*cf.* § 8.3.2 page 162).

Tri topologique. Le tri topologique est réalisé avec une seule `direct-association`, à l'aide du r-opérateur binaire maxinc défini sur $\mathbb{N}$ par $\mathsf{maxinc}(x, y) = \max(x, y+1)$. Cet opérateur n'est pas idempotent ; nous l'employons donc sur un DAG[(b)]. Le problème du tri topologique consiste, de toute façon, à construire une numérotation isotone[102] qui n'existe que si le graphe est sans circuit.

Algorithme 8 TriTopologique(**pvar**⟨ **masque** ⟩ net_G)

pvar $p \leftarrow 1$ ▶ *p est une pvar qui vaut 1 partout.*
$p \leftarrow$ `maxinc-direct-association`(net_G, p)
retourner p ▶ *p est une numérotation isotone du réseau codé par* net$_G$.

102. Numérotation isotone : $\mathsf{ni}_G : V \to \mathbb{N}$ est une numérotation isotone du graphe $G(V, E)$ si, pour tout arc (u, v) de E, on a $\mathsf{ni}_G(u) < \mathsf{ni}_G(v)$.

(b) Dans [Gondran, 1975] (§ 2.6 page 88), la structure $(\mathbb{R} \cup \{-\infty\}, \max, +)$ est proposée pour rechercher les plus longs chemins. Cela n'est possible que si le graphe est sans circuit, car tout circuit (composé d'arcs valués à 1) ne serait pas 0-absorbant. De même, la non-idempotence du r-opérateur maxinc restreint son utilisation aux DAGs.

L'algorithme 8 fabrique une numérotation isotone qui n'est pas injective. Une numérotation isotone injective est obtenue par les éléments $(\mathsf{ni}_G[v], \imath[v])$ de $\mathbb{N}^2$, que l'on ordonne par $(a,b) < (c,d)$ si $a < c$ ou si $a = c$ et $b < d$. En effet, la numérotation isotone ni_G garantit la croissance, et la pvar index $\imath$ (qui est injective) garantit l'unicité des couples. Notons qu'il s'agit d'une pvar de «largeur double», alors que tout DAG admet une numérotation isotone avec les valeurs**(c)** $1, \ldots, |V|$.

Détection de DAGs. L'algorithme 8 termine uniquement si le réseau est sans circuit. Cette propriété peut être utilisée pour détecter les DAGs. Pour cela, il nous faut fabriquer un opérateur qui termine toujours, même en présence d'un circuit dans le réseau. On borne donc les résultats de l'opérateur maxinc par une constante C. On définit l'opérateur $\mathsf{minmaxinc}_C$ par**(d)** :

$$\mathsf{minmaxinc}_C(x,y) = \mathsf{min}(C, \mathsf{max}(x, y+1))$$

La constante C peut être différente d'un nœud à un autre, et on la note C_v sur le nœud v (opérateur de réseau). Elle doit être plus grande que le nombre d'antécédents du nœud v. Puisque la pvar index $\imath$ est injective, elle garantit en particulier que tous les antécédents d'un nœud ont une valeur différente. Aussi le plus grand numéro des antécédents est une borne sur le nombre d'antécédents du nœud, et $C_v = \mathsf{max}\left\{\imath[u], u \in \Gamma_G^-(v)\right\} + 1$ convient.

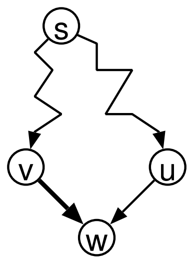


FIG. 75 – Un arc inverse (v,w) implique un chemin multiple ou un circuit avec la numérotation de la distance depuis l'unique source s de w.

Algorithme 9 Dag1(**pvar**⟨ **masque** ⟩ *net*$_G$)

pvar $C \leftarrow$ `max-direct-association`$(net_G, \imath)$ ▶ *Pvar des constantes de* $\mathsf{minmaxinc}_C$.
pvar $p \leftarrow 1$ ▶ *p vaut 1 partout.*
$p \leftarrow 1 + \mathsf{minmaxinc}_C$`-direct-association`$(net_G, p)$
partout où $p == C$ **faire**
  $p \leftarrow 0$ ▶ *Si* $p = C$ *quelque part, alors le graphe n'est pas un DAG.*
ailleurs
  $p \leftarrow 1$
fin partout où
retourner `Global-and`(p) ▶ *On retourne 1 si le graphe est un DAG, 0 sinon.*

Arcs inverses. Nous proposons maintenant un algorithme itératif, basé sur la recherche des arcs inverses. L'algorithme 10 est aussi utilisé pour la recherche d'un parcours en profondeur (*cf.* algorithme 14 page 132).

Définition 97 *Un arc* (u,v) *d'un graphe* $G(V,E)$*, muni d'une numérotation* f *quelconque des sommets, est appelé arc inverse si* $\mathsf{f}(u) > \mathsf{f}(v)$.

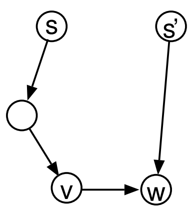


FIG. 76 – Si w a plusieurs sources dans ses antécédents, la proposition 98 est fausse.

L'algorithme 10 détecte les arcs inverses. La `step-exclusive-association` permet de récupérer la plus grande valeur des antécédents. Il ne reste plus qu'à effectuer une comparaison.

Algorithme 10 ArcInverse(**pvar**⟨ **masque** ⟩ *net*$_G$, **pvar** f)

pvar $r \leftarrow$ faux
partout où `max-step-exclusive-association`$(net_G, \mathsf{f}) > \mathsf{f}$ **faire**
  $r \leftarrow$ vrai
fin partout où
retourner r ▶ *r est vrai sur les sommets ayant un arc entrant inverse pour la numérotation* f.

On appelle *numérotation de la distance à la source* la numérotation d'un graphe obtenue en affectant à chaque nœud la plus petite distance le séparant d'une source logique.

Proposition 98 *Dans un graphe* $G(V,E)$ *admettant une unique source logique* s*, s'il y a un arc inverse pour la numérotation de la distance à la source, alors il y a un chemin multiple ou un circuit.*

(c) Il est aussi possible d'obtenir une numérotation injective avec une technique similaire à celle développée dans l'algorithme 13 page 131, qui fabrique une numérotation injective (numérotation du parcours en profondeur) à partir d'une numérotation non-injective (la distance à la source).

(d) Notons que l'opérateur $\mathsf{minmaxinc}_C$ est k-idempotent sur $\mathbb{N}$ (avec $k = C$), mais il est seulement k-idempotent faible sur $\mathbb{Z}$ (définitions 2 page 31 et 42 page 84).

Preuve. Soit (v, w) un arc inverse dans G. On a $\mathrm{d}_G(s, v) > \mathrm{d}_G(s, w)$ (figure 75 page précédente). Il existe un antécédent direct u de w tel que $\mathrm{d}_G(s, u) + 1 = \mathrm{d}_G(s, w)$. On a donc $u \neq v$. Si le chemin de s à v ne passe pas par u, alors il existe un chemin multiple : $s \rightsquigarrow v \to w$ et $s \rightsquigarrow u \to w$. S'il passe par u mais pas par w, alors il existe aussi un chemin multiple : $u \rightsquigarrow w$ et $u \rightsquigarrow v \to w$. Enfin, s'il passe par l'arc (u, w), alors le graphe présente un circuit : $w \rightsquigarrow v \to w$. □

La proposition 98 n'est pas vraie lorsque G possède plusieurs sources, comme l'indique la figure 76.

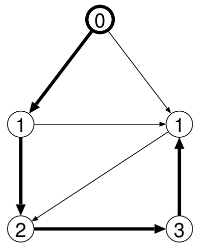


FIG. 77 – Graphe G numéroté avec la distance à la source logique ; T est obtenu en choisissant un plus grand antécédent (en gras). La source est indiquée en gras.

Proposition 99 *Un graphe $G(V, E)$ ayant un circuit admet un arc inverse avec la numérotation de la distance à la source.*

Preuve. Soit $(v_0, v_1), \ldots, (v_k, v_0)$ un circuit de G. Les sommets du circuit ont les mêmes ensembles d'antécédents ($\Gamma_G^-(v_i) = \Gamma_G^-(v_j)$, $0 \leq i, j \leq k$). En particulier, ils admettent les mêmes sources logiques. Soit L l'ensemble de leurs sources logiques. S'il n'y a pas d'arc inverse, on a $\mathsf{min}\{\mathrm{d}_G(s, v_0), s \in L\} < \mathsf{min}\{\mathrm{d}_G(s, v_1), s \in L\} < \ldots < \mathsf{min}\{\mathrm{d}_G(s, v_k)\} < \mathsf{min}\{\mathrm{d}_G(s, v_0), s \in L\}$, ce qui est impossible. □

Proposition 100 *Un graphe $G(V, E)$ de degré entrant $\delta_G^- = 1$ (i.e., au plus un seul arc entrant sur chaque nœud) possède un circuit si, et seulement si, G possède un arc inverse avec la numérotation de la distance à la source.*

Preuve. Tout circuit engendre un arc inverse avec la numérotation de la distance à la source (proposition 99). Supposons maintenant qu'il y ait un arc inverse dans G. Puisque $\delta_G^- = 1$, chaque sommet a une unique source logique dans ses antécédents. Dans ce cas, G possède un chemin multiple ou un circuit (proposition 98). Il ne peut s'agir que d'un circuit, car le degré entrant $\delta_v^- = 1$ interdit à tout sommet v d'être l'arrivée de plus d'un chemin. □

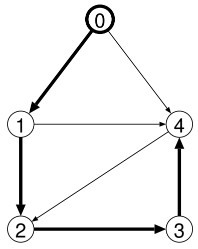


FIG. 78 – Numérotation de la distance à la source dans T.

En utilisant les propositions précédentes, on construit l'algorithme itératif 11 pour détecter la nature de graphe sans circuit (DAG) du réseau. Cet algorithme utilise le calcul de la distance, que nous présentons au § 7.4.1. Son principe est le suivant. Tout d'abord on recherche les sources logiques du réseau G ; on calcule ensuite la distance aux sources, et chaque nœud choisit un seul plus grand antécédent ; on obtient ainsi un sous-graphe T de G. Celui-ci est soit une forêt[103] couvrante T, enracinée aux sources logiques de G, soit un graphe ayant au moins un circuit. Pour savoir si T admet un circuit, on recherche ses sources logiques, et on calcule, pour chacun des nœuds de T, la distance à la source la plus proche. On détecte alors les éventuels arcs inverses, avec l'algorithme 10 page précédente. Si T présente un arc inverse, alors il admet un circuit, car le degré entrant de T est 1 (proposition 100). Dans ce cas, G admet aussi un circuit, et n'est donc pas un DAG. Mais l'absence de circuit dans T ne garantit pas l'absence de circuit dans G, comme l'indique la figure 77. Cela n'est vrai que si la distance calculée dans T ne remet pas en cause le choix du plus grand antécédent qui a servi à construire T. On met donc à jour le choix des antécédents, qui a conduit au sous-graphe T, avec la numérotation de la distance à la source calculée dans T, et on recommence (figures 78 et 79). La proposition 101 page suivante montre que ces itérations prennent fin.

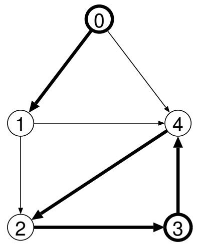


FIG. 79 – Mise à jour de T avec le nouvelle numérotation : choix d'un plus grand antécédent. Détermination des sources logiques dans T.

Algorithme 11 Dag2(**pvar**⟨ **masque** ⟩ net_G)

pvar⟨ **masque** ⟩ net_T ▶ *Pvar de masques de la «forêt» couvrante.*
pvar d ▶ *Note les distances dans la forêt courante.*
pvar $s \leftarrow$ SourcesLogiques(net_G) ▶ *0 sur les sources logiques (algorithme 6 page 127).*
pvar $d' \leftarrow$ Distance(net_G, s) ▶ *Utile pour la détection de la stabilité.*
pvar r ▶ *Pvar booléenne pour le résultat.*
répéter
    $d \leftarrow d'$
    $net_T \leftarrow$ choix d'un seul plus grand antécédent dans net_G, selon la pvar d.
    ▶ net_T *n'est pas forcément une forêt ; il pourrait contenir des circuits.*
    $s \leftarrow$ SourcesLogiques(net_T)
    $d \leftarrow$ Distance(net_T, s)

103. Forêt couvrante de G : ensemble d'arborescences disjointes, recouvrant le graphe G.

11 $r \leftarrow$ ArcInverse(net_T, d)
12 **si** `Global-or`(r) == vrai **alors**
13 $r \leftarrow$ faux ▶ *Il y a un circuit dans* net$_T$ *(proposition 100).*
14 **retourner** r
15 **fin si**
▶ *À ce stade,* net$_T$ *est une forêt enracinée aux sources logiques de* net$_G$.
16 $d' \leftarrow$ Distance(net_T,s)
17 **jusqu'à ce que** $d' = d$
18 $r \leftarrow$ vrai ▶ net$_G$ *est un DAG.*
19 **retourner** r

Proposition 101 *L'algorithme 11 page précédente détecte en $O(|V|)$ étapes globales si un graphe $G(V,E)$ est un DAG ou non.*

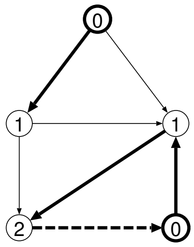


FIG. 80 – Calcul de la distance à la source la plus proche dans T. L'arc inverse (en pointillé) est détecté. G n'est donc pas un DAG.

Preuve. Montrons tout d'abord que l'algorithme termine. À chaque itération, on obtient à la ligne 15 une forêt couvrante de G. Les sources logiques de G sont aussi sources logiques dans T. Puisqu'il n'y a pas de circuit et que le degré entrant de T est 1, tout sommet a une et une seule de ces sources dans ses antécédents. Puisque T est un DAG, ses racines sont donc les sources logiques de G. Ainsi, à chaque étape, les racines de la forêt sont les sources logiques de G ; cela permet de comparer les distances calculées sur la forêt d'une itération à une autre. De par le choix effectué à la ligne 8, la distance calculée à la ligne 16 ne peut qu'augmenter ou rester stable d'une itération à une autre. Si elle reste partout stable, alors l'algorithme se termine (ligne 17). Mais cette distance est bornée par $|V|$, et l'algorithme se termine en au plus $|V|$ étapes, qui comportent un nombre constant d'`associations`[(e)].

Il reste à montrer que l'algorithme donne bien le bon résultat. C'est le cas lorsqu'il termine à la ligne 14 d'après la proposition 100). Lorsqu'il termine à la ligne 19, la numérotation de la pvar d est telle que, pour tout arc (u,w) de T, on a $d[u] < d[w]$ et, si u' est un autre antécédent de w qui n'a pas été choisi pour la construction de T, alors $d[u'] \leq d[u]$. On a donc, pour tout $v \in \Gamma_G^{-1}(w)$, $d[v] < d[w]$. La pvar d définit donc une numérotation isotone de G, ce qui n'est possible que si G est un DAG. □

Détection de graphe préfixe. Un graphe préfixe est un DAG qui n'a pas de double chemin (définition 4 page 32). L'intérêt d'un tel graphe est qu'il permet de retrouver le résultat d'une `association` utilisant un opérateur non-idempotent avec une seule `direct-association` (proposition 6 page 32). Lorsqu'un graphe a une seule source logique, il y a un seul chemin depuis cette source jusqu'à chaque sommet. L'addition sur le graphe doit donc redonner la distance à la source (ce n'est pas vrai si le graphe a plusieurs sources, *cf.* figure 34 page 33). Cela nous donne un moyen simple de vérifier si un graphe est préfixe (dans le cas d'une unique source). L'algorithme suivant utilise le calcul de la distance, présenté au 7.4.1.

Algorithme 12 GraphePréfixe(**pvar**$\langle$ **masque** $\rangle$ net_G, **pvar** s)

▶ *La pvar s indique la source (elle doit être nulle sur un seul sommet, et infinie ailleurs) ;* net$_G$ *doit être un DAG.*
pvar $r \leftarrow$ vrai ▶ *Pvar booléenne du résultat.*
pvar $a \leftarrow 1$
partout où `plus-direct-association`(net_G, a) $>$ Distance(net_G, s) **faire**
$r \leftarrow$ faux
fin partout où
$r \leftarrow$ `Global-and`(r)
retourner r ▶ *La pvar r est vraie partout si le graphe est préfixe.*

(e) Cette borne est largement surestimée, et en pratique (en moyenne) l'algorithme devrait terminer beaucoup plus rapidement. Il semble que les circuits sont, la plupart du temps, détectés avant δ_G^- étapes.

7.2.3 Arborescence en profondeur

Etat de l'art.* Le parcours en profondeur d'un graphe[(f)] est l'un des algorithmes de traversée les plus intéressants [Aho et al., 1974] [Gondran et Minoux, 1979] [Sedgewick, 1983]. Intuitivement, il s'agit de visiter tous les sommets du graphe en privilégiant toujours la découverte d'un nouveau sommet plutôt que de procéder à la visite systématique des descendants d'un nœud[(g)]. Nous l'utiliserons pour trouver les composantes fortement connexes d'un graphe. C'est aussi l'un des algorithmes les plus difficiles à mettre en œuvre de façon parallèle. Dans [Reif, 1985], l'auteur montre que ce parcours est intrinsèquement séquentiel, c'est-à-dire qu'il est illusoire de vouloir accélérer l'algorithme en organisant des activités simultanées à des endroits différents du réseau. La définition d'un algorithme de parcours en profondeur dans un contexte distribué a entrainé énormément de recherches [Cheung, 1983], [Awerbuch, 1985] [Zhu et Cheung, 1987] [Cidon, 1988] [Sharma et al., 1989]. Les variantes portent sur l'optimisation d'un paramètre, tel que le nombre de messages, au détriment d'un autre (complexité en temps ou taille des messages). Dans [Makki et Havas, 1996], les auteurs obtiennent un algorithme dont le nombre de messages et le nombre d'étapes sont $(1+r) \times |V|$, où r est un paramètre du réseau et de son routage compris entre 0 et 1.

Algorithme pour les réseaux associatifs. Un parcours en profondeur d'un graphe $G(V,E)$ est défini par la donnée d'une arborescence couvrante T, et d'une numérotation $\mathsf{n}_p\ : V \to \mathbb{N}$ des sommets de G, qui est injective, croissante sur les arcs[104] de T, et qui vérifie :

$$\forall u,v \in V, \quad v \in \Gamma^+_G(u) \Rightarrow \begin{cases} v \in \Gamma^+_T(u) \quad \text{et} \quad \mathsf{n}_p(u) < \mathsf{n}_p(v) \\ ou \\ v \notin \Gamma^+_T(u) \quad \text{et} \quad \mathsf{n}_p(u) > \mathsf{n}_p(v) \end{cases} \tag{7.1}$$

104. $\mathsf{n}_p(v) > \mathsf{n}_p(u)$, pour tout arc (u,v) de T.

Nous proposons une implémentation originale du parcours en profondeur pour les réseaux associatifs. L'arborescence du parcours est construite itérativement, et numérotée en profondeur à chaque étape. Cette numérotation permet de contrôler, à chaque étape, si l'arborescence et la numérotation constituent ou non un parcours en profondeur. Cette méthode ne s'applique que lorsque le graphe de départ a une seule racine (recherche d'une arborescence en profondeur et non d'une forêt).

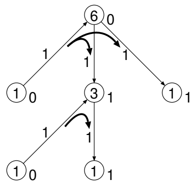


FIG. 81 – Première remontée-descente. La valeur dans le nœud indique son nombre de descendants. La valeur à côté du nœud est l'accumulation des descentes.

Numérotation de l'arborescence. L'algorithme 13 permet de numéroter une arborescence T en profondeur avec un nombre d'étapes élémentaires proportionnel à son degré sortant ($O(\delta^+_T)$). L'algorithme calcule tout d'abord le nombre de descendants de chaque nœud. Puis ce nombre est reporté, de branche en branche, par une remontée suivie d'une descente (figures 81 et 82). Chaque nœud connaît alors le nombre de sommets sur les branches le précédant dans la numérotation (*i.e.*, celles qui sont à sa gauche sur les figures 81, 82 et 83). Ajouté à la distance (*cf.* algorithme 19 page 138), cela donne une numérotation en profondeur de l'arborescence (*cf.* figure 83).

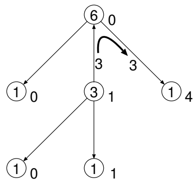


FIG. 82 – Seconde remontée-descente.

Algorithme 13 NumProfondeurArb(**pvar**⟨ **masque** ⟩ net_T)

pvar $s \leftarrow$ SourcesLogiques(net_T)
pvar $p_{down} \leftarrow 1$
$p_{down} \leftarrow$ `plus-direct-association`($net_T{}^{-1}, p_{down}$) ▶ *Donne le nombre de descendants.*
pvar⟨ **masque** ⟩ net_{up} ▶ *Sous-réseau pour les remontées.*
pvar⟨ **masque** ⟩ net_{down} ▶ *Sous-réseau pour les descentes.*
pvar $q_{up} \leftarrow 0$ ▶ *Pvar stockant les remontées.*
pvar $q_{down} \leftarrow 0$ ▶ *Pvar stockant les descentes.*
▶ *Boucle sur le degré sortant :*
pour $i = 1$ à $\delta^+_T - 1$ **faire**
  $net_{up} \leftarrow i^{\text{ème}}$ lien entrant de $net_T{}^{-1}$
  $net_{down} \leftarrow net_{down} - net_{up}{}^{-1}$
  ▶ *Le sous-réseau de la descente est composé des arcs non encore traités à la remontée.*

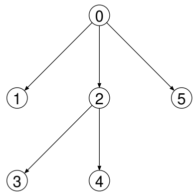


FIG. 83 – Addition des descentes à la distance : numérotation en profondeur de l'arborescence.

(f) Cet algorithme est aussi appelé algorithme de Trémaux ou de Tarry [Xuong, 1992].

(g) Dans [Eco, 1981] pp. 190-191, Guillaume de Baskerville explique à Adso de Melk comment utiliser cet algorithme pour sortir d'un labyrinthe. Il attribut cet algorithme à un «texte antique».

$q_{up} \leftarrow$ `plus-step-exclusive-association`(net_{up}, p_{down}) ▶ *Remontée.*

$q_{down} \leftarrow q_{down}$ + `plus-direct-association`(net_{down}, q_{up}) ▶ *Descente.*

fin pour

▶ *Accumulation des descentes :*

$q_{down} \leftarrow$ `plus-association`(net_T, q_{down})

▶ *La numérotation en profondeur* n_p *est obtenue en ajoutant à l'accumulation des descentes* q_{down} *la distance plus 1 (car les distances commencent à 0 et le parcours en profondeur à 1).*

pvar $d \leftarrow$ Distance(net_T, s)

pvar $\mathsf{n}_p \leftarrow d + q_{down} + 1$

retourner $(\mathsf{n}_p, q_{down}, p_{down}, d)$ ▶ q_{down}, p_{down} *et* d *sont utilisées dans l'algorithme 16.*

Obtention de l'arborescence. Certains graphes G admettent des arborescences couvrantes T pour lesquelles il n'existe aucune numérotation en profondeur dans T qui soit aussi une numérotation en profondeur dans G (*cf.* figure 84). Nous proposons donc un algorithme itératif pour obtenir une numérotation en profondeur d'un graphe. Cet algorithme part d'une arborescence couvrante quelconque, la numérote en profondeur, et la modifie tant qu'il existe des arcs mettant en défaut la numérotation obtenue sur T dans G, telle que définie par l'équation 7.1 page précédente. Il est nécessaire que G possède une seule racine (T est une arborescence et non une forêt).

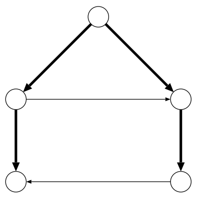


FIG. 84 – Il n'existe pas de numérotation en profondeur de T (en gras) qui soit aussi numérotation en profondeur de G.

Algorithme 14 NumProfondeur(**pvar**$\langle$ **masque** $\rangle$ net_G, **pvar** r)

▶ *r vaut 0 sur un seul nœud, qui est la racine de l'arborescence recherchée.*

1 **pvar**$\langle$ **masque** $\rangle$ $net_T \leftarrow$ ForêtCouvrante(net_T, r) ▶ *Algorithme 20 page 138.*

2 **pvar** n_1 ▶ *Numérotation des nœuds.*

3 **pvar** n_2 ▶ *Plus grand numéro des descendants.*

4 **pvar** q_{down} ▶ *Pvar renvoyée par l'algorithme NumProfondeurArb.*

5 **pvar** p_{down} ▶ *Idem.*

6 **pvar** d ▶ *Idem (ces pvars sont utilisées par l'algorithme 16 page 134).*

7 **pvar** *stable* ▶ *Détection de la stabilité.*

8 **répéter**

9 $(n_1, q_{down}, p_{down}, d) \leftarrow$ NumProfondeurArb(net_T) ▶ *Algorithme 13 page précédente.*

10 $n_2 \leftarrow$ `max-direct-association`$({net_T}^{-1}, n_1)$

11 *stable* $\leftarrow$ vrai

12 **partout où** ArcInverse$(net_G - net_T, n_1, n_2)$ **faire**

▶ *(cf. algorithme 10 page 128)*

13 net_T est mis à jour en choisissant un plus petit antécédent direct dans $net_G - net_T$.

14 *stable* $\leftarrow$ faux

15 **fin partout où**

16 **jusqu'à ce que** `Global-or(`*stable*`)` == vrai

17 **retourner** $(n_1, q_{down}, p_{down}, d)$ ▶ q_{down}, p_{down} *et* d *sont utilisées dans l'algorithme 16.*

Proposition 102 *Pour tout graphe G, l'algorithme 14 construit une arborescence couvrante T de G et une numérotation en profondeur de T, qui est aussi numérotation en profondeur de G, en au plus $h(T) \times \delta_G^+$ étapes, où $h(T)$ désigne la hauteur de T.*

Preuve. À chaque étape de l'algorithme, l'arborescence T est modifiée (ligne 13). Supposons que le sommet v choisisse le sommet u' comme antécédent direct dans l'arborescence plutôt que le sommet u. Cette modification de T entraîne des modifications dans la numérotation des descendants de v. Mais elle est sans conséquence pour les antécédents de u et u'. Ainsi, les sommets à distance 1 de la source sont sûrs après l'étape 1, ceux à distance 2 après l'étape 2, et ainsi de suite. Tous les sommets sont donc sûrs à l'étape $j \leq h(T)$.

Par construction, la numérotation n_1 correspond au parcours en profondeur de G selon l'arborescence T (*cf.* équation 7.1 page précédente). □

7.2.4 Composantes connexes et fortement connexes

La recherche des composantes fortement connexes est nécessaire pour obtenir le graphe réduit. Celui-ci permet, dans certains cas, de calculer une `association` avec des `direct-associations` (*cf.* § 3.3.4 page 42). Le graphe réduit est aussi utile pour calculer une `direct-association` en partant des CFCs sources du graphe réduit (*cf.* 8.3.2 page 162), après qu'il ait été ordonné par le tri topologique (algorithme 8 page 127). Nous commençons par le problème plus simple des composantes connexes. Puis nous proposons un algorithme basé sur le parcours en profondeur pour trouver les CFCs. Pour finir, nous donnons un algorithme immédiat mais utilisant des pvars de listes, et un autre n'utilisant ni parcours en profondeur ni pvar de listes, mais nécessitant des itérations.

Composantes connexes. Trouver les composantes connexes ne pose pas de difficulté majeure si on admet les hypothèses de l'index et du graphe inverse (hypothèses 10 page 36 et 93 page 125). L'algorithme suivant diffuse, dans chaque composante, le plus petit index des sommets la composant. L'injection de la numérotation $\imath$ (*i.e.*, unicité des index) assure que chaque composante héritera d'un numéro unique.

Algorithme 15 CC(**pvar**⟨ **masque** ⟩ net_G)

pvar $n \leftarrow$ `min-direct-association`$(net_G \cup net_G{}^{-1}, \imath)$
retourner n

Composantes fortement connexes : état de l'art.* Le principal algorithme de recherche des composantes fortement connexes est dû à Trémaux en 1882 ; il a été reformulé sous une forme plus moderne par Tarjan en 1972 [Gondran et Minoux, 1979]. C'est une adaptation de cet algorithme qui est utilisée dans ANET (*cf.* § 8.3.2 page 162). Il utilise un parcours en profondeur du graphe, et une pile, dans laquelle les sommets sont stockés dès qu'ils sont rencontrés [Aho et al., 1974] [Sedgewick, 1983]. Dans [Nuutila et Soisalon-Soininen, 1994], les auteurs proposent d'empiler les sommets uniquement lorsque tous leurs descendants ont été visités, ce qui évite d'empiler pour tout de suite dépiler des sommets qui sont, à eux seuls, composante fortement connexe. Dans [Jiang, 1993], une *pile adaptative* est utilisée : l'empilement d'un sommet supprime les autres copies du sommet déjà présentes dans la pile. Le parcours du graphe n'est donc pas en largeur d'abord, ni en profondeur d'abord. Dans [Gondran et Minoux, 1979], une formulation non-récursive est proposée. Toutes ces variantes de l'algorithme de Trémaux-Tarjan sont difficilement adaptables au cas d'un algorithme de réseau.

Dans [Aho et al., 1983], les auteurs donnent un algorithme utilisant deux parcours en profondeur, fabriquant chacun deux numérotations. La formulation suivante est de [Cormen et al., 1994]. La première traversée en profondeur du graphe G utilise une variable k initialisée à 1 au départ, et fabrique les numérotations n_1^1 et n_2^1 comme suit (*cf.* figures 85 et 86). Lors de la première arrivée sur un sommet u, $\mathsf{n}_1^1[u]$ prend la valeur de k, et k est incrémentée. Lorsqu'il est nécessaire de rebrousser chemin pour trouver une branche non encore explorée, on positionne $\mathsf{n}_2^1[v]$ à k si tous les descendants de v ont été explorés ; k est ensuite incrémentée. Le second parcours en profondeur est semblable au précédent, sauf qu'il est effectué sur le graphe inverse G^{-1}, en partant du sommet ayant obtenu la valeur maximale de n_2^1. Si ce second parcours donne une forêt, alors les racines r_i sont toujours choisies en prenant celle qui a la plus grande valeur de n_2^1 parmi les sommets restants. À la fin de ce second parcours, les composantes fortement connexes sont les ensembles de sommets u vérifiant $\mathsf{n}_1^2(r_i) \leq \mathsf{n}_1^2(u) < \mathsf{n}_2^2(u) \leq \mathsf{n}_2^2(r_i)$, pour chaque racine r_i.

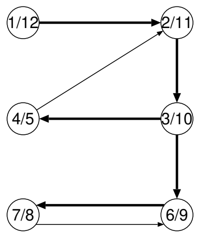


FIG. 85 – Premier parcours en profondeur sur G.

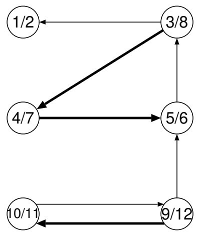


FIG. 86 – Second parcours en profondeur sur G^{-1}.

Adaptation aux réseaux associatifs. Nous proposons une adaptation de ce second algorithme aux réseaux associatifs. Pour cela, on remarque que la numérotation n_2^1 est *pseudo-isotone* (voir ci-dessous), et que c'est cela qui permet de trouver les composantes fortement connexes (le second parcours a un rôle de diffusion de l'information de n_2^1). Nous construisons donc cette numérotation, en utilisant les informations renvoyées par l'algorithme 14 de parcours en profondeur.

Définition 103 *Une numérotation injective* n *d'un graphe* G *est pseudo-isotone si la numérotation* $\widetilde{\mathsf{n}}$, *obtenue en prenant le maximum des valeurs de* n *sur chaque composante connexe de* G, *donne une numérotation isotone du graphe réduit* $\widetilde{G}$ *(cf. définition 17 page 42).*

Proposition 104 *La numérotation* n_2^1 *est une numérotation pseudo-isotone de* G^{-1}.

Preuve. Soit u le premier sommet de la CFC $\widetilde{u}$ rencontré lors du (premier) parcours en profondeur, et v celui de la CFC $\widetilde{v}$, telle que $\widetilde{v}$ soit un descendant de $\widetilde{u}$ dans $\widetilde{G}$.

Supposons que u ait été visité avant v lors du premier parcours. Dans ce cas, puisque v est un descendant de u dans G, v est un descendant de u dans l'arborescence en profondeur. Donc $\mathsf{n}_2^1[v]$ sera attribué avant $\mathsf{n}_2^1[u]$, et on aura

$$\mathsf{n}_2^1[v] = \mathsf{max}\{\mathsf{n}_2^1[w], w \in \widetilde{v}\} < \mathsf{max}\{\mathsf{n}_2^1[w], w \in \widetilde{u}\} = \mathsf{n}_2^1[u] \tag{7.2}$$

Maintenant, supposons que v ait été visité avant u. Dans ce cas, puisque u n'est pas un descendant de v, le parcours revient au sommet v avant d'atteindre le sommet u ; l'équation 7.2 est donc vérifiée.

La numérotation construite en prenant les maximums de n_2^1 sur chaque composante connexe est donc isotone sur $\widetilde{G^{-1}}$; n_2^1 est donc pseudo-isotone sur G^{-1}. □

Proposition 105 *Si la pvar p est une numérotation pseudo-isotone sur le sous-réseau G codé par la pvar de masques* net_G*, alors les composantes fortement connexes de G sont obtenues avec une* max-*direct-association*(net_G, p) *: deux sommets obtiendront la même valeur si et seulement s'ils appartiennent à la même composante fortement connexe.*

Preuve. La `direct-association` diffuse le maximum de chaque composante fortement connexe sur tous les sommets de la composante, ainsi que sur toutes les composantes descendantes. La numérotation isotone assure que chaque maximum ne sera pas diffusé plus loin que sa propre composante. □

Proposition 106 *La numérotation n_2^1 appliquée au graphe $G(V, E)$ vérifie, pour tout $v \in V$,*

$$\mathsf{n}_2^1[v] = \mathrm{d}_T(r, v) + 2 \times \left(|\Gamma_T^+(v)| + q_{\mathrm{down}} + 1\right) \tag{7.3}$$

où q_{down} est la pvar de l'accumulation des descentes fabriquée par l'algorithme 13 page 131.

Preuve. On a tout d'abord $\mathsf{n}_2^1[v] = \mathsf{n}_1^1[v] + 2 \times |\Gamma_G^-(v)| + 1$, pour tout sommet v du graphe G. En effet, après avoir affecté $\mathsf{n}_1^1[v]$, la variable de parcours k est incrémentée une fois, puis ensuite deux fois par descendant w de v (lors de l'affectation de $\mathsf{n}_1^1[w]$ et de $\mathsf{n}_2^1[w]$).

Ensuite, on a $\mathsf{n}_1^1[v] = \mathrm{d}_T(r, v) + 2 \times q_{down} + 1$, où r est la racine de l'arborescence de T. En effet, on a $\mathsf{n}_p[v] = \mathrm{d}_T(r_v, v) + q_{down} + 1$, où n_p est la numérotation en profondeur classique — obtenue lorsque k n'est incrémentée que lors des descentes — (*cf.* algorithme 13 page 131). On a donc $\mathsf{n}_p[v] - \mathrm{d}_T(r_v, v) - 1 = q_{down}$, et q_{down} est la correction à apporter à la numérotation de la distance à la racine pour obtenir une numérotation en profondeur. Pour un sommet v, cette correction est liée au nombre d'antécédents de v dans l'arborescence (*cf.* figures 81, 82 et 83 page 131). La numérotation n_1^1 est semblable à la numérotation en profondeur classique n_p sauf qu'à l'arrivée en v, deux valeurs ont été attribuée à chacun des antécédents de v dans l'arborescence. La correction à apporter par rapport à la distance est donc double.

En combinant les deux égalités, on obtient l'équation 7.3. □

Algorithme 16 CFC1(**pvar**⟨ **masque** ⟩ net_G)

pvar n_p
pvar q_{down}
pvar p_{down}
pvar d
pvar $s \leftarrow$ Source(net_G) ▶ *Algorithme 6 page 127.*
$(\mathsf{n}_p, q_{down}, p_{down}, d) \leftarrow$ NumProfondeur(net_G, s) ▶ *Algorithme 14 page 132.*
pvar $p \leftarrow d + 2 \times (q_{down} + p_{down} + 1)$
`max-direct-association`$({net_G}^{-1}, p)$
pvar⟨ **masque** ⟩ $net_{CFC} \leftarrow$ `link-with-equals(`net_G`,` p`,` p`)`
retourner net_{CFC}

Algorithme avec une pvar de liste. L'algorithme 17 permet de trouver les composantes fortement connexes en deux `direct-associations`, mais il utilise des pvars de listes (plus précisément, d'ensembles, représentés par des listes). Celles-ci peuvent contenir, dans le pire des cas, autant d'éléments que de nœuds dans le réseau. Les CFCs sont obtenues en construisant, sur chaque nœud, l'intersection des ensembles d'antécédents et de descendants. On utilise pour cela les s-opérateurs $\cup$ et $\cap$, respectivement d'union, et d'intersection d'ensemble. Les sommets ayant les mêmes ensembles d'antécédents et de descendants font partie de la même composante fortement connexe ; le sous-réseau représentant les CFCs est construit avec la primitive `link-with-equals`.

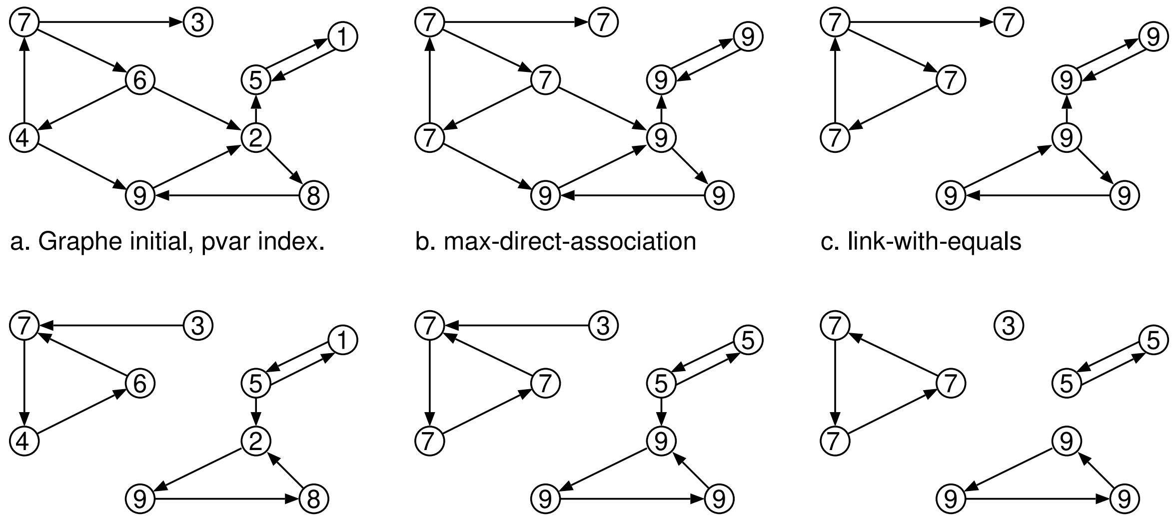
Algorithme 17 CFC2(**pvar**⟨ **masque** ⟩ net_G)

pvar⟨ **masque** ⟩ net_{CFC}
pvar $p \leftarrow \{\imath\}$
pvar $q \leftarrow \{\imath\}$
$p \leftarrow$ $\cup$-`direct-association`(net_G, p)
$q \leftarrow$ $\cup$-`direct-association`(${net_G}^{-1}, q$)
pvar⟨ **masque** ⟩ $net_{CFC} \leftarrow$ `link-with-equals`(net_{CFC}, $p \cap q$, $p \cap q$)
retourner net_{CFC}

Algorithme itératif. L'algorithme 18 utilise, pour un réseau de $|V|$ nœuds, des pvars de $\log_2(|V|)$ (*i.e.*, largeur de la pvar index $\imath$). Son principe est comparable au tri à bulle, présenté page 11 : les composantes sont isolées en fonction du plus grand index des nœuds les composant. Il nécessite donc, dans le pire cas, un nombre d'`associations` égal à la longueur d'un plus grand chemin dans le graphe réduit. La figure 87 illustre une itération.

Algorithme 18 CFC3(**pvar**⟨ **masque** ⟩ net_G)

pvar⟨ **masque** ⟩ $net_{CFC} \leftarrow net_G$
pvar⟨ **masque** ⟩ net_{tmp}
pvar p
répéter
  $net_{tmp} \leftarrow net_{CFC}$
  $p \leftarrow$ max-`direct-association`($net_{CFC}, \imath$) ▶ *$\imath$ est la pvar des index.*
  $net_{CFC} \leftarrow$ `link-with-equals`(net_{CFC}, p, p)
  $p \leftarrow$ max-`direct-association`(${net_{CFC}}^{-1}, \imath$)
  $net_{CFC} \leftarrow$ `link-with-equals`(net_{CFC}, p, p)
jusqu'à ce que $net_{CFC} \neq net_{tmp}$
retourner net_{CFC}



a. Graphe initial, pvar index. b. max-direct-association c. link-with-equals

d. Graphe inversé, pvar index. e. max-direct-association f. link-with-equals

FIG. 87 – *Une itération de l'algorithme 18.*

7.3 Algorithmes dirigés par les données

Nous illustrons ici une application du modèle et de la `direct-association` pour décrire et réaliser des calculs sur le graphe de précédence d'un algorithme. Cette application montre que tout algorithme peut, en théorie, être modélisé par une `direct-association` sur un réseau et avec des opérateurs adéquats. En outre, les calculs exprimés sur les graphes de précédence font ressortir un parallélisme maximal, et, si le support d'exécution peut l'exploiter, de bonnes performances peuvent être attendues (cas des transformées de Fourier dans un réseau de processeurs en forme de papillon par exemple).

7.3.1 Présentation

Partant d'un algorithme présentant ou non du parallélisme, on peut exprimer l'enchaînement des calculs (ou, plus généralement, des tâches) à réaliser sous forme d'un *graphe de précédence*, dont les arcs représentent les relations de précédence entre les tâches. Lorsque l'algorithme est complètement déroulé, il ne présente pas de boucles ; c'est un DAG dont la forme peut dépendre des données. Il s'agit en fait de remplacer les «étapes temporelles» par des «étapes spatiales». Un algorithme ainsi décrit peut être exécuté en une seule `direct-association` sur ce graphe, avec, en chaque nœud, un opérateur réalisant la tâche correspondante. Lorsque, sur chaque nœud, les calculs sont effectués dès que possible, on est assuré que le temps de calcul minimal sera obtenu. De plus, un effet de pipeline peut être obtenu sur certaines applications, telles que le calcul de la transformée de Fourier ou de la programmation dynamique. Ce principe est utilisé par les machines dirigées par les données[(h)] .

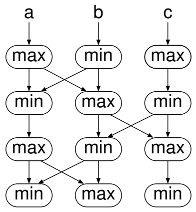


FIG. 88 – Tri de valeurs en une `direct-association`.

La figure 36 page 34 donne le réseau permettant de calculer les coefficients du binôme de Newton. Le réseau de la figure 88 permet, quant à lui, de trier en une phase globale les données en entrée du réseau (tri pair-impair).

7.3.2 Programmation dynamique*

Afin d'illustrer le calcul sur un graphe de précédence, nous appliquons la méthode de la programmation dynamique au problème du *Knapsack*, qui peut être formulé comme suit. Un voyageur s'aperçoit que les N bagages qu'il a préparés ne peuvent être acceptés dans l'avion, car le poids total par passager est limité à L kg. Il doit donc choisir les bagages qu'il emmène, et ceux qu'il abandonne à l'aéroport. Plus un bagage est volumineux, plus il contient d'affaires ; le voyageur cherche donc le sous-ensemble des N bagages qui a un volume maximal pour un poids n'excédant pas la limite de L kg. On note w_i le poids du $i^{\text{ème}}$ bagage, v_i son volume, et x_i sa présence dans le sous-ensemble des bagages emenés : x_i vaut 0 (absence) ou 1 (présence). Le voyageur cherche à résoudre le problème K_L^N qui consiste à trouver les x_i tels que $\sum_{i=1}^{N} w_i \cdot x_i \leq L$ et $\sum_{i=1}^{N} v_i \cdot x_i$ est maximal. On note $V_N(L)$ le volume idéal du sous-ensemble des n bagages (volume maximum pour un poids inférieur à L kg).

La méthode de la programmation dynamique consiste à décomposer le problème K_L^N en $(N+1) \times (L+1)$ petits problèmes K_l^n, avec $0 \leq n \leq N$ et $0 \leq l \leq L$. Par exemple, le problème K_1^1 correspond à un bagage, et une limite de 1 kg. Si le poids w_1 de ce bagage est inférieur à 1 kg, alors le voyageur emmène l'objet ($x_1 = 1$), sinon il le laisse ($x_1 = 0$). On a

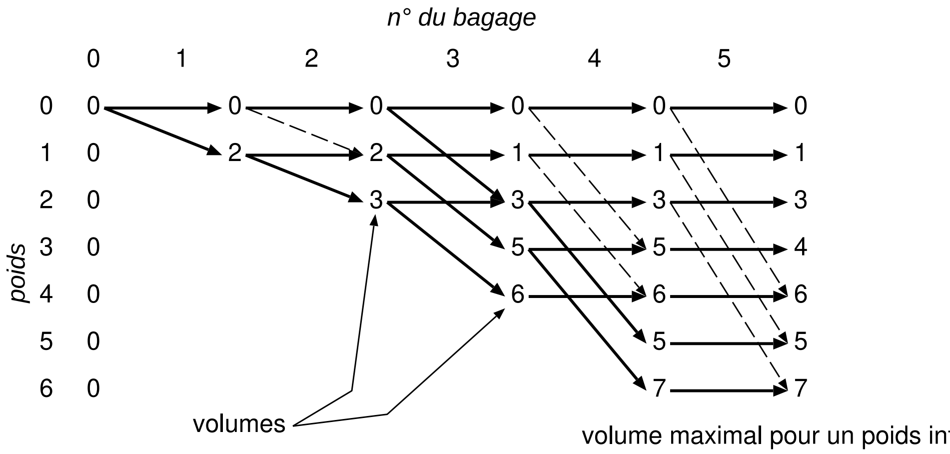
(h) Voir à ce sujet la note **(f)** page 34.

$$
\begin{aligned}
V_{n+1}(l) &= \max_{(x_1,\dots,x_{n+1})\in\{0,1\}^{n+1}} \left(\sum_{i=1}^{n+1} v_i \cdot x_i,\ \text{tel que} \sum_{i=1}^{n+1} w_i \cdot x_i \le l\right) \\
&= \max\Bigg(\max_{(x_1,\dots,x_n,0)\in\{0,1\}^{n+1}} \left(\sum_{i=1}^{n+1} v_i \cdot x_i,\ \text{tel que} \sum_{i=1}^{n+1} w_i \cdot x_i \le l\right), \\
&\qquad \max_{(x_1,\dots,x_n,1)\in\{0,1\}^{n+1}} \left(\sum_{i=1}^{n+1} v_i \cdot x_i,\ \text{tel que} \sum_{i=1}^{n+1} w_i \cdot x_i \le l\right) \Bigg) \\
&= \max\Bigg(\max_{(x_1,\dots,x_n)\in\{0,1\}^{n}} \left(\sum_{i=1}^{n} v_i \cdot x_i,\ \text{tel que} \sum_{i=1}^{n} w_i \cdot x_i \le l\right), \\
&\qquad \max_{(x_1,\dots,x_n)\in\{0,1\}^{n}} \left(\sum_{i=1}^{n} v_i \cdot x_i,\ \text{tel que} \sum_{i=1}^{n} w_i \cdot x_i \le l - w_{n+1}\right) + v_{n+1} \Bigg)
\end{aligned}
$$

On obtient donc la formule de récurrence $V_{n+1}(l) = \max\left(V_n(l), V_n(l - w_{n+1}) + v_{n+1}\right)$ dont les valeurs initiales sont $\forall l, 0 \le l \le L, V_0(l) = 0$ et $\forall n, 0 \le n \le N, V_n(0) = 0$. Cette récurrence peut s'exprimer par un graphe de précédence, tel que celui de la figure 89

7.3.3 Programmation dynamique et réseaux associatifs

L'opération réalisée par chaque nœud de la colonne i du DAG est donnée par le r-opérateur binaire maxinc_v défini par $(x, y) \mapsto \mathsf{max}(x, y+v_i)$. Ce r-opérateur n'est pas idempotent, mais le réseau est un graphe de précédence, qui est acyclique. Les calculs du graphe de précédence sont donc décrits avec une seule maxinc_v-`direct-association`. Notons que de nombreux problèmes d'optimisation combinatoire peuvent être formulés ainsi.



bagages

n°	poids	volume
0	0	0
1	1	2
2	1	1
3	2	3
4	3	2
5	4	3

FIG. 89 – *Exemple de graphe de précédence des récurrences de la programmation dynamique. Une flèche horizontale dans la $i^{ème}$ colonne signifie que le bagage numéro i n'est pas pris, tandis qu'une flèche oblique signifie qu'il est pris. Quand un objet est pris, le poids et le volume total sont augmentés. Quand un choix est possible, c'est le plus grand volume qui l'emporte (flèche pleine).*

7.4 Application des r-opérateurs à l'algorithmique de graphe

Dans cette section, nous donnons plusieurs applications des r-opérateurs, définis au chapitre 5. Certains de ces algorithmes sont auto-stabilisants (*cf.* chapitre 6).

7.4.1 Forêt des chemins de poids minimaux

Distance. Nous avons défini, à la proposition 47 page 86, le r-opérateur minc sur $\overline{\mathbb{N}}$ par $\mathsf{minc}(x,y) = \mathsf{min}(x, y+1)$. Cet opérateur est idempotent ; la `minc-direct-association` termine donc sur tout réseau (théorème 59 page 90) et donne le résultat

$$b_{\text{out}}[w] = \bigoplus_{v \in \overline{\Gamma_G^-}(w)\}} \left(\mathrm{d}_G(v,w) + b^0_{\text{in}}[v]\right)$$

Ainsi, si un nœud *racine* r (qui est un antécédent de tous les sommets), est initialisé à 0, et les autres nœuds à $+\infty$, l'opérateur minc permet de calculer la distance à r, en une `direct-association`. De même, si plusieurs racines sont initialisées à 0, chaque nœud finira avec la distance à la racine la plus proche.

Algorithme 19 Distance(**pvar**⟨ **masque** ⟩ *net*$_G$, **pvar** *racine*)

▶ *La pvar* racine *est nulle sur les racines, infinie ailleurs.*
pvar $d \leftarrow$ `minc-direct-association`(*net*$_G$, *racine*)
retourner d ▶ *La pvar d est égale à la distance depuis la racine la plus proche.*

Forêt couvrante. Une fois connue la distance à la racine la plus proche (pvar d), si chaque nœud v choisit un antécédent u tel que $d[u] < d[v]$ et $d[u] = \mathsf{min}(d[u'], u' \in \Gamma_G^{-1}(v))$ — il y en a toujours un, sauf pour les racines —, alors il n'y a pas d'arc inverse (définition 97 page 128). Suivant la proposition 100 page 129, le sous-réseau ainsi obtenu est un DAG, et donc une forêt puisque le degré entrant est 1.

Algorithme 20 ForêtCouvrante(**pvar**⟨ **masque** ⟩ *net*$_G$, **pvar** *racine*)

▶ *La pvar racine est nulle sur les racines, infinie ailleurs.*
pvar⟨ **masque** ⟩ *net*$_T \leftarrow 0$ ▶ *Sous-réseau ayant aucun arc.*
pvar $d \leftarrow$ Distance(*net*$_G$, *racine*) ▶ *Algorithme 19.*
La pvar de masques *net*$_T$ est initialisée en ne prenant qu'un seul antécédent, s'il existe, qui a une valeur plus petite que celle du nœud et au plus égale à toutes celles des autres antécédents.
retourner *net*$_T$ ▶ net$_T$ *est le sous-réseau codant la forêt.*

Lorsqu'il n'y a qu'une seule racine r (et que tout sommet v admet r dans ses antécédents), la forêt est, par construction, l'arborescence des plus courts chemins, aussi appelé *arbre en largeur*, du nom du parcours qui permet habituellement de l'obtenir. Lorsqu'il y a plusieurs racines (et que tout sommet en a au moins une dans ses antécédents), chaque nœud choisit l'une des racines les plus proches. L'algorithme 20 résout donc, en une `direct-association` suivie d'une opération locale, l'intéressant problème de la recherche de la forêt des plus courts chemins.

Chemins de poids minimum. Lorsque les arcs du graphe $G(V,E)$ sont pondérés (il existe une fonction $w : E \to \mathbb{S}$), le problème de la recherche de l'arborescence de poids minimum[105] nécessite un r-opérateur n-aire. En effet, l'opération à réaliser au niveau de chaque nœud consiste en un minimum des valeurs des antécédents augmentées par le poids des arcs. On utilise donc l'opérateur minc_w, défini sur $\overline{\mathbb{N}}$ ou $\overline{\mathbb{R}}$ par l'équation 5.1 page 92, que nous écrivons sous une forme adaptée à notre problème : sur chaque nœud v, l'opérateur minc_w est défini par[106] :

$$\mathsf{minc}_w\left(x_0, \ldots, x_{\delta_v^-}\right) = \mathsf{min}\left(x_0, x_1 + w\big(\,(\mathbf{i}_v^{-1}(1), v)\,\big), \ldots, x_{\delta_v^-} + w\big(\,(\mathbf{i}_v^{-1}(\delta_v^-), v)\,\big)\right) \tag{7.4}$$

Il s'agit d'un r-opérateur n-aire idempotent ; la minc_w`-direct-association` termine donc, et donne le résultat (*cf.* théorème 69 page 96) :

$$b_{\text{out}}[v] = \bigoplus_{u \in \overline{\Gamma_G^-}(v)} \mathsf{r}_{P_{u \to v}}\left(b^0_{\text{in}}[u]\right)$$

Les r-fonctions de chemin r_P sont les compositions des fonctions $x \mapsto x + w((u,v))$, et correspondent à la somme des poids des chemins. La `direct-association` donne donc le poids du

105. En anglais, *Single source shortest path problem.*

106. L'application $\mathbf{i}$ de $\Gamma_G^-(v)$ dans $\{1, \ldots, \delta_v^-\}$ associe à l'antécédent u l'indice $\mathbf{i}(u)$ de l'arc entrant (u,v) sur v (*cf.* § 3.2.1 page 36). Sa réciproque $\mathbf{i}^{-1}$ donne l'antécédent correspondant à un numéro donné.

chemin de poids minimal depuis la racine. Il ne reste plus qu'à choisir, comme pour l'algorithme 20 page ci-contre, un plus petit antécédent pour résoudre le problème de l'arborescence de poids minimum. Notons que s'il y a plusieurs racine, le même algorithme résout le problème de la forêt couvrante de poids minimum[107], qui a de nombreuses applications (*cf.* § 7.5 page 142).

107. En anglais, *Multiple sources shortest paths.*

Algorithme 21 FCPoidsMin(**pvar**⟨ **masque** ⟩ net_G, **pvar** *racine*, **pvar** w)

▶ *La pvar* racine *est nulle sur les racines, infinie ailleurs.* w *est utilisée par l'opérateur* minc_w.
pvar⟨ **masque** ⟩ $net_T \leftarrow 0$ ▶ *Sous-réseau n'ayant aucun arc.*
pvar $d_w \leftarrow \mathsf{minc}_w$-`direct-association`($net_G$, *racine*) ▶ d_w *est la distance pondérée.*
La pvar de masques net_T est initialisée en ne prenant qu'un seul antécédent, s'il existe, qui a une valeur plus petite que celle du nœud et au plus égale à toutes celles des autres antécédents.
retourner net_T ▶ net_T *est le sous-réseau codant la forêt des poids minimums.*

7.4.2 Forêt des chemins de fiabilité maximale

Dans un réseau de communications où certains terminaux doivent choisir leur centre de retransmission, la distance n'est pas toujours le meilleur critère, qu'elle soit calculée avec le nombre de nœuds intermédiaires (avec le r-opérateur minc) ou avec la somme des longueurs des arcs (avec le r-opérateur minc_w). En effet, le critère de fiabilité des communications peut être tout aussi important. Dans un réseau associatif, le problème devient la recherche de la forêt couvrante qui permet à chaque nœud v de recevoir l'information la plus fiable possible depuis une racine. On suppose connu le taux d'erreur $\pi(\,(u,v)\,)$, obtenu lors des retransmissions sur chaque lien (u,v) du réseau ($0 < \pi(\,(u,v)\,) < 1$ [(i)]). Le taux de transmission fiable est donc $\overline{\pi}(\,(u,v)\,) = 1 - \pi(\,(u,v)\,)$. On définit alors, pour chaque nœud v, l'opérateur $\mathsf{maxmul}_{\overline{\pi}}$ sur l'intervalle $[0,1]$ de $\mathbb{R}$ par :

$$\mathsf{maxmul}_{\overline{\pi}}\left(x_0,\ldots,x_{\delta_v^-}\right) = \max\left(x_0, x_1 \times \overline{\pi}\big(\,(\mathbf{i}_v^{-1}(1),v)\,\big),\ldots,x_{\delta_v^-} \times \overline{\pi}\big(\,(\mathbf{i}_v^{-1}(\delta_v^-),v)\,\big)\right)$$

Il s'agit d'un r-opérateur n-aire basé sur le s-opérateur max, et dont les r-fonctions $r_v^i(x) = x \times \overline{\pi}(\,(\mathbf{i}_v^{-1}(i),u)\,)$ vérifient $x \prec_{\mathsf{max}} r_v^i(x)$ et maxmul est (strictement) idempotent[108]. La maxmul-`direct-association` termine donc sur tout réseau et donne le résultat (*cf.* théorème 69 page 96) :

108. L'opérateur max définit une relation d'ordre totale $\preceq_{\mathsf{max}}$ sur $[0,1] \subset \mathbb{R}$ qui est la relation $\geq$ habituelle. Les taux étant bornés strictement par 1, les r-fonctions sont strictement croissantes pour $\preceq_{\mathsf{max}}$ (*i.e.*, strictement décroissantes pour $<$). Le r-opérateur est donc strictement idempotent (définition 78 page 109.

$$b_{\text{out}}[v] = \bigoplus_{u \in \overline{\Gamma_G^-}(v)} \mathsf{r}_{P_{u\to v}}\left(b_{\text{in}}^0[u]\right)$$

Les r-fonctions de chemin r_P correspondent au produit des fiabilités $\overline{\pi}$ de chaque arc du chemin ; la `direct-association` donne donc la fiabilité maximale de tous les chemins en provenance des racines. Il ne reste plus qu'à choisir le meilleur antécédent pour obtenir la forêt des chemins de fiabilité maximale. L'algorithme est semblable au précédent.

Algorithme 22 FCFiabilitéMax(**pvar**⟨ **masque** ⟩ net_G, **pvar** *racine*, **pvar** $\overline{\pi}$)

▶ *La pvar* racine *vaut 1 sur les racines, 0 ailleurs. La pvar* $\overline{\pi}$ *code les fiabilités ; elle est utilisée par l'opérateur* $\mathsf{maxmul}_{\overline{\pi}}$.
pvar⟨ **masque** ⟩ $net_T \leftarrow 0$ ▶ *Sous-réseau ayant aucun arc.*
pvar $f_{\mathsf{max}} \leftarrow \mathsf{maxmul}_{\overline{\pi}}$-`direct-association`($net_G$, *racine*)
▶ f_{max} *est la meilleure fiabilité depuis une racine antécédente.*
net_T est initialisé en ne prennant qu'un seul antécédent, s'il existe, qui a une valeur plus grande que celle du nœud et au moins égale à toutes celles des autres antécédents.
retourner net_T ▶ net_T *est le sous-réseau codant la forêt des fiabilités maximales.*

(i) Nous supposons, pour simplifier l'exposé, que la connexité du réseau est préservée, d'où $\pi(\,(u,v)\,) < 1$. De même, pour assurer la stricte idempotence du r-opérateur maxmul, nous supposons que toute communication admet un léger taux d'erreur, qui peut, cependant être très proche de 0 ($0,0\ldots01$). D'où $0 < \pi(\,(u,v)\,)$.

7.4.3 Forêt des chemins de capacité maximale

Outre la distance (physique ou en nombre de nœuds), et la fiabilité, un autre critère important dans le choix d'un centre de retransmission est le débit qu'il est possible de recevoir depuis ce nœud. Plus généralement, on parlera de *capacité* du lien ; il pourrait s'agir, par exemple, de la disponibilité du canal de communication. On suppose que les capacités sont des réels ; la fonction κ de E dans $\mathbb{R}$ associe, à tout arc (u, v) du graphe $G(V, E)$, la capacité $\kappa(\,(u, v)\,)$. Pour trouver la forêt de diffusion construite avec les chemins de capacité maximale, on définit, pour chaque nœud v du réseau, un opérateur n-aire de $\mathbb{R}^{\delta_v^-}$ dans $\mathbb{R}$ par :

$$\mathsf{maxmin}_\kappa\left(x_0, \ldots, x_{\delta_v^-}\right) = \mathsf{max}\Big(x_0, \mathsf{min}\left(x_1, \kappa(\,(\mathbf{i}_v^{-1}(1), v)\,)\right), \ldots, \mathsf{min}\left(x_{\delta_v^-}, \kappa(\,(\mathbf{i}_v^{-1}(\delta_v^-), v)\,)\right)\Big)$$

109. Il n'est pas strictement idempotent ; il suffit de considérer $\mathsf{r}_i\left(\kappa(\,(\mathbf{i}_v^{-1}(i), v)\,)\right)$.

Il s'agit d'un r-opérateur n-aire dont les r-fonctions $\mathsf{r}_i\ : x \mapsto \mathsf{min}\left(x_i, \kappa(\,(\mathbf{i}_v^{-1}(i), v)\,)\right)$ vérifient $\mathsf{r}_i(x) \leq x$. L'opérateur maxmin_κ est donc idempotent[109]. La maxmin_κ-`direct-association` termine donc sur tout réseau. Les r-fonctions de chemin donnent la plus petite capacité des arcs le composant, c'est-à-dire la capacité du chemin. L'algorithme précédent, instancié avec cet opérateur, fournit donc la forêt des chemins de capacité minimale :

Algorithme 23 FCCapacitéMax(**pvar**⟨ **masque** ⟩ net_G, **pvar** *racine*, **pvar** $\overline{\kappa}$)

▶ *La pvar* racine *vaut 1 sur les racines, 0 ailleurs. La pvar κ code les capacités ; elle est utilisée par l'opérateur* maxmin_κ.
1 **pvar**⟨ **masque** ⟩ $net_T \leftarrow 0$ ▶ *Sous-réseau ayant aucun arc.*
2 **pvar** $f_{\mathsf{max}} \leftarrow \mathsf{maxmin}_\kappa$-`direct-association`($net_G$, *racine*)
▶ f_{max} *est la meilleure capacité de tous les chemins depuis une racine antécédente.*
3 La pvar de masques net_T est initialisée en ne prenant qu'un seul antécédent, s'il existe, qui a une valeur plus grande que celle du nœud et au moins égale à toutes celles des autres antécédents.
4 **retourner** net_T ▶ net_T *est le sous-réseau codant la forêt des capacités maximales.*

7.4.4 Arborescence en profondeur avec un r-opérateur

Nous proposons un r-opérateur travaillant sur les listes et bâti avec une relation d'ordre non numérique. Il permet d'obtenir une arborescence en profondeur en une `direct-association`. On considère l'ensemble des listes de mots, écrits sur un alphabet quelconque. L'ordre lexicographique de l'alphabet est étendu en un ordre sur les mots, puis sur les listes, lues de gauche à droite. L'ensemble $\mathbb{S}$ est donc totalement ordonné. On note $\oplus$ le s-opérateur qui donne la plus petite liste d'un ensemble de listes. On suppose que chaque nœud détient une liste initiale d'un mot unique (ce pourrait être la liste contenant son index $\{\imath\}$). La pvar de ces listes est notée l. Par exemple, on aura (*pauillac*, *margaux*, *pomerol*) $\oplus$ (*pomerol*, *cahors*) $=$ (*pauillac*, *margaux*, *pomerol*), où les mots sont des noms de vin ici.

Pour chaque nœud du réseau, on définit une application r de $\mathbb{S}$ dans $\mathbb{S}$, qui concatène la liste initiale du nœud aux listes reçues des antécédents directs. Par exemple, sur le nœud détenant la liste initiale $\{$*sancerre*$\}$, on a $\mathsf{r}(\{$*saumur*, *chinon*$\}) = \{$*saumur*, *chinon*, *sancerre*$\}$. On définit alors l'opérateur binaire $\mathsf{lexicat}_l$ sur $\mathbb{S}$ par :

$$\mathsf{lexicat}_l\,(l_1, l_2) = l_1 \oplus \mathsf{r}(l_2)$$

Par exemple, sur le nœud détenant la liste initiale *gamay*, et recevant les listes $\{$*pommard*, *julienas*$\}$ et $\{$*julienas*, *bourgueil*$\}$, on a :

$$\begin{aligned}&\mathsf{lexicat}_l\big(\{\textit{pommard}, \textit{julienas}\}, \{\textit{julienas}, \textit{bourgueil}\}\,\big) =\\ &\qquad\{\textit{pommard}, \textit{julienas}\} \oplus \{\textit{julienas}, \textit{bourgueil}, \textit{gamay}\} =\\ &\qquad\{\textit{julienas}, \textit{bourgueil}, \textit{gamay}\}\end{aligned}$$

L'application r n'est pas bijective et $\mathsf{lexicat}_l$ n'est donc pas un r-opérateur binaire. Mais étendu en opérateur n-aire, il s'agit d'un r-opérateur généralisé[(j)]. De plus, la r-fonction r vérifie $l \preceq_{\oplus} \mathsf{r}(l)$ pour toute liste $l \in \mathbb{S}$ et lexicat est idempotent. La $\mathsf{lexicat}_l$-`direct-association` termine donc sur tout réseau. La proposition suivante montre que l'algorithme 24 fournit une arborescence et une numérotation en profondeur du réseau.

Proposition 107 *Si le réseau est initialisé avec une pvar de liste l, telle que chaque nœud v du réseau $G(V, E)$ détienne une liste unique $l[v] = \{\imath[\mathtt{self}]\}$ (i.e., la liste contenant son index), et si le nœud s détenant la plus petite liste (au sens de l'ordre lexicographique étendu aux listes) est une source principale du réseau, alors la $\mathsf{lexicat}_l$-`direct-association` suivie du choix du plus petit antécédent fournit une numérotation n_p et une arborescence en profondeur enracinée en s. De plus, la liste détenue par chaque nœud permet de retrouver le chemin depuis la source s jusqu'au nœud.*

Preuve. La $\mathsf{lexicat}_l$-`direct-association` termine sur tout réseau car $\mathsf{lexicat}_l$ étendu à un opérateur de n variables est un r-opérateur généralisé idempotent (*cf.* théorème 76 page 99). Le sous-réseau obtenu par le choix d'un plus petit antécédent est une arborescence enracinée en s. En effet, toutes les listes détenues par les nœuds commencent par le mot m_s. Elles ont donc été construites à la suite de la réception d'une liste plus petite, en provenance d'un antécédent, sauf pour s qui n'a pas de plus petit antécédent. Tous les sommets, sauf s, ont donc un plus petit antécédent. Le choix d'un tel antécédent assure que le graphe résultant est une arborescence couvrante enracinée en s. En effet, il n'y a pas d'arc inverse, et puisque le degré entrant est 1, c'est un DAG (proposition 100 page 129), et donc une arborescence (car aucun chemin n'est l'arrivée d'un double chemin, sinon il serait de degré entrant 2). Soit T_s l'arborescence couvrante enracinée en s.

110. La concaténation de deux listes est notée par $\cdot$.

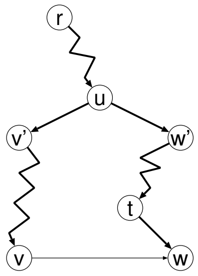


FIG. 90 – Si (v, w) n'est pas dans l'arborescence T_r (en gras), alors $\mathsf{n}_r[v] > \mathsf{n}_r[w]$.

Supposons que n_p ne soit pas une numérotation en profondeur sur T_s. Puisque toutes les valeurs de la pvar n_p sont différentes, et qu'elle est croissante le long des arcs de T_s, il reste à examiner le cas d'un arc (v, w) de G qui vérifierait $\mathsf{n}_p[v] < \mathsf{n}_p[w]$ et $w \notin T_v$. Dans ce cas, on aurait $w \notin T_v$ et $v \notin T_w$. Il existerait donc un sommet u dans l'arborescence T_s antécédent de v et w mais différent de l'un et de l'autre. Soit v' et w' les descendants de u tels que v' soit dans la branche conduisant à v (figure 90) et w' dans celle conduisant à w. Soit aussi t l'antécédent direct de w dans T_s. On a[110] $\mathsf{n}_p[v] = \mathsf{n}_p[u] \cdot \mathsf{n}[v'] \cdots v$, $\mathsf{n}_p[w] = \mathsf{n}_p[u] \cdot \mathsf{n}[w'] \cdots t \cdot w$. Puisque $\mathsf{n}_p[v] < \mathsf{n}_p[w]$, on a $\mathsf{n}_p[v'] < \mathsf{n}_p[w']$. Dans ce cas, le nœud w aurait choisi l'antécédent direct v plutôt que t, car $\mathsf{n}_p[v] < \mathsf{n}_p[t]$, ce qui est en contradiction avec le fait que t soit l'antécédent de w dans l'arborescence.

Enfin, chaque valeur de la numérotation n_p est une liste contenant les valeurs initiales des nœuds composant le parcours. Elle permet donc de retrouver le chemin depuis s (l'identification du chemin est facilitée lorsque la liste initiale l contient les index : $l = \{\imath\}$). $\square$

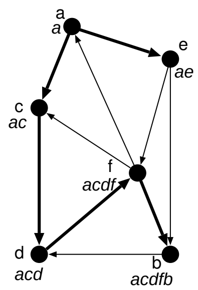


FIG. 91 – Application de l'algorithme 24. Les valeurs initiales sont des lettres ; la numérotation est donnée par les mots (en italique).

Algorithme 24 ArbProfondeur(**pvar**$\langle$ **masque** $\rangle$ net_G, **pvar**$\langle$ **liste** $\rangle$ l)

▶ *l est une pvar de liste, dont chaque composante est unique ; la plus petite valeur est détenue sur un nœud s qui est source principale du réseau.*

pvar$\langle$ **masque** $\rangle$ $net_{T_s} \leftarrow 0$ ▶ *Sous-réseau ayant aucun arc.*

pvar $\mathsf{n}_p \leftarrow \mathsf{lexicat}_l$-`direct-association`$(net_G, l)$

La pvar de masques net_{T_s} est initialisée en ne prenant qu'un seul antécédent, s'il existe, qui a une valeur plus grande que celle du nœud et plus grande ou égale que toutes celles des autres antécédents.

▶ n_p *est une numérotation en profondeur sur l'arborescence T_s, enracinée en s.*

retourner $(net_{T_s}, \mathsf{n}_p)$

(j) Il est cependant possible de définir une bijection pour ce problème. En effet, il est suffisant de considérer l'ensemble $\mathbb{S}$ des listes ayant au plus $|V|$ éléments et de définir l'application r depuis l'ensemble des listes ayant au plus $|V| - 1$ éléments dans l'ensemble des listes ayant au moins un élément et au plus $|V|$. Dans ce cas, la concaténation r est une bijection. Pour simplifier l'exposé, nous préférons utiliser le résultat plus faible des r-opérateurs généralisés obtenu au § 5.4.2 page 98. Pour cela, l'opérateur binaire lexicat est étendu aux opérateurs n-aires : $\mathsf{lexicat}(x_0, \ldots, x_{n-1}) = x_0 \oplus \mathsf{r}(x_1) \oplus \cdots \oplus \mathsf{r}(x_{n-1})$.

7.4.5 Calcul du diamètre

Nous proposons maintenant un opérateur manipulant des pvars encore plus larges, mais qui fournit une information encore plus riche. Soit $\mathbb{S}$ l'ensemble des listes d'ensembles d'index. Par exemple, si a, b, c et d sont des index des nœuds, alors la liste $(\{d\},\{b\},\{a,c\})$ est un élément de $\mathbb{S}$. On identifie toute liste $(S_1,\ldots,S_k)$ à celle obtenue en lui concaténant des ensembles vides[(k)] $(S_1,\ldots,S_k) = (S_1,\ldots,S_k,\emptyset,\ldots,\emptyset)$. On considère l'opérateur $\oplus$ sur $\mathbb{S}$ qui fusionne terme à terme les éléments (ensembles) des listes de $\mathbb{S}$, et qui supprime les éléments qui apparaissent préalablement dans les ensembles de la liste. Par exemple :

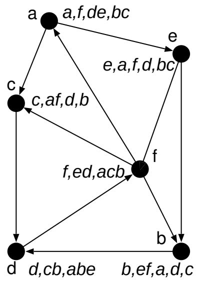


FIG. 92 – Application de l'opérateur Ant. Les index sont des lettres. La pvar de liste d'ensembles résultat est représentée en italique.

$$
\begin{aligned}
&(\{d\},\{b\},\{a,c\}) \oplus (\{c\},\{a,e\},\{b\}) = \\
&(\{d,c\},\{b,a,e\},\{\cancel{a},\cancel{c},\cancel{b}\}) = \\
&(\{d,c\},\{b,a,e\})
\end{aligned}
$$

On montre sans difficulté qu'il s'agit d'un s-opérateur. On considère l'application r de $\mathbb{S}$ dans $\mathbb{S}$ qui, à la liste $l = \{S_1,\ldots,S_k\}$, associe la liste $\mathsf{r}(l) = \{\emptyset, S_1,\ldots,S_k\}$. Par exemple :

$$\mathsf{r}\,(\,(\{d\},\{b\},\{a,c\})\,) = (\,\{\emptyset,\{d\},\{b\},\{a,c\}\}\,)$$

Comme pour l'exemple précédent, r n'est pas une bijection. L'opérateur Ant, défini avec le s-opérateur $\oplus$ et la r-fonction r par $\mathsf{Ant}(l_1,l_2) = l_1 \oplus \mathsf{r}(l_2)$, n'est donc pas un r-opérateur binaire. Mais étendu en un opérateur n-aire, on vérifie qu'il s'agit d'un r-opérateur généralisé : il préserve sa valeur initiale et r est un homomorphisme de $\mathbb{S}$ (*cf.* définition 72 page 98). Il est idempotent car $(S_1,\ldots,S_k) \preceq_\oplus \mathsf{r}(\,S_1,\ldots S_k)\,) \equiv (S_1,\ldots,S_k) \oplus (\emptyset,S_1,\ldots,S_k) = (S_1,\ldots,S_k)$. La `Ant-direct-association` termine donc sur tout réseau. Puisque la r-fonction r est une application d'une seule variable, le résultat est identique à celui fourni par le théorème 69 page 96 (*cf.* § 5.4.2 page 98). Quand chaque nœud v est initialisé avec la liste $(\{v\})$, alors le résultat sur v est :

$$b_{\text{out}}[v] \quad = \quad \bigoplus \left\{ r^{\mathrm{d}_G(u,v)}(\{u\}), u \in \Gamma_G^-(v) \right\}$$

Il s'agit d'une complète représentation des antécédents de v, sous forme d'une liste, ordonnée par la distance à v (figure 92).

Cette information très riche peut, par exemple, être utilisée pour calculer le diamètre. Pour cela, il suffit de diffuser la plus grande longueur de liste sur le réseau symétrisé.

Algorithme 25 Diamètre(**pvar**$\langle$ **masque** $\rangle$ net_G)

pvar$\langle$ **liste d'ensemble** $\rangle$ $l \leftarrow (\,\{\,\imath\,\}\,)$ ▶ *Sur le nœud v, $l[v] = (\,\{\,\imath[v]\,\}\,) = (\,\{\,v\,\}\,)$.*
$l \leftarrow$ `Ant-direct-association`(net_G, l)
pvar $d \leftarrow$ longueur de la liste l ▶ *Calcul local sur chaque nœud.*
$d \leftarrow$ `max-direct-association`$(net_G \cup net_G{}^{-1}, d)$
retourner d ▶ *d est égale, sur chaque nœud, au diamètre de G.*

7.5 Algorithmique géométrique

Les algorithmes géométriques[111] sont souvent considérés comme des briques de base, utilisés dans des constructions plus complexes, en traitement d'images, conception assistée par ordinateur, robotique, etc. Nous illustrons les applications de notre modèle et des r-opérateurs à un sous-ensemble de ce vaste domaine, dont une présentation détaillée pourra être trouvée dans [Preparata et Shamos, 1985].

111. En anglais, *computational geometry algorithms.*

112. Distance Euclidienne : Dans le plan $\mathbb{R}^2$, $d(a,b) = \left((x_a - x_b)^2 + (y_a - y_b)^2\right)^{1/2}$.

(k) Plus formellement, on peut définir une relation d'équivalence $\mathcal{R}$ telles que l et l' soient dans la même classe d'équivalence, et travailler dans l'ensemble quotient $\mathbb{S}/\mathcal{R}$.

7.5.1 Métrique

Distance euclidienne et approximations. L'algorithmique géométrique nécessite la définition d'un espace normé ; la notion de distance y est donc primordiale. La topologie est fortement influencée par le choix de la distance. On considère généralement la distance Euclidienne[112] mais son coût de calcul est élevé. Des approximations de la distance Euclidienne ont donc été proposées, pour les applications de traitement d'images.

La *distance de Manhattan*[113] d_4 est définie par $d_4(p,q) = |x_p - x_q| + |y_p - y_q|$ (p et q sont des points de coordonnées respectives dans le plan (x_p, y_p) et (x_q, y_q)). La *distance de l'échiquier*[114] est définie par $d_8(p,q) = \mathsf{max}\left(|x_p - x_q|, |y_p - y_q|\right)$. Des approximations plus complexes, telles que la somme pondérée de la distance d'un chemin purement diagonal et de celle d'un chemin vertical ou horizontal ont aussi été proposées (figure 93). La longueur du chemin diagonal est donnée par $\mathsf{min}\left(|x_p - x_q|, |y_p - y_q|\right)$; celle du chemin vertical ou horizontale par $\mathsf{max}\left(|x_p - x_q|, |y_p - y_q|\right) - \mathsf{min}\left(|x_p - x_q|, |y_p - y_q|\right))$. La *distance pondérée* entre le point p et le point q est donc $d(p,q) = a \times \mathsf{min}\left(|x_p - x_q|, |y_p - y_q|\right) + b \times \left(\mathsf{max}\left(|x_p - x_q|, |y_p - y_q|\right) - \mathsf{min}\left(|x_p - x_q|, |y_p - y_q|\right)\right)$. Les pondérations a et b prennent, par exemple, les valeurs entières 3 et 2, ou 7 et 5 [Borgefors, 1986]. Dans le premier cas, on trouve une estimation du double de la distance Euclidienne ($b = 2$) et dans l'autre du quintuple ($b = 5$).

113. Aussi appelée *city bloc*.

114. En anglais, *chess board metric*.

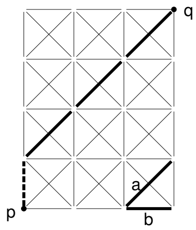


FIG. 93 – Distance pondérée par a pour les chemins diagonaux, et par b pour les chemins horizontaux ou verticaux (hachuré).

Mise en œuvre avec les r-opérateurs. Comme au § 7.4.1 page 138, nous utilisons le r-opérateur binaire minc, et ses dérivés n-aires minc_w, pour résoudre le calcul de la distance.

Si le plan discrétisé est modélisé par une grille à deux dimensions, alors une seule `minc-association` donne la distance d_4. La figure 94 représente l'image — calculée avec la bibliothèque ANET (*cf.* chapitre 8) — obtenue avec une seule `minc-direct-association` sur un réseau en forme de grille sans diagonale, et sur une image initialisée à 0 au centre, et 255 ailleurs (l'élément neutre de minc sur les pvars de 8 bits). Si maintenant le plan est modélisé par une grille avec diagonales, alors une seule `minc-direct-association` permet de calculer la distance d_8 ; on obtient ainsi l'image 95.

Pour la distance pondérée, on utilise l'opérateur minc_w (*cf.* équation 7.4 page 138) : elle est calculée en une seule minc_w`-direct-association` sur la grille pondérée avec a sur les diagonales et b sur les autres arcs.

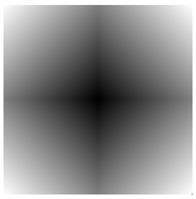


FIG. 94 – d_4 calculé avec une `minc-direct-association` sur une grille en 4-connexité.

7.5.2 Polygones de Voronoï

État de l'art.* Les régions de Voronoï ont été définies dans un contexte de géométrie algébrique, par Voronoï en 1908 [Joly, 1990]. Ces régions forment une partition du plan en régions connexes centrées sur des *graines*, de telle sorte que tout point se retrouve dans la région de la graine la plus proche. Les polygones sont les frontières des régions (figure 96 page suivante). Nous avons déjà évoqué l'intérêt des régions de Voronoï pour le traitement d'images au chapitre 2. En fait, la recherche des régions ou des polygones de Voronoï est un problème clé de l'algorithmique géométrique, qui a énormément d'applications [Preparata et Shamos, 1985]. Nous revenons sur certaines d'entre elles ci-dessous.

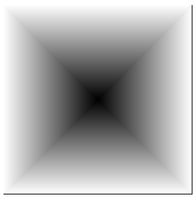


FIG. 95 – d_8 calculée avec une `minc-direct-association` sur une grille en 8-connexité.

Pour construire les régions de Voronoï, chaque point de l'espace doit «choisir» une des graines, parmi celles qui sont les plus proches de lui. Les polygones de Voronoï sont généralement construits de façon récursive, en partitionnant le plan. Dans [Dehne et Klein, 1994], les auteurs proposent de construire les polygones en faisant évoluer dynamiquement le lieu géométrique des points à égale distance des graines et d'une droite qui se déplace sur le plan. Dans [Du, 1998], l'auteur propose un algorithme basé sur le calcul des angles.

Implémentation dans les réseaux associatifs. Nous proposons ici un algorithme plus simple, utilisant pleinement l'asynchronisme de la `direct-association`. Les régions grandissent à partir des graines, petit à petit, jusqu'à ce qu'elles recouvrent tout le plan. L'asynchronisme du calcul global assure que la croissance des régions se fait «à la même vitesse» d'une région à une autre. Plus exactement, l'asynchronisme agit comme une relaxation, qui remet en cause, au niveau local, les choix de la meilleur graine, jusqu'à la complète stabilisation du réseau. La stabilité — et la fin du calcul global — n'intervient que lorsque les régions ont équitablement partitionné l'espace.

Les algorithmes que nous donnons ici utilisent la métrique d_4, c'est-à-dire que la grille G, qui modélise l'espace discret, n'a pas de diagonale (*cf.* § 7.5.1 page précédente). Le programme que nous avons écrit avec la bibliothèque ANET, et qui est présenté en annexe au § A.2 page 187, utilise la métrique d_8 (et donc une grille avec diagonale). La grille avec diagonale n'est pas planaire. Aussi, lors de la construction de la forêt, il faut sélectionner en priorité le plus petit antécédent le long des arcs horizontaux ou verticaux, afin d'éviter que des régions se chevauchent par les diagonales. Pour utiliser la distance pondérée, il faut prendre l'opérateur $\mathtt{minc}_w$ à la place de `minc`, avec les poids appropriés.

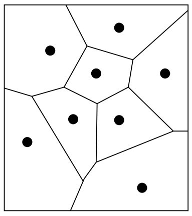


FIG. 96 – Polygones de Voronoï.

L'algorithme 26 fournit un sous-réseau codant les régions de Voronoï. Il ne nécessite qu'une seule `direct-association`. Cet algorithme améliore sensiblement la version itérative à base de s-opérateurs, que nous avons décrite au § 2.2.2 page 24. Il est utilisé par l'algorithme 27, qui construit le sous-réseau des polygones de Voronoï en reliant correctement les sommets ayant un arc dans le graphe frontière (*i.e.*, $net_{frontière} \neq 0$). La figure 97 donne les régions de Voronoï correspondant à la figure 96, avec la métrique d_8. Comme nous l'avons signalé, le choix de la métrique influence la topologie, et donc la forme des régions de Voronoï[(1)] (voir aussi les formes des régions de Voronoï avec la métrique d_8 présentées à l'annexe § A.3 page 188). Enfin, la figure 98 donne les polygones de Voronoï, tels qu'obtenus par l'algorithme 27.

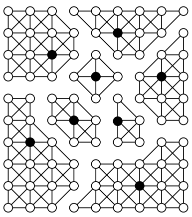


FIG. 97 – Sous-réseau des régions de Voronoï.

Algorithme 26 RégionsVoronoï(**pvar**⟨ **masque** ⟩ net_G, **pvar** g)

▶ net_G *code la grille sans diagonale ; g vaut 0 sur les graines,* $+\infty$ *ailleurs.*
pvar $d \leftarrow$ `minc-direct-association`(net_G, g)
pvar⟨ **masque** ⟩ $net_{régions} \leftarrow$ choix d'un plut petit antécédent s'il existe.
▶ $\mathrm{net}_{\mathrm{régions}}$ *est une forêt orientée des feuilles vers les graines.*
partout où $g = 0$ **faire**
  $d \leftarrow \imath$ ▶ *d prend l'index des graines sur les graines,*
ailleurs
  $d \leftarrow 0$ ▶ *et 0 ailleurs.*
fin partout où
$d \leftarrow$ `max-association`$({net_{régions}}^{-1}, \imath)$ ▶ *Diffusion de l'index de la graine à la région.*
$net_{régions} \leftarrow$ `link-with-equals`(net_G, d, d) ▶ *Graphe des régions de Voronoï.*
retourner $net_{régions}$

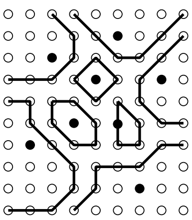


FIG. 98 – Sous-réseau des polygones de Voronoï.

Algorithme 27 PolygoneVoronoï(**pvar**⟨ **masque** ⟩ net_G, **pvar** g)

▶ net_G *code la grille sans diagonale ; g vaut 0 sur les graines,* $+\infty$ *ailleurs.*
pvar⟨ **masque** ⟩ $net_{régions} \leftarrow$ RégionsVoronoï(net_G, g)
pvar⟨ **masque** ⟩ $net_{frontière} \leftarrow$ complémentaire de $net_{régions}$
▶ $\mathrm{net}_{\mathrm{frontière}}$ *représente toutes les connexions entre régions voisines.*
pvar $p \leftarrow 0$
partout où $net_{frontière} \neq 0$ **faire**
  $p \leftarrow 1$ ▶ *p vaut 1 sur les points des polygones.*
fin partout où
$net_{frontière} \leftarrow$ `link-with-equals`(net_G, p, p)
▶ $\mathrm{net}_{\mathrm{frontière}}$ *inclut maintenant les arcs formant les polygones.*
$net_{frontière} \leftarrow net_{frontière} \cap net_G$
▶ $\mathrm{net}_{\mathrm{frontière}}$ *inclut maintenant uniquement les arcs formant les polygones.*
retourner $net_{frontière}$

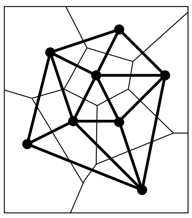


FIG. 99 – Triangulation à l'aide des régions de Voronoï.

7.5.3 Triangulation

Dans [Preparata et Shamos, 1985], les auteurs étudient le problème de la triangulation (problème P.4 p. 183). La *triangulation* d'un ensemble de points du plan est donnée par des lignes droites reliant ces points, telles qu'aucune d'entre elles ne se chevauchent, et telles que toutes les régions internes, formées par ces lignes, sont des triangles. La relation entre les régions de Voronoï et la triangulation

(1) Pour la figure 97, la finesse de la discrétisation du plan intervient aussi.

a été montrée par Delaunay en 1934. Le problème est appelé *triangulation de Delaunay* dans [Du, 1998]. Il a été adapté aux maillages utilisés par les éléments finis par Hermeline en 1980 [Joly, 1990].

On définit le *dual*[115] des polygones de Voronoï comme étant le graphe obtenu en reliant les graines par des lignes si et seulement si elles partagent un segment des polygones. Delaunay a montré que ce dual est une triangulation des graines (figure 99 page ci-contre).

115. En anglais, *straight-line dual.*

Dans le modèle des réseaux associatifs, pour obtenir le dual des régions de Voronoï, et donc la triangulation des graines, il suffit de construire le graphe virtuel associé (*cf.* § 2.2.1 page 22). Le graphe des sommets du graphe virtuel est constitué des composantes connexes du graphe des régions (c'est-à-dire les régions de Voronoï, *cf.* figure 97 page ci-contre). Le graphe des arcs est constitué des arcs du graphe frontière, obtenu en complémentant le graphe des régions (comme dans l'algorithme 27 page précédente).

7.5.4 Enveloppe convexe

Étant donné des points du plan, le problème de l'enveloppe convexe consiste à trouver le polygone convexe englobant tous les points (figure 100) [Preparata et Shamos, 1985]. Ce problème a, tout comme les polygones de Voronoï, de nombreuses applications. Citons, par exemple, la recherche du domaine englobant des données statistiques bi-dimensionnelles, en statistique. L'enveloppe convexe est généralement construite récursivement. Dans [Blelloch, 1990b], l'auteur présente un algorithme à base de *scans*, appelé *Quickhull* car il ressemble au tri rapide : recherche d'un pivot, séparation des données en deux ensembles, et application récursive sur les deux sous-ensembles. Nous proposons un algorithme utilisant un nombre constant de `direct-associations`.

Lorsque la portion du plan considérée est suffisament grande, les régions de Voronoï centrées sur les graines de l'enveloppe convexe sont les seules à toucher les bords de la portion (c'est le cas sur la figure 100). Cela permet de retrouver les régions extérieures, et donc d'identifier les points de l'enveloppe, qui sont les graines de ces régions. L'algorithme 28 fournit les points de l'enveloppe convexe. Pour ensuite obtenir l'enveloppe en elle-même (*i.e.*, les liens entre les points de l'enveloppe), on pourra faire appel au graphe virtuel (*cf.* § 7.5.3). Notons que si la portion du plan considérée n'est pas assez grande, l'algorithme donne tout de même une bonne approximation de l'enveloppe.

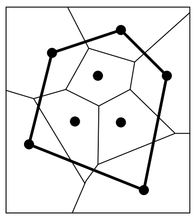


FIG. 100 – Enveloppe convexe.

Algorithme 28 PointsEnvConvexe(**pvar**⟨ **masque** ⟩ net_G, **pvar** g)

▶ net_G *est la pvar de masques de la grille sans diagonale modélisant le plan discrétisé ; g vaut 0 sur les points à envelopper,* $+\infty$ *ailleurs.*

pvar⟨ **masque** ⟩ $net_{regions} \leftarrow$ RégionsVoronoi(net_G, g)
pvar p
partout où $net_G \neq 1111$ **faire**
  $p \leftarrow +\infty$
  ▶ *Seuls les points des bords de la grille n'ont pas un masque complet (*1111 *car* $\delta_G^- = 4$*).*
ailleurs
  $p \leftarrow 0$
fin partout où
$p \leftarrow$ `max-direct-association`($net_{régions}, p$)
▶ *Seules les points des régions extérieures ont obtenus* $+\infty$.
partout où $p \neq +\infty$ **faire**
  $g \leftarrow +\infty$ ▶ *Suppression des graines des régions internes.*
fin partout où
retourner g ▶ $g = 0$ *sur les points de l'enveloppe,* $+\infty$ *ailleurs.*

7.5.5 Conclusion

Nous avons montré comment utiliser la `direct-association` pour résoudre certains algorithmes géométriques, dont le problème central des régions de Voronoï, et ses applications directes. Ces algorithmes étant basés sur la notion de distance, ils tirent avantageusement parti de la `direct-association` asynchrone et du r-opérateur `minc`. La manipulation des structures géométrique est aisée avec les sous-graphes.

Les réseaux associatifs semblent donc tout particulièrement adaptés pour traiter ce genre d'algorithmes. Les applications des réseaux associatifs ne s'arrêtent pas aux exemples détaillés précédemment.

Nous signalons pour finir deux autres adaptations de problèmes classiques d'algoithmique géométrique.

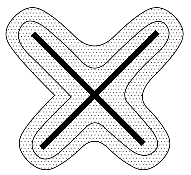


FIG. 101 – Exemple d'application de la transformée en distance : la squelétisation d'une image permet de reconnaître les formes.

La recherche de la paire de points la plus proche (problème P.1 p. 180 dans [Preparata et Shamos, 1985]) est réalisée comme suit. On obtient les polygones de Voronoï par l'algorithme 27. Puis on obtient le graphe des frontières en prenant le complémentaire du graphe des régions. Tous les points ayant un arc incident dans ce sous-réseau gardent la valeur de la distance à leur graine, obtenue par la minc-`direct-association` ; les autres prennent $+\infty$. Il ne reste plus qu'à réaliser une max-`direct-association` sur toute la grille pour connaitre la plus petite des demi-distances entre graines. Ensuite, les points frontières qui possédaient cette distance sont aisément identifiés ; cette information est ensuite transmise à leurs graines respectives. En reportant ces informations dans le graphe virtuel, les graines sélectionnées retrouvent rapidement leur plus proches voisines.

La recherche du plus proche voisin de chaque point (problème P.2 p. 180 dans [Preparata et Shamos, 1985]) est effectuée par une min-`direct-association` sur le graphe des régions, avec la pvar des distances à la graine, préalablement mise à $+\infty$ sur les sommets qui ne sont pas sur la frontières (*i.e.*, ceux qui n'ont pas d'arc incidents dans le graphe frontière, complémentaire dans la grille du graphe des régions).

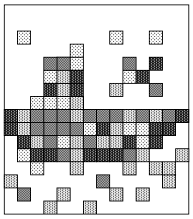


FIG. 102 – Image initiale.

7.6 Algorithmes de traitement d'images

Au chapitre 2, nous avons illustré la bonne adéquation du modèle au traitement d'images. Celle-ci est notamment due aux outils de manipulation des graphes, ainsi qu'aux primitives de calcul sur ces structures, qui se révèlent fort utiles pour l'algorithmique d'images. Dans cette section, nous indiquons comment les algorithmes de traitement d'images peuvent tirer parti des r-opérateurs, et de l'asynchronisme de la `direct-association`.

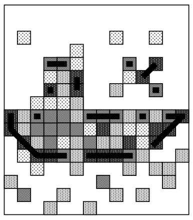


FIG. 103 – Obtention de morceaux de contours par des opérateurs de bas niveau.

7.6.1 Distance et applications

Les résultats suivants sont tirés de [Ducourthial et Mérigot, 1998].

Transformée en distance. La *transformée en distance*[116] consiste à étiqueter chaque pixel avec la plus petite distance le séparant de «pixels-graines». Il s'agit d'un algorithme de base, utilisé pour d'autres transformations, telles que la tranformée en axe médian, la «squelettisation» (figure 101), etc. Tous ces algorithmes bénéficient du calcul de la distance en une `direct-association`, grâce au r-opérateur minc ou minc_w, que nous avons présentés au § 7.5.1 page 143.

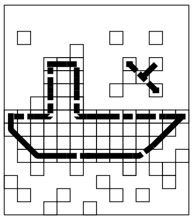


FIG. 104 – Fermeture des contours : recherche d'un meilleur chemin dans la grille avec une $\mathtt{minc}_w$-`direct-association`.

Segmentation de Voronoï. L'algorithme 2 page 25, qui construit les régions de Voronoï sans r-opérateur, utilise une boucle sur les régions voisines, elle-même incluse dans une boucle jusqu'à la stabilité. Chaque boucle interne contient un calcul de distance et des `associations`. Outre l'optimisation de l'algorithme, la `direct-association` asynchrone offre un code plus concis pour le même algorithme ; une quantité de travail plus importante est confiée au modèle de programmation. Les codes des deux versions de l'algorithme de Voronoï sont donnés à l'annexe A page 185.

Fermeture de contours. La fermeture de contour est un problème classique de l'analyse d'images. L'application d'opérateurs de bas niveau permet d'obtenir une mesure de l'intensité des contours sur tous les pixels. Mais du bruit peut perturber le résultat, et le contour peut, au final, ne pas être clos (figure 102 et 103). La fermeture des contours peut être réalisée en cherchant le meilleur chemin reliant deux extrémités d'un contour.

Après la mesure de l'intensité, les points au dessus d'un seuil sont considérés comme sûrs, tandis que les autres reçoivent un coefficient inversement proportionnel à leur intensité. Il s'agit alors de

116. En anglais, *Distance transform (DT).*

trouver un chemin reliant deux pixels sûrs (*i.e.*, faisant partie du contour), et présentant une somme des coefficients minimale. Le problème est résolu en une $\mathtt{minc}_w$-`direct-association`, sur le graphe de la grille, dont les arcs (u, v) sont valués par le coefficient du pixel u.

La méthode peut être raffinée en prenant en compte l'orientation du contour : le poids de l'arc (u, v) est réduit si l'orientation du contour local sur le pixel u ne va pas vers le pixel v.

7.6.2 Segmentation par partage des eaux

La segmentation d'images par ligne de partage des eaux assimile l'image à un relief (figure 105-gauche). Les minimas locaux sont considérés comme des sources, qui inondent le paysage. Les petites vallées sont englouties par les eaux, mais celles-ci ne franchissent pas les crêtes importantes, laissant apparaître les gros reliefs. Plus l'eau monte, et moins il reste de détails dans l'image. Les algorithmes sont itératifs : chaque pas correspond à une montée supplémentaire des eaux ; on contrôle alors la surface de chaque lac (ou sa profondeur, son volume, *etc.*), de façon à déterminer s'ils correspondent à un relief suffisament important pour lui interdire de fusionner avec les lacs voisins [Guezguez et al., 1998]. Les critères d'arrêt de la montée des eaux sont des paramètres de l'algorithme. La figure 105-milieu illustre la montée des eaux dans le relief de gauche tandis que le résultat final de la segmentation est donné par la figure 105-droite (vue de haut).

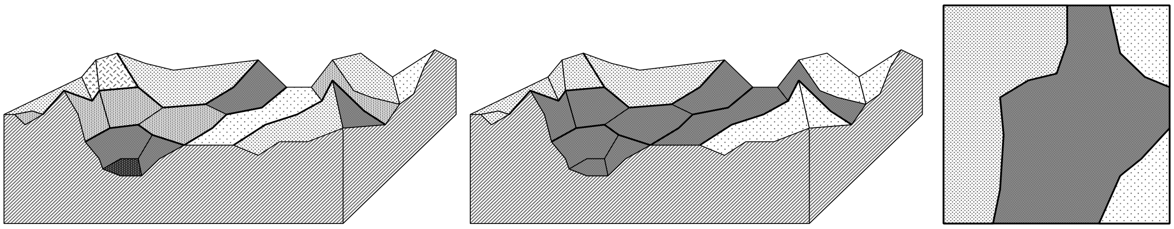


FIG. 105 – *Segmentation par ligne de partage des eaux.*

Nous indiquons maintenant une adaptation de la segmentation par ligne de partage des eaux au modèle des réseaux associatifs, basée sur la propagation asynchrone de l'eau, et un r-opérateur. Une adaptation utilisant des s-opérateurs, et destinée à la Maille Associative, peut être trouvée dans [Guezguez et al., 1998].

On commence par fabriquer le sous-réseau de la montée des eaux, en masquant tous les arcs partant d'un pixel de niveau clair vers un pixel de niveau foncé (figure 109 page suivante). On utilise le r-opérateur n-aire $\mathtt{minc}_w$. Les poids utilisés influence l'heuristique. Ils doivent refléter la progression plus rapide de l'eau sur pente douce que sur pente raide ; ils doivent aussi rendre compte de l'avancée de l'eau sur terrain plat. Deux critères sont donc à combiner : différence de niveau de gris et distance. Nous proposons d'attribuer à l'arc (u, v) reliant les pixels u et v de l'image codée par la pvar p, le poids $w\,(\,(u, v)\,) = p[v] - p[u] + 1$. Tous les poids sont positifs dans le sous-réseau de la montée des eaux.

Tout pixel qui est un minimum local dans l'image est une source pour la montée des eaux. Une telle source est initialisée avec la valeur du pixel correspondant. Les autres sommets sont initialisés avec $+\infty$. La montée des eaux est réalisée par une $\mathtt{minc}_w$-`direct-association`.

La valeur obtenue par chaque nœud correspond à la hauteur de l'eau dont il est recouvert. Cependant, l'eau a englouti toutes les régions. Il nous faut déterminer les quantités d'eau qui ont rempli chacune de ces régions, et décider celles qui doivent être englouties et celles qui ne le doivent pas. Pour cela, nous retournons le graphe des régions. Les pixels des crêtes n'ont pas d'antécédent dans ce sous-réseau. Tous les sommets sont initialisés à 0 sauf ces pixels-crêtes, qui gardent leur hauteur d'eau. En calculant une `min-direct-association`, chaque source (minimum local, desquelles «sortait l'eau») connaît la hauteur de la plus basse crête de sa région. Elle peut alors calculer la hauteur de la région, et décider de disparaître si elle ne respecte pas les critères de hauteur de région de l'algorithme.

Après la suppression d'éventuelles sources, une nouvelle itération commence. Ces itérations se terminent lorsqu'aucune source n'a été supprimée. La figure 109 illustre le résultat final de ce procédé itératif. Notons qu'une seule `direct-association` suffit pour simuler la progression de l'eau.

Pour finir, signalons qu'il est aussi possible de calculer les critères plus complexes de surface ou de volume des «cuvettes» (régions). Pour cela, on forme un sous-réseau des régions obtenues à l'étape courante, en masquant les arcs du sous-réseau de la montée des eaux qui atteignent un pixel plus haut que la plus basse crête de la région. Dans le sous-réseau des régions, on calcule la surface par une plus-`direct-association` sur une arborescence couvrante, après avoir initialisé tous les nœuds à 1. Le volume est estimé en sommant les différences de hauteur entre la crête et les pixels.

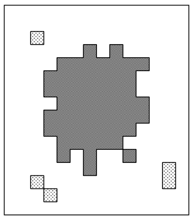


FIG. 106 – Image initiale.

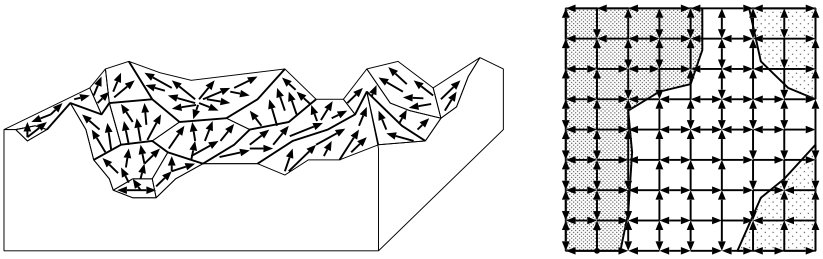


FIG. 109 – *Construction du sous-réseau de la montée des eaux.*

7.6.3 Morphologie mathématique

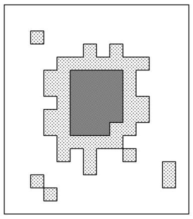


FIG. 107 – Image érodée (en foncé). Les défauts sortants sont gommés.

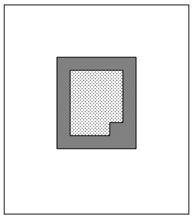


FIG. 108 – Image dilatée de l'érodée. Les défauts entrants sont gommés. L'image finale (en foncé) est plus simple que l'image initiale de la figure 106, mais les formes sont respectées.

La morphologie mathématique a pour but de prendre en compte des critères plus proches de ceux retenus par l'œil humain pour l'analyse d'images. Nous illustrons les possibilités des calculs asyncrhones et des r-opérateurs dans ce domaine en considérant l'ouverture et la fermeture par reconstruction. Il s'agit d'opérateurs de morphologie mathématique qui permettent de simplifier l'image — et en particulier de supprimer le bruit — tout en préservant les contours. Il font partie des *filtres par reconstruction morphologiques*, qui sont largement utilisés dans les phases de prétraitement précédent les segmentations. Ces opérateurs font partie d'une classe plus large, appelée *opérateurs connectés*[117] [Salembier et Pardàs, 1994] [Salembier et Serra, 1995]. Ceux-ci ont la particularité de ne pas travailler sur les variations de fréquence, mais d'interagir avec le signal (*cf.* § 1.1.1 page 3) sur les *zones plates*[118] (niveaux de gris constant).

L'opérateur d'*érosion* donne au pixel le minimum des valeurs dans un voisinage dont il est le centre. Lorsque le voisinage est 3×3 (voisins du pixel dans la grille en 8-connexité), l'opérateur d'érosion est défini par $\mathsf{min}(x_0, \ldots, x_8)$, où x_0 est la valeur du pixel, et $x_1 \ldots x_8$ cells de ces 8 voisins. L'opérateur de *dilatation* donne au pixel la plus grande valeur de son voisinage : $\mathsf{max}(x_0, \ldots, x_8)$. L'opérateur d'érosion diminue la surface des objets, en supprimant tous les petits défauts des contours qui sortent de l'objet (figure 106 et 107), tandis que l'opérateur de dilatation agrandit les objets, en supprimant les défauts entrants (figure 108). L'*érosion géodésique* consiste à borner le processus d'érosion, en prennant le maximum de l'image érodée et d'une image de référence : $\mathsf{max}(y_0, \mathsf{min}(x_0, \ldots, x_8))$. De même, la *dilatation géodésique* borne la croissance des objets : $\mathsf{min}(y_0, \mathsf{max}(x_0, \ldots, x_8))$. Finalement, la *fermeture par reconstruction* consiste à appliquer jusqu'à stabilisation — et de façon synchrone — une érosion géodésique en prennant toujours l'image initiale comme référence. De même, l'*ouverture par reconstruction* consiste à appliquer successivement des dilatation géodésiques avec toujours l'image initiale comme référence. Ces opérateurs peuvent s'écrire sous la forme :

$$
\begin{aligned}
\textit{FerRec}\,(x_0, \ldots, x_8) &= \mathsf{min}\,(x_0, \mathsf{max}\,(x_0, \ldots, x_8)) \\
\textit{OuvRec}\,(x_0, \ldots, x_8) &= \mathsf{max}\,(x_0, \mathsf{min}\,(x_0, \ldots, x_8))
\end{aligned}
$$

Les opérateurs *FerRec* et *OuvRec* sont de rs-fonctions préservant la valeur initiale, et dont l'unique r-fonction est expansive (mais pas strictement). Il s'agit donc de r-opérateurs généralisés idempotents, et une `direct-association` utilisant un tel opérateur, appliquée sur le réseau de la maille non

117. En anglais, *Connected operator.*

118. En anglais, *flat zone.*

torique (grille en 8-connexité), converge[(m)]. Mais on ne sait pas *a priori* si le résultat des itérations asynchrones donnera le même résultat que les itérations synchrones (*cf.* 5.4.2 page 98). Cependant, dans [Planet et al., 1993] [Privat et al., 1995], les auteurs étudient l'application asynchrone de ces opérateurs, pour obtenir des circuits cellulaires asynchrones rapides, et on trouve une preuve de l'équivalence entre les itérations synchrones et asynchrones dans [Robin et al., 1997]. Ces résultats indiquent que les itérations asynchrones calculées par la `direct-association` donnent le même résultat que les itérations synchrones.

En conclusion, les opérateurs de morphologie mathématique de fermeture par reconstruction et d'ouverture par reconstruction peuvent être calculés avec une seule `direct-association` utilisant les opérateurs *FerRec* et *OuvRec*.

7.7 Conclusion

7.7.1 Bilan

Dans ce chapitre, nous avons développé une algorithmique pour le modèle des réseaux associatifs, plus générale que le traitement d'images. Au § 7.2, nous avons donné des algorithmes pour reconnaître la nature du réseau. Au § 7.4, nous avons illustré l'intérêt des r-opérateurs pour divers problèmes de cheminement dans les graphes. Au § 7.5, nous avons étudié les applications de l'opérateur minc en algorithmique géométrique. Nous avons fini par présenter l'intérêt des r-opérateurs en traitement d'images.

En particulier, une seule minc-`direct-association` calcule une distance dans une image, et permet d'obtenir les régions de Voronoï en une seule opération globale. Cela améliore très sensiblement l'algorithme que nous avons détaillé au chapitre 2, et dans [Ducourthial et al., 1998]. De même, le calcul de la forêt couvrante des plus courts chemins dans un graphe ne nécessite qu'une seule étape globale, ce qui améliore grandement l'algorithme itératif à base de s-opérateur donné dans [Mérigot, 1997a]. Cet algorithme procédait par améliorations successives d'une arborescence couvrante sur laquelle la distance était calculée avec une plus-`prefix-association`.

7.7.2 Programmation asynchrone

On peut dégager de cette étude algorithmique une méthodologie de programmation asynchrone. Le point central est l'idempotence de l'opérateur par rapport à la nature du sous-réseau.

Si l'opérateur est idempotent, il peut être appliqué sur tout type de réseau. S'il ne l'est pas, il faut envisager un sous-réseau préfixe (généralement une arborescence couvrante), qui évolue avec les besoins de l'algorithme. Nous avons illustré cette technique dans les algorithmes itératifs 11 page 129 (détection de DAG) et 14 page 132 (arborescence en profondeur).

Il peut cependant exister un opérateur plus long à converger (borné), pour obtenir le résultat en une seule `direct-association`. C'est ce que nous avons montré avec l'algorithme de détection des DAGs qui utilise l'opérateur borné minmaxinc. De même, des opérateurs plus complexe (et nécessitant des pvars plus larges) permettent de remplacer les itérations. Nous avons illustré cette possibilité avec le r-opérateur lexicat, qui travaille sur des listes, mais qui fournit une arborescence en profondeur en une seule `direct-association`.

Il n'est pas exclu de trouver des opérateurs plus complexes pour les problèmes qui nécessitent encore actuellement des itérations. L'intérêt du remplacement des itérations par le calcul d'une seule `direct-association` utilisant un opérateur plus complexe est illustrée au chapitre suivant. Indépendamment de ces performances, on peut conclure ici sur l'intérêt de l'asynchronisme

(m) Notons que si le voisinage est plus grand que le carré 3×3, on peut considérer un sous-réseau tel que les pixels à distance 2 les uns des autres dans l'image soient voisins dans le sous-réseau. Mais le voisinage 3×3 est généralement suffisant ; il est préférable pour les circuits électroniques spécialisés.

dans l'écriture des algorithmes. La `direct-association` offre une expressivité accrue aux programmeurs.

7.7.3 Adaptation à l'algèbre de chemins

Certains algorithmes que nous avons présentés sous la forme d'algorithmes de réseau asynchrone, grâce à un r-opérateur, sont des adaptations d'algorithmes proposés pour l'algèbre de chemins. Ainsi, dans [Gondran, 1975], l'auteur propose un algorithme matriciel synchrone pour la recherche des chemins les plus courts, les plus fiables ou de capacités maximales. Une version lexicographique de la recherche d'une arborescence en profondeur a été étudiée dans [de la Torre et Kruskal, 1995] ; mais le graphe ne devait pas avoir de circuit. L'opérateur lexicat que nous avons proposé est idempotent (et même strictement idempotent), et peut donc être appliqué dans un graphe ayant des circuits.

D'un autre côté, les algorithmes construits avec une seule `direct-association` utilisant un r-opérateur binaire ou n-aire peuvent s'adapter à l'algèbre de chemins généralisé (*cf.* § 4.1.7 page 60) : les étiquettes des arcs sont les r-fonctions. Ainsi, les opérateurs lexicat et Ant induisent des algorithmes matriciels (synchrones ou non) pour résoudre le problème de l'arborescence en profondeur ou de l'obtention des antécédents (et du calcul du diamètre).

Adaptation en systèmes distribués. L'algorithmique des réseaux associatifs est essentiellement basée sur le concept d'algorithme de réseau. Cette façon de penser les algorithmes permet des adaptations en systèmes distribués. La comparaison est aisée lorsqu'une seule `direct-association` est nécessaire pour l'algorithme, et qu'il existe un nœud privilégié, tel qu'une racine (ces conditions sont remplies par un certain nombre d'algorithmes créant des arborescences couvrantes particulières, *cf.* § 7.4 page 137). En effet, on utilise l'adaptation de la `direct-association` en vague, qui inclut sa propre détection de terminaison (*cf.* § 6.1.4 page 107). On peut aussi considérer des tâches silencieuses, qui ne prennent jamais fin, et qui sont destinées à maintenir une information (telle qu'une arborescence couvrante) dans un réseau, malgré les pannes (*cf.* § 4.3.2 page 73 et § 6.2 page 109).

Si l'algorithme comporte plusieurs `direct-associations`, on peut considérer des algorithmes en phases [Raynal et Hélary, 1988] [Tel, 1994]. Chaque `direct-association` correspond à des phases différentes. Il est alors nécessaire de synchroniser les nœuds entre chaque phase, ou d'étiqueter les messages par le numéro de la phase, pour ne pas les mélanger (un nœud ne participe à la phase $i + 1$ que s'il a terminé la phase i). S'il n'y a qu'un seul initiateur, l'adaptation en vague de la `direct-association` reste valable pour ce cas là aussi.

Pour finir signalons que la complexité en temps est toujours privilégiée dans les réseaux associatifs, au détriment de la complexité en messages. L'équivalent en systèmes distribués des algorithmes que nous avons présenté n'est donc pas toujours pertinente, selon les modèles utilisés [Ramarao et Venkatesan, 1992].

Un certain nombre d'algorithmes présentés bénéficient de la preuve générique de l'auto-stabilisation de la `direct-association` (*cf.* § 6.2 page 109). Les tableaux suivants récapitulent les caractéristiques des algorithmes de ce chapitre.

Nom	**Numéro**	**Description**	**Complexité** (en phases globales)
SourcesLogiques	6 page 127	Détecte les sources logiques.	2
SourcesPrincipales	7 page 127	Détecte les sources. Basé sur l'algorithme 6.	2
TriTopologique	8 page 127	Fournit une numérotation isotone non injective sur un DAG. Une phase globale. r-opérateur maxinc non idempotent. Auto-stabilisant sur un DAG.	1
Dag1	9 page 128	Détecte si le réseau est un DAG. Version avec opérateur borné.	1
ArcInverse	10 page 128	Détecte les arcs inverses.	2
Dag2	11 page 129	Détecte si le réseau est un DAG. Version itérative. Basé sur les algorithmes 6, 19 et 10.	$O(\|V\|)$
GraphePréfixe	12 page 130	Détecte si un DAG ayant une seule source est un graphe préfixe. Basé sur l'algorithme 19.	2
NumProfondeurArb	13 page 131	Numérote en profondeur une arborescence T.	$O(\delta_T^+)$
NumProfondeur	14 page 132	Fournit une numérotation en profondeur (et l'arborescence T correspondante) d'un graphe G. Basé sur les algorithmes 10, 20.	$O(h(T) \times \delta_G^+)$
CC	15 page 133	Détecte les CCs d'un graphe.	1
CFC1	16 page 134	Détecte les CFCs. Version utilisant la numérotation en profondeur. Basé sur les algorithmes 7 et 14.	$O(h(T) \times \delta_G^+)$
CFC2	17 page 135	Détecte les CFCs. Version utilisant un opérateur de listes.	2
CFC3	18 page 135	Détecte les CFCs. Itérations.	$O(\|V\|)$

FIG. 110 – *Algorithmes de graphe.*

Nom	**Numéro**	**Description**	**r-opérateur**	**Auto-stabilisant**
Distance	19 page 138	Calcule la distance.	minc	oui
ForêtCouvrante	20 page 138	Fournit une forêt couvrante des plus court chemins.	minc	oui
FCPoidsMin	21 page 139	Fournit une forêt couvrante des chemins de poids minimum.	minc_w	oui
FCFiabilitéMax	22 page 139	Fournit une forêt couvrante des chemins de fiabilité maximale.	$\mathsf{maxmul}_{\overline{\pi}}$	oui
FCCapacitéMax	23 page 140	Fournit une forêt couvrante des chemins de capacité maximale.	maxmin_κ	non
ArbProfondeur	24 page 141	Fournit une arborescence couvrante en profondeur enracinée sur une source principale.	$\mathsf{lexicat}_l$	
Diamètre	25 page 142	Fournit le diamètre du réseau. Permet aussi d'obtenir tous les antécédents de chaque nœud.	Ant	

FIG. 111 – *Algorithmes de parcours utilisant des* r*-opérateurs.*

Nom	Numéro	Description	Complexité
RégionsVoronoï	26 page 144	Fournit les régions de Voronoï dans le plan discrétisé modélisé par une grille. Opérateur `minc`.	2 phases globales, 3 locales.
PolygoneVoronoï	27 page 144	Fournit les polygones de Voronoï. Basé sur l'algorithme 26.	2 phases globales, 6 locales.
PointsEnvConvexe	28 page 145	Fournit l'enveloppe convexe de points dans une portion du plan (suffisamment grande). Basé sur l'algorithme 26.	3 phases globales, 5 locales.

FIG. 112 – *Algorithmes géométriques.*

Chapitre 8

La bibliothèque ANET

Après la définition du modèle dans la première partie et son étude théorique dans la seconde, nous avons abordé son algorithmique au chapitre 7 page 123. Nous continuons cette troisième partie par l'étude de la mise en œuvre du modèle. Nous proposons à cette fin une bibliothèque de programmation parallèle.

Cette bibliothèque, appelée ANET, a pour but de montrer que les idées théoriques développées précédemment peuvent être mises en pratique en programmation.

ANET est une bibliothèque de programmation data-parallèle, qui manipule des objets irréguliers, et qui gère l'irrégularité des traitements par les processus légers de l'environnement PM^2 (*Parallel Multithreaded Machine*). Elle propose au programmeur les différentes primitives de calcul des réseaux associatifs, dont la `direct-association` asynchrone.

Nous commençons au § 8.1 par présenter les diverses pistes possibles pour la mise en œuvre du modèle des réseaux associatifs redéfini, et nous précisons nos objectifs pour la bibliothèque parallèle. Nous proposons ensuite une courte présentation aux *threads*. Puis nous détaillons au § 8.3 les structures de données d'ANET, qui permettent la manipulation de données irrégulières et de topologies quelconques, et nous expliquons comment les primitives de calcul sont mises en œuvre, et parallélisées. Enfin, au § 8.4, nous relatons les expérimentations que nous avons menées sur diverses plates-formes, afin d'évaluer et de calibrer la bibliothèque. Nous concluons au § 8.5, sur les différentes idées avancées pour la conception d'ANET.

8.1 Mise en œuvre des réseaux associatifs

Nous avons étudié la mise en œuvre de la `direct-association` dans un contexte distribué au chapitre 6. Nous avons aussi indiqué quels sont les algorithmes auto-stabilisants, basés sur un r-opérateur, qui sont applicables dans ce cas. Ces algorithmes sont destinés au contrôle du réseau, par exemple pour maintenir les routes les plus fiables, ou les plus courtes, *etc.*

Nous envisageons ici une mise en œuvre du modèle à des fins de calcul parallèle. Nous commençons par étudier l'adaptation de la Maille Associative au modèle redéfini et aux r-opérateurs. Nous abordons la mise en œuvre logicielle ensuite.

8.1.1 Mise en œuvre dans la Maille Associative

Limitation des opérateurs. À l'origine, le modèle des réseaux associatifs a été conçu en étroite relation avec une étude architecturale, la Maille Associative (§ 1.2.2 page 13). Nous avons mis en avant certaines différences entre le modèle de calcul théorique et le modèle d'exécution de la Maille (§ 3.1 page 29). Partant de là, nous avons redéfini les réseaux associatifs sous une forme plus proche de la Maille, en intégrant notamment l'asynchronisme de cette machine dans les primitives du modèle (§ 3.2 page 35). Aussi le modèle peut, *a priori*, être mis en œuvre dans la Maille Associative.

Cependant, si le modèle d'exécution de cette machine est en parfaite adéquation avec le modèle de calcul que nous avons étudié, son électronique souffre de quelques limitations quant au choix des opérateurs utilisables. Les opérateurs complexes, tels que ceux travaillant sur des listes, sont à exclure. Mais les r-opérateurs «numériques», basés sur les s-opérateurs min et max, tel que le minc, posent également des problèmes.

En effet, les chemins de communication de la machine permettent de ne véhiculer qu'un seul bit à la fois, ce qui implique de sérialiser les communications. Ainsi, pour réaliser une min-association sur une pvar d'octets (8 bits), on effectuera 8 min-associations avec des pvars de un bit. Ces pvars proviennent des bits des octets de la donnée initiale, que l'on prend un par un, en commençant par les poids forts (le calcul du minimum des deux octets s'effectue de la gauche vers la droite). Il en est de même pour le maximum. Par contre, pour une addition, le calcul s'effectuera de la droite vers la gauche. Une plus-prefix-association sur des octets est donc mise en œuvre dans la Maille Associative par huit plus-prefix-associations sur des pvars de bits provenant des octets lus de la droite vers la gauche.

Le mixage de l'opérateur plus et de l'opérateur min est donc difficilement conciliable avec des communications bit-séries.

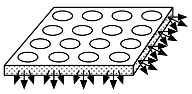


FIG. 113 – La largeur de coupe de la maille limite le nombre de nœuds dans le circuit. Seules les connexions sortantes sont représentées ici.

Chemin de données plus large. L'utilisation des r-opérateurs dans la Maille Associative implique donc d'avoir un chemin de données plus large[a]. Un chemin de données de 8 bits permettrait de traiter les images en 256 niveaux de gris en une association pour les s-opérateurs. Il permettrait aussi de calculer les distances dans les mailles de taille 256×256, lorsque les connexions toriques, ou les diagonales, ne sont pas masquées (dans les deux cas, le diamètre du réseau est 256). Les distances pondérées seraient cependant limitées à des mailles plus petites, ou à des sous-réseaux de diamètre plus faible (régions).

Une Maille Associative peut difficilement être réalisée en assemblant plusieurs circuits intégrés sur une carte. En effet, si nous considérons une maille 256×256 (pour des images courantes), il faut 16 boîtiers de 64×64 processeurs élémentaires (PE). Pour connecter ces boîtiers, il doivent avoir les entrées-sorties nécessaires. Or une maille $n \times n$ présente une largeur de coupe (*cf.* figure 113) de $3n$ connexions et $12n - 4$ connexions extérieures au total, lorsqu'elle n'est pas torique (ce qui est nécessaire pour abouter plusieurs circuits dans le but d'en construire un plus grand). Pour une maille 64×64, cela fait plus de 900 entrées-sorties, lorsqu'on ajoute le *scan* d'entrée / sortie de l'image et les signaux de contrôle (instructions, *Global-or*, *etc.*). Puisque les circuits intégrés ont une limitation technologique dans le nombre de connexions extérieures admises, il n'est guère envisageable de construire une grande Maille Associative par un assemblage de plusieurs petites, sauf à avoir de trop nombreux petits circuits, ce qui grèverait les performances en multipliant les communications entre les circuits intégrés (qui sont plus lentes). Le passage des communications bit-série aux communications à 8 bits agrandit la surface du circuit, mais il est illusoire de compter sur l'augmentation du nombre de circuits élémentaires pour obtenir une plus grande surface de circuit.

Il faut donc envisager un seul circuit intégré, duquel les connexions de la maille n'auront pas besoin de sortir. Seules les entrées / sorties d'images et les signaux de contrôle seront nécessaires, ce qui reste raisonnable. Pour une maille de 256×256 PE, il faudrait un circuit d'environ 5 cm de côté en technologie 0,25 μm. Pour ce type de réalisation, les techniques de *wafer scale*, qui consistent à utiliser au maximum la surface d'une galette de silicium, semblent appropriées. Mais elles sont chères. Une solution envisageable est celle du *multi-chip module*, qui consiste à apposer des circuits intégrés non encore insérés dans un boîtier sur un substrat de silicium. Contrairement au *wafer scale*, il permet de fabriquer les petits circuits par une technique classique moins onéreuse, et ainsi de limiter les coûts. Cette solution permet d'avoir plus de connexions sortantes de chaque boîtier que ne le permet l'assemblage des circuits intégrés sur une carte, car les connexions sont directement routées dans le substrat de silicium.

Enfin, notons que le chemin de données asynchrone d'une Maille Associative représente seulement 5 % de son électronique. La multiplication par 8 de la surface de ces chemins ne devrait pas

(a) La technique des «codes redondants» permet d'effectuer tous les calculs en prenant les bits dans le même sens, quel que soit l'opérateur. Mais le nombre de bits pour coder une donnée est beaucoup plus important. En outre, il est nécessaire d'indiquer le numéro du bit véhiculé sur un lien de communication, ce qui en augmente la largeur. Cette technique ne semble donc pas résoudre efficacement le problème de l'implémentation des r-opérateurs dans la Maille.

augmenter la surface du circuit de plus de 35 %.

Conclusion. L'utilisation des r-opérateurs dans une Maille Associative nécessite de la modifier pour avoir un chemin de données plus large. Cela ne peut se faire qu'en construisant un circuit de type *multi-chip module* d'une surface d'environ 35 % supérieure à celle prévue initialement. Notons que cette technologie est courante. Par exemple, les processeurs Pentium de la société Intel sont conçus ainsi : le cache L2 est un module séparé du reste du processeur, et les morceaux sont assemblés sur un seul circuit intégré. Sans sous-estimer les problèmes de l'industrialisation d'un tel circuit, il semble que cette solution soit réalisable ; le problème principal est d'ordre financier.

8.1.2 ANET

Intérêt dans l'étude. La validation d'un modèle de calcul théorique par une architecture électronique est, certes, attirante. Néanmoins il semble opportun de ne pas cantonner les applications du modèle à une machine dédiée ; cela limiterait considérablement son utilisation. La réalisation d'un environnement logiciel apparaît donc comme une étape incontournable dans l'étude d'un modèle de calcul. En fait, cette ultime étape est celle qui justifie les précédentes : définition, étude, algorithmes. En effet, comme nous le signalions dans l'introduction, un modèle de programmation présente une ambivalence typique entre le haut et le bas niveau, c'est-à-dire entre l'abstraction pratique pour programmer et les facilités de mises en œuvre aidant les implémentations. Ignorer la mise en œuvre du modèle revient à effacer un pan de la difficulté inhérente à l'étude d'un modèle de calcul. Le reste de ce chapitre est consacré à l'étude de la mise en œuvre logicielle du modèle des réseaux associatifs.

Stratégies d'implémentation. Trois solutions sont envisageables pour construire cette plate-forme logicielle. La première consiste à développer un langage data-parallèle approprié. Cette méthode est assez complexe. Elle présente l'inconvénient de développer *from scratch*, et de se trouver confronté à des difficultés rencontrées par d'autres, et résolues ailleurs. En outre, l'acceptation et l'utilisation d'un nouveau langage n'est jamais facile. La seconde solution consiste à adapter un langage data-parallèle existant aux besoins des réseaux associatifs. Une première expérience avait été tentée à l'IEF avec le langage Hyper C. Cette tentative n'avait pas abouti, notamment parce que le langage a cessé d'être maintenu. En fait, le modèle des réseaux associatifs est assez différent des langages data-parallèles existants, et construire une couche logicielle au dessus de ces langages ne peut conduire à une réalisation efficace. Nous avons opté pour une troisième solution, qui consiste à développer, non pas un langage mais une bibliothèque de programmation, écrite dans un langage classique. Cette solution est plus simple que la création d'un langage data-parallèle complet, et facilite la diffusion de la plate-forme logicielle auprès des utilisateurs du langage. En outre, elle permet de mettre en place des optimisations délicates, sinon interdites dans la deuxième solution, et de se baser sur des supports d'exécution multi-fils[119], maintenant classiques, voire incontournables, pour ce type de réalisation (*cf.* 8.2).

119. En anglais, *multithread*.

Généralisation de SAM. Un simulateur fonctionnel de la Maille Associative a été développé par S. Guezguez et D. Dulac [Guezguez et al., 1998]. Orienté vers l'étude de l'architecture, il ne permet cependant pas l'exécution efficace d'algorithmes. Aussi A. Mérigot a développé une bibliothèque séquentielle écrite en C++ appelée SAM. Plus éloignée de l'architecture, mais reprenant cependant son principe de fonctionnement, SAM permet de programmer et d'exécuter des algorithmes dans le modèle des réseaux associatifs conforme à la première définition, sur une topologie en maille, sans que le ralentissement de la simulation ne soit pénalisant.

Par un souci d'efficacité et de compatibilité des programmes développés, nous avons décidé de construire la bibliothèque de programmation parallèle ANET en partant de SAM. ANET généralise SAM selon trois axes. Premièrement la topologie d'ANET est paramétrable, ce qui permet de programmer sur un réseau quelconque, tandis que la topologie utilisée dans SAM est figée, en forme de maille. Ensuite, ANET inclut les primitives de SAM (`associations`) pour des raisons de compatibilité des programmes, mais propose aussi au programmeur la `direct-association` du modèle redéfini. Enfin, ANET est résolument tournée vers le calcul parallèle plutôt que la simulation de la Maille Associative. Il s'agit d'une bibliothèque parallèle portable destinée au «calcul intensif».

On retrouve cependant dans ANET des fonctionnalités de SAM telles que les contextes d'exécution `WHERE(condition)-ELSEWHERE-ENDWHERE`[120], les opérateurs de base, les primitives de manipulation des sous-réseaux, certains algorithmes de parcours de graphe, et le calcul des `associations`. Ces parties n'ont pas été redéveloppées dans ANET mais simplement adaptées. Les programmes écrits pour SAM peuvent être compilés avec ANET, à quelques renommages près.

120. Contexte d'exécution : structure algorithmique permettant d'appliquer une suite d'instructions à une sous-partie du réseau, dont les nœuds vérifient la `condition`.

121. En anglais, *SMP : Share Memory Processors*

Objectifs de la parallélisation. Nous envisageons d'utiliser ANET sur machines à mémoire partagée[121], car les stations de travail ou les PC bi- ou quadri-processeurs sont de plus en plus courants. Mais nous souhaitons aussi pouvoir utiliser ANET sur réseau de stations de travail, car ces plates-formes se développent actuellement. Elles bénéficient d'un regain d'intérêt grâce aux réseaux rapides, qui en font des machines parallèles bon marché.

Le modèle des réseaux associatifs est basé sur un réseau de base, sur le maniement de ses sous-réseaux, et sur les calculs dans ces structures. La parallélisation des `direct-associations` doit respecter l'irrégularité des objets, et leur dynamicité (régions rapidement, et régulièrement, modifiées par exemple). En outre, à l'instar de la segmentation de Voronoï (*cf.* chapitre 2), les algorithmes que nous avons étudiés présentent une forte irrégularité des traitements. La parallélisation doit permettre d'équilibrer la charge des processeurs.

Le réseau de base s'apparente à une machine virtuelle d'instanciation ; tous les objets (pvars) créés ont la forme de cette machine. Grâce aux masques, des objets très irréguliers peuvent être «dessinés» dans cette topologie. Les `direct-associations` appliquent un calcul asynchrone indépendant à chacune des composantes connexes des sous-réseaux. Par contre, le calcul au sein d'une composante connexe peut difficilement être découpé en plusieurs calculs plus petits (en environnement à mémoire distribuée au moins), car il correspond à de multiples calculs locaux concurrents, exécutés jusqu'à la stabilisation.

Le placement de la machine virtuelle sur les processeurs physiques doit donc respecter l'«atomicité» des composantes connexes, leur irrégularité, leur dynamicité, et l'hétérogénéité des traitements à y appliquer. Ces caractéristiques, et les supports matériels auxquels nous destinons ANET, nous ont conduit à utiliser un environnement d'exécution multi-fil : PM^2 (*Parallel Multithreaded Machine*). Nous le présentons dans la section suivante.

Description d'ANET. ANET est une bibliothèque C++ qui permet l'écriture de programmes data-parallèles à données et traitements irréguliers, en utilisant la `direct-association` asynchrone. Elle repose sur les *threads* de PM^2, et peut s'exécuter sur toutes les plates-formes autorisées par ce support d'exécution, telles que les réseaux de stations de travail, les machines à mémoire partagée, et, plus généralement, toute plate-forme UNIX disposant de TCP / IP.

Un programme écrit avec ANET lance un processus «maître» et plusieurs copies «esclaves» sur des machines distantes. Les diverses `associations` sont parallélisées en respectant l'irrégularité des données (et des traitements) car les *threads* se répartissent les composantes connexes du sous-réseau. Le faible coût de création d'un *thread* PM^2 autorise une adaptation rapide à l'irrégularité des données et des traitements.

La topologie du réseau associatif est paramétrable mais figée durant l'exécution. ANET s'apparente donc à un environnement data-parallèle à machine d'instanciation, mais qui gère cependant des données irrégulières, grâce aux pvars de masques.

Ainsi, les caractéristiques majeures de la bibliothèque sont : **(a)** codage de l'irrégularité des données par des sous-réseaux, dans une machine d'instanciation figée au début du programme, **(b)** primitives de calcul data-parallèles originales, dont la `direct-association`, qui effectue un calcul asynchrone sur le sous-réseau, et **(c)** parallélisation des `associations` par des *threads* PM^2 qui exécutent les calculs en respectant l'irrégularité des données (attribution des *threads* aux composantes connexes du sous-réseau).

8.2 Fils d'exécution et environnements data-parallèles

Dans cette section, nous présentons une courte introduction aux *fils d'exécution*[122]. Nous nous concentrons sur leur utilisation dans la conception des plates-formes de programmation data-parallèles. Puis nous présentons l'environnement PM^2, qui est utilisé dans ANET.

122. En anglais, *thread.*

8.2.1 Introduction aux *threads**

Le principe des *threads* est la séparation des flots d'exécution des gestionnaires de ressources. On retrouve ce principe à plusieurs niveaux en informatique.

Différents types.* Un *thread* est obtenu en scindant en plusieurs fils d'exécutions, ou *processus léger*, un «processus lourd» géré par le système d'exploitation. Un processus léger ne peut vivre sans son processus lourd, qui lui sert de «carapace» vis-à-vis du système d'exploitation. Mais profitant de cette carapace, des processus légers sont rapidement créés ou détruits. Cette souplesse intéresse les environnements de programmations data-parallèles ; nous y revenons ci-après.

La notion de *thread* a tout d'abord fait son apparition dans les systèmes d'exploitation. On parle de *thread système* ; ils sont créés en contexte système, c'est-à-dire que le système doit intervenir pour leur création. Ils permettent de s'adapter aux temps de réponse des requêtes sur les diverses ressources de la machine. Par exemple, les *threads* aideront un serveur à répondre à des demandes de service «simultanées». Les *threads* systèmes permettent aussi de paralléliser le noyau sur une machine à mémoire partagée, en garantissant un bon équilibre de charge, comme dans le système Solaris de la société Sun.

On trouve aussi des *threads* dans les *processeurs multithreads*, qui gèrent des flots d'instructions indépendants, et peuvent passer de l'un à l'autre très rapidement [Lee et Hurson, 1994]. Ces *threads* sont des suites d'instructions qui ne sont pas interrompus, une fois qu'ils ont commencé leur exécution. Ils sont prêts lorsque les données qu'ils manipulent le sont. Les *threads* ne sont pas synchronisés ; ils sont ordonnancés par le processeur. En ce sens, les processeurs *multithreads* sont apparus comme une alternative aux architectures flot de données, qui se sont révélées plus complexes à réaliser que prévu. Ils combineraient, selon [Roh et al., 1996], les avantages des architectures de Von Neumann et des architectures dirigées par les données. Les processeurs *multithreads* permettent aussi de supporter une forte latence de la mémoire et du réseau, inhérente aux gros systèmes. En effet, lorsqu'une requête de lecture est lancée, le processeur passe très rapidement à un flot d'instructions différent pour ne pas perdre de temps. C'est la technique proposée dans le système MTA de la société Tera [Briggs, 1997].

Nous nous intéressons ici aux processus légers, créés en contexte utilisateur (c'est-à-dire sans faire appel au système). Ces *threads* sont plus rapidement créés et détruits ; ils présentent une plus grande souplesse d'utilisation. Nous expliquons leur intérêt pour les environnements à parallélisme de données avant de présenter des bibliothèques de programmation *multithreads*.

Intérêt des processus légers en data-parallélisme.* Plusieurs raisons motivent l'utilisation des *threads* pour la programmation data-parallèle : la finesse du processus léger est plus adaptée au grain fin des programmes data-parallèles ; sa souplesse permet de s'adapter à la dynamicité des algorithmes et de mieux recouvrir calcul et communications ; enfin, le lancement des *threads* à distance ou leur migration d'un processeur à un autre permet d'équilibrer la charge.

La compilation des programmes data-parallèles sur les machines à mémoire distribuée ne présente pas de difficulté majeure pour les problèmes réguliers : le parallélisme du système est correctement utilisé. Par contre, les algorithmes irréguliers nécessitent de trouver un bon placement des traitements sur les processeurs, et un bon ordonnancement des calculs les uns par rapport aux autres. Ces difficultés engendrent encore de nombreuses recherches [Roucairol, 1996] [Briat et al., 1997b]. Les langages data-parallèles tels que HPF supportent encore mal les algorithmes irréguliers. Les bibliothèques de communication telles que PVM ou MPI [Calvin et Colombet, 1994] s'accommodent de l'irrégularité des traitements, mais ce sont des outils de bas niveau, et le lourd travail d'équilibrage de charge est reporté vers le programmeur. De plus, le grain fin des environnements data-parallèles nécessite des mécanismes de partitionnement des traitements peu chers. Or PVM ou MPI n'autorisent que la création de processus lourds, peu souples, moins nombreux et dont l'ordonnancement est difficile à contrôler [Briat et al., 1997b]. Au contraire, le processus léger correspond à l'exécution d'une procédure et non d'un programme ; cette quantité de traitement se trouve en meilleure adéquation avec le grain fin du parallélisme de données. Les études actuelles semblent montrer qu'un algorithme irrégulier aura avantage à être programmé comme un réseau dynamique de *threads* communicants.

De plus, en jouant sur la taille des tâches, on peut mieux recouvrir les communications par des calculs, c'est-à-dire que certaines tâches s'exécutent pendant que d'autres sont en attente de communications [Briat et al., 1997a]. Les *threads* apparaissent comme une unité de traitement plus petite et donc plus précise. Leur souplesse autorise des ajustements dynamiques qui ne pénalisent pas le programme.

Enfin, les bibliothèques de programmation *multithreads* permettent de lancer un processus léger sur un processeur distant. Cette technique peut être utilisée pour mettre en œuvre diverses politiques d'équilibrage de la charge. La plate-forme PM^2 offre en outre la migration des processus légers, ce qui permet de réguler la charge *a posteriori*. Cela se révèle nécessaire pour les problèmes à l'irrégularité totalement imprévisible (tel que le *Branch and Bound*). Nous présentons maintenant certaines bibliothèques *multithreads*.

Bibliothèques *multithreads*.* Plusieurs bibliothèques de programmation parallèle à processus légers ont été développées. Citons par exemple Nexus, qui repose sur les *Remote Service Request* [Foster et al., 1996]. Elle sert de base notamment à l'environnement de programmation Nexeme [Moreau et al., 1997]. On trouve aussi la plate-forme Athapascan développée au LMC IMAG de Grenoble (projet APACHE), et l'environnement PM^2 étudié au LIFL à Lille (projet ESPACE), et au LIP à Lyon. Nous comparons rapidement ces deux dernières, avant d'expliquer notre choix pour PM^2.

Athapascan propose une gestion dynamique du parallélisme des programmes semi-réguliers [Briat et al., 1997b]. Il s'agit d'une bibliothèque C++ reposant sur PVM 3 ou MPI, et utilisant les *threads* POSIX (elle est utilisable sur toute machine ayant ses composants logiciels). Le système se compose de deux couches. La première, Athapascan 0, offre les outils de gestion des *threads* et de lancement à distance. La seconde, Athapascan 1, propose des mécanismes d'ordonnancement des calculs. Un calcul est découpé en sous-calculs, et à chaque étape, le grain du parallélisme est adapté dynamiquement à la situation du système. La parallélisme est explicite. Chaque appel de procédure parallèle (n-aire) fait l'objet d'une analyse par un régulateur, qui prend une décision quant à l'ordonnancement et au placement des calculs sur les machines. L'intérêt de ce système réside dans le fait que l'exécution suit le graphe de précédence de l'algorithme ; une exécution symbolique permet en outre d'obtenir les informations nécessaires à l'étude d'un ordonnancement statique qui pourra être utilisé pour les futures exécutions du programme.

La plate-forme de programmation *Parallel Multithreaded Machine*, ou PM^2, est destinée au *multithread* distribué. Elle est composée d'une bibliothèque de gestion des *threads* appelée *Marcel* [Namyst et Méhaut, 1995a], qui complète la norme POSIX, et d'une couche de communication optimisée, la bibliothèque Madeleine[123] [Bougé et al., 1998]. Un programme PM^2 est portable sur tous les environnements acceptés par Madeleine : BIP / Myrinet, SBP / Fast-Ethernet, Dolphin-SCI-driver / SCI, TCP / IP mais aussi PVM et MPI. PM^2 est fondé sur l'appel de procédure à distance, qui se traduit par l'exécution d'un *thread* distant. La taille de la pile des *threads* peut être manipulée pour faire face aux nombreux appels récursifs de certains programmes. L'ordonnancement des processus légers PM^2 est contrôlé à travers des priorités attribuées aux *threads* [Denneulin, 1997]. Les *threads* PM^2 peuvent migrer d'un site à un autre. Le coût de gestion des *threads* est faible, et l'ensemble est destiné au parallélisme massif et adaptatif. Il autorise la mise en place de diverses politiques de régulation de charge.

123. La version de PM^2 basée sur Madeleine est appelée PM^2 *High Perf.*

Conclusion. Comme nous l'avons expliqué dans la section précédente, les contraintes du modèle des réseaux associatifs, ainsi que les plate-formes envisagées pour ANET, nous conduisent à utiliser un support *multithread* pour cette bibliothèque. Notre choix s'est porté sur l'environnement PM^2.

PM^2 propose une interface de programmation réduite et une simplicité d'utilisation intéressante. Cet environnement est portable. Il obtient de très bonnes performances sur les réseaux Myrinet de grappe de PC, grâce à la bibliothèque Madeleine, qui utilise le protocole de communication BIP, optimisé pour ce support [Prylli et Tourancheau, 1997]. Enfin, PM^2 est prochainement disponible sur environnement SMP, ce qui permettra d'exécuter ANET à la fois sur réseau de stations de travail, et sur machines multi-processeurs.

8.2.2 L'environnement d'exécution PM^2 *

Nous donnons ici une rapide introduction à PM^2. De plus amples informations peuvent être trouvées dans [Namyst, 1997] ou [Namyst et Méhaut, 1995b].

Principe. Un programme PM^2 se compose d'un *module* (processus UNIX) par processeur[124] (figure 114). Chaque module possède le même code, mais les exécutions peuvent être différenciées en utilisant le numéro du module. L'utilisation de PM^2 repose sur la notion de *services* (fonctions C), et destinés à être exécutés par des processus légers. Ces services fournissent des mécanismes de désignation de fonction et d'empaquetage et de dépaquetage[125] des arguments.

On distingue trois types de services, appelés LRPC[126]. Un LRPC synchrone se comporte comme un appel de fonction normal, mais l'exécution de cette fonction est assurée par un *thread*, sur une autre machine. Le *thread* appelant est bloqué tant que le résultat ne lui a pas été envoyé. Un LRPC asynchrone n'attend pas le retour du résultat de la fonction : appelant et appelé poursuivent leur exécution en parallèle. Enfin, un LRPC à attente différée est un (double) LRPC asynchrone avec gestion d'une structure d'attente du résultat.

124. Dans la version SMP, un seul processus sera présent par machine, et non par processeur.

125. En anglais, *pack* et *unpack*.

126. *Lightweight Remote Procedure Call*, c'est-à-dire RPC léger (appel de procédure à distance léger).

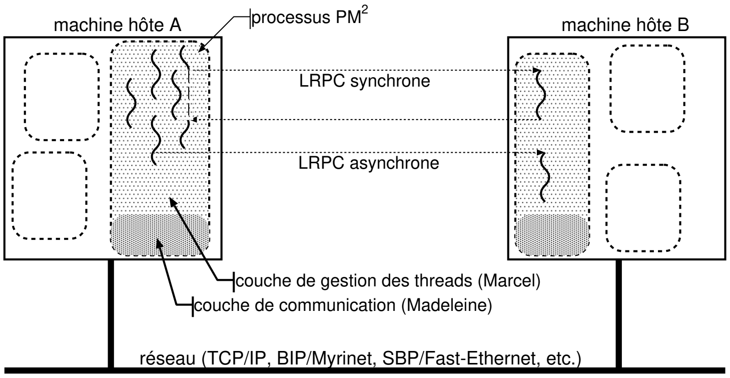


FIG. 114 – *Représentation imagée de PM^2 sur un environnement à mémoire distribuée.*

Utilisation. Le développement sous PM^2 se déroule en trois phases. La première consiste à spécifier les services : définition des paramètres d'entrée et de sortie et programmation des *souches*[127] qui correspondent à l'empaquetage des données au départ, et au dépaquetage des données à l'arrivée. Ensuite, il faut écrire la fonction réalisant le service proprement dit. Finalement, il reste à faire appel aux services ainsi définis. Cette dernière phase consiste à écrire le programme PM^2 qui lance les *threads* à distance, et réceptionne les résultats. Les déclarations des services sont facilités par des macro-commandes. Tout le code PM^2 doit être en langage C. Le code C suivant illustre la déclaration d'un service qui réalise l'addition de deux entiers à distance, par un *thread* lancé avec un LRPC à attente différée.

127. En anglais, *stub*.

```
 /*arguments d'entrée et de sortie */
LRPC_DECL_REQ(SOMME, int a, int b;)
LRPC_DECL_RES(SOMME, int a, int b;)

/* souche de lancement */
PACK_REQ_STUB(SOMME) /* départ */
  pm2_p_int(PM2_BY_COPY, &arg->a, 1);
  pm2_p_int(PM2_BY_COPY, &arg->b, 1);
END_STUB
UNPACK_REQ_STUB(SOMME) /* arrivée */
  pm2_u_str(PM2_BY_COPY, &arg->a, 1);
  pm2_u_str(PM2_BY_COPY, &arg->b, 1);
END_STUB

/* souche de retour */
PACK_RES_STUB(SOMME) /* départ */
  pm2_p_int(PM2_BY_COPY, &arg->c, 1);
END_STUB
UNPACK_RES_STUB(SOMME) /* arrivée */
  pm2_upkint(&arg->c, 1, 1);
END_STUB

/* code du service (addition)*/
BEGIN_SERVICE(SOMME)
  res.c = req.a + req.b;
END_SERVICE(SOMME)
```

```
/* declaration des structures */
LRPC_REQ(SOMME) *req;
LRPC_RES(SOMME) *res;
pm2_rpc_wait    *att;

/* Allocations. tmalloc permet de bloquer
la préemption des threads pendant le malloc. */
req = (LRPC_REQ(SOMME)*)
          tmalloc( sizeof(LRPC_REQ(SOMME)) );
res = (LRPC_RES(SOMME)*)
          tmalloc( sizeof(LRPC_RES(SOMME)) );
att = (pm2_rpc_wait*)
          tmalloc( sizeof(pm2_rpc_wait)   );

req->a = 5; req->b = 10; /* Initialisations */

/* Execution de l'addition a distance sur le module
(processus) numéro n. LRPC à attente différée. */
LRP_CALL( n, LRPC_SOMME, STD_PRIO, DEFAULT_STACK,
          &req, &res, &att);

/* Attente du retour du thread */
LRP_WAIT(att);
tprintf("résultat : %d", res->c);
```

8.3 Description d'ANET

Dans cette section, nous décrivons succinctement la bibliothèque ANET. De plus amples informations peuvent être trouvées dans [Sicard, 1998a] et [Ducourthial et Sicard, 1998], aussi laisserons-nous de côté certains aspects plus techniques pour nous concentrer sur les idées qui ont guidé ce travail[(b)].

8.3.1 Structure interne

Afin d'offrir au programmeur la possibilité de définir ses propres objets (pvars de pixels mais aussi de listes, d'ensembles, *etc.*), et ses propres opérateurs pour les manipuler, il nous a semblé important de développer ANET dans un langage objet. Nous décrivons dans cette section la structure objet de la bibliothèque. Elle est illustrée par la figure 115 page suivante, qui utilise le formalisme OMT[128].

128. *Object Modeling Technique.*

Hiérarchie objet. L'arborescence des classes C++ qui composent ANET est chapeautée par la classe `AnetObject` (figure 115 page ci-contre) qui stocke des constantes et méthodes communes à toute la hiérarchie, telle que la taille du réseau ou l'adresse des classes `Anet` et `Session` (qui sont uniques dans un programme ANET). La classe `session` gère les paramètres de la bibliothèque, décode les arguments du programme utilisateur, et sert d'interface avec PM^2 (lancement des modules, etc.).

(b) La conception de la bibliothèque ANET est un travail d'équipe. L'idée remonte aux développements de SAM en avril 1997. Le support *multithread* est apparu nécessaire après discussions en mai 1997 à Lausanne (RenPar 9). Les choses se sont précisées en décembre 1997, grâce à l'aide de J.-F. Méhaut, à Aussois (ICARE 97) puis à l'IEF et au LRI, où, avec Yves Denneulin, il nous présenta PM^2. L'idée de reprendre les bases de SAM, et d'utiliser un graphe réduit pour coder les données irrégulières est née de discussions avec A. Mérigot. Une fois la structure globale dessinée, il restait à en préciser les détails. N. Sicard a contribué à ce travail durant son stage de deuxième année de Magistère [Sicard, 1998b]. Nous avons été aidé par l'équipe de Lille (Y. Denneulin), et de Lyon (R. Namyst). Les réunions Grappes, Renpar 10, et des visites réciproques à Lille et à Orsay ont été l'occasion d'échanges fructueux. N. Sicard a résolu les principales difficultés techniques : l'interface entre les *threads* PM^2, destinés aux programmes C, et ANET, héritière de SAM, écrite en C++ ; la construction optimisée du graphe réduit ; le bon usage des templates (réduction de la taille de la bibliothèque compilée), limitant les recours à Perl, utilisé dans SAM, etc. Nous avons aussi bénéficié des conseils de P. Bosch et des idées d'A. Mérigot pour cette partie. Les tests d'ANET ont également été effectués durant le stage de N. Sicard. Nous avons été aidé par O. Richard pour l'utilisation de la plate-forme Myrinet.

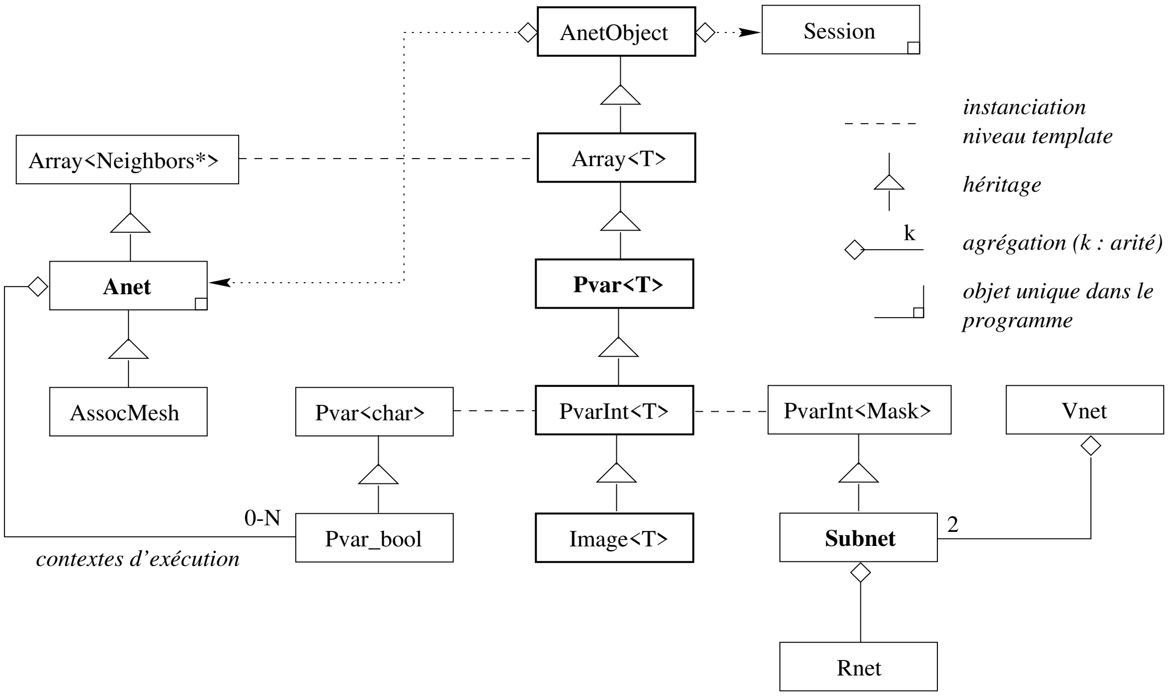


FIG. 115 – *Diagramme d'héritage d'*ANET *[Sicard, 1998a].*

La classe `Anet`. La classe `Anet` est une représentation du réseau associatif. Sa structure est donc celle d'un graphe orienté, de topologie quelconque. La bibliothèque ANET n'autorise l'utilisation que d'une seule instance de cette classe, ou d'une classe dérivée, dans un programme. En fait, cette limitation n'est pas d'ordre technique ; elle est imposée par respect de la philosophie du modèle. Tous les objets sont codés par des pvars, ayant la forme d'`Anet` (voir ci-après). En ce sens ANET s'apparente à un environnement data-parallèle basé sur une machine virtuelle d'instanciation, telle que C$\star$, DPCE ou Hyper C. La sémantique d'un calcul tel que $p += q$ est donc toujours clairement définie. Il n'y pas d'alignement des objets avant calcul, comme c'est le cas dans Fortran 90 ou HPF, ni de calcul uniquement sur les processeurs virtuels détenant à la fois p et q, comme dans Help ou Idole [Dekeyser et al., 1996]. Cette solution est plus simple à réaliser, mais est souvent considérée comme incompatible avec le codage des objets irréguliers. En fait, il n'en est rien : les contextes d'exécution permettent d'appliquer un calcul à une sous-partie d'une pvar, et les pvars de masques (`subnet`) permettent le codage de tout type d'objets (arbres, hypercubes, régions d'une image, etc.).

Codage d'`Anet`. La structure `Anet` doit permettre de représenter toute sorte de graphe orienté, tout en assurant un accès rapide aux informations (par exemple retrouver les valeurs des voisins d'un nœud), et un encombrement mémoire minimal.

Dans ANET, le réseau est codé par un tableau de nœuds ; un nœud est lui-même représenté par un tableau de références relatives à ses voisins : `Neighbors`. La référence relative du nœud u détenue par le nœud v est donnée par la différence des indices de u et de v. Le temps d'accès est comparable à celui d'un simple tableau. Mais lorsque les références relatives sont identiques pour plusieurs nœuds, un seul tableau peut être partagé par ces nœuds. `Anet` est donc un tableau de `Neighbors*`. Cela donne un codage compact.

Signalons que le codage des graphes «réguliers» est particulièrement efficace. C'est le cas des graphes de Cayley (telle que la maille, *cf.* annexe C), ou bien des grilles en 4- ou 8-connexité.

Initialisation d'`Anet`. Trois façons différentes permettent d'initialiser la topologie d'ANET : l'utilisation d'une topologie pré-définie dans la bibliothèque, telle que la maille avec `AssocMesh` (*cf.* figure 115), la programmation d'une sous-classe d'`Anet` et de son constructeur qui spécifie la topologie recherchée, ou la lecture d'un fichier de configuration. Un tel fichier présente une ligne par nœuds, qui indique ses références relatives. Il est possible de renvoyer vers une référence à un tableau défini précédemment dans le fichier ; il est aussi possible de sauter des lignes : les nœuds correspondants seront créés avec le tableau de référence dernièrement rencontré[(c)] .

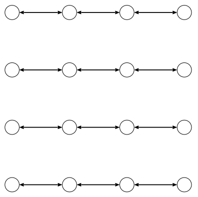


FIG. 116 – Codage de lignes par un `subnet`.

`Pvar.` Les variables parallèles du modèle des réseaux associatifs sont codés par la classe template `pvar`. Cela permet de définir des templates pour tout type d'objets (pvar de listes par exemple, pour l'algorithme d'arbre en profondeur, *cf.* § 7.4.4 page 140). Les opérateurs $=$, $+=$, etc. ont été redéfinis pour les pvars. On trouve aussi les opérateurs globaux de type `Global-plus` (β réduction des valeurs de la pvar, ici avec l'opérateur $+$). La classe template dérivée `PvarInt` propose en outre des opérateurs numériques, tels que les décalages, les `and` et `or` bit-à-bit, etc. La classe `Pvar_bool` est utilisée pour les tests de conditions, dans les `if`, mais aussi dans les contextes d'exécutions `WHERE(cond)`. Ceux-ci peuvent être imbriqués. Les `associations` sont toujours réalisées sur l'ensemble du graphe d'`Anet`, mais l'affectation du résultat est sujette aux contextes.

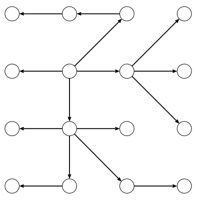


FIG. 117 – Codage d'un arbre par un `subnet`.

`Subnet.` La classe `Subnet` est une pvar entière de masques, qui sont des entiers de 8, 16 ou 32 bits selon le degré du graphe de base. Un masque de 8 bits code l'état masqué ou non-masqué de 8 connexions entrantes, selon la valeur 0 ou 1 de ses bits (§ 1.2.1 page 12). Toutes formes d'objets peuvent être codées par cette méthode, comme illustré par les figures 116, 117, 118 et 119. La machine virtuelle codée par `Anet`, et figée durant tout le programme, n'est donc pas incompatible avec le maniement des objets irréguliers. Le codage du `subnet` sous forme d'une pvar permet de lui appliquer tous les opérateurs des pvars, les affectations, les contextes, et même les `associations`. Un graphe virtuel `vnet` est composé de deux `subnets`, l'un pour les sommets, l'autre pour les arcs (§ 2.2.1 page 22).

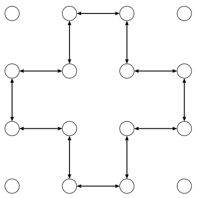


FIG. 118 – Codage de contours par un `subnet`.

Un graphe réduit (`rnet`) est associé à chaque `subnet`. Il s'agit d'une liste de CCs[129], elles-mêmes codées sous la forme d'une liste de CFCs[130], stockées dans l'ordre des étages du graphe réduit (qui est un DAG) : les CFCs sources sont en début de liste, les feuilles en fin de liste. À des fins d'optimisation, ces structures sont stockées sous forme de tableaux ; elles servent à la parallélisation des `associations`.

8.3.2 Parallélisation des `associations`

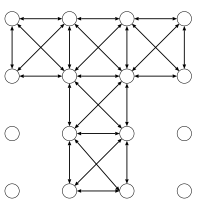


FIG. 119 – Codage de régions par un `subnet`.

Utilisation de PM^2. Un programme ANET est constitué d'un module PM^2 maître et de plusieurs modules esclaves. Le programme est exécuté par le *thread* principal du module maître ; celui-ci lance d'autres *threads* pour réaliser les calculs, qui restent en local ou partent sur les modules esclaves.

Dans la version actuelle d'ANET, un seul calcul est effectué à la fois[(d)]. Les calculs élémentaires de type $p+=q$ sont réalisés en local car ils sont trop simples pour qu'il y ait un quelconque avantage à les mettre en œuvre à distance. Actuellement, ils sont effectués par le *thread* principal, mais ils pourront être découpés en plusieurs *threads* quand PM^2 sera utilisé en mode SMP.

Parallélisation des `associations`. Le calcul des `associations` est parallélisé par les *threads*. Mais l'attribution des données à traiter aux *threads* respecte l'irrégularité. Ainsi, le graphe d'un sous-réseau (objet `subnet`) est traduit en un graphe réduit (objet `rnet`). Celui-ci code le graphe sous la forme d'une liste de CC, elles-mêmes constituées de listes de CFC (*cf.* 8.3.1). Puisqu'il n'y aura pas de communication entre composantes connexes, on associe un *thread* à chacune d'elles. Les *threads* appliquent le calcul de l'`association` en parcourant la liste des composantes fortement connexes. Les CFCs en amont sont en début de liste ; elles seront donc traitées avant leurs descendantes. Le respect de cet ordre est primordial car les CFCs filles ont besoin des résultats obtenus sur les CFCs

129. Composante Connexe.
130. Composante fortement connexe.

(c) La grille en 4-connexité est codée avec 5 lignes (voisinages centre, haut, bas, gauche et droite) et la maille avec 9 (4 voisinages en plus pour les angles, qui sont différents).

(d) Cette limitation est liée à un problème technique sur l'arrêt d'un *thread* PM^2. La gestion des dépendances de données sont cependant prévues dans ANET ; celles-ci sont contrôlées par des sémaphores stockées dans les objets pvars, et qui indiquent si l'objet est prêt ou en cours de traitement.

mères. Notons qu'en mémoire partagée, plusieurs *threads* pourraient être associés à une même CC, chacun partant d'une CFC source dans le graphe réduit.

Le calcul dans chaque CFC dépend du type de l'`association`. Pour les `associations`, ou les `direct-associations` utilisant un s-opérateur tel que min, les nœuds de la CFC sont parcourus les uns à la suite des autres pour calculer l'unique résultat, qui sera ensuite distribué à tous les nœuds. Pour une `direct-association` utilisant un r-opérateur $\triangleleft$, le calcul est actuellement effectué en itérant jusqu'à stabilité le calcul local de chaque nœud :

Algorithme 29 Calcul de la $\triangleleft$-`direct-association`

répéter
  pour tous les nœuds de la CFC (ordre des indices) **faire**
    calcul local avec l'opérateur $\triangleleft$ et les valeurs des antécédents.
  fin pour
jusqu'à ce que tous les calculs soient stables.

Cet algorithme a une complexité au pire cas de N^2 pour N nœuds[e]. En effet, tout r-opérateur étant basé sur un s-opérateur, il existe des nœuds, appelés minima, qui détiennent une plus petite valeur de toutes celles de la composante (l'ordre n'est pas forcément total). Les minima sont stables ; leur stabilité se répand dans le réseau en au pire N itérations (cas d'une chaîne avec un seul minima à une extrémité), et une itération est composée de N calculs locaux. L'algorithme se termine uniquement si l'opérateur est idempotent. Lorsque la bibliothèque est compilée avec l'option `ENABLE_CONTROL`, elle propose des fonctions permettant de vérifier la nature d'un réseau (DAG, arbre), et un contrôle est mis en place durant le calcul.

À terme, l'algorithme 29 pourra être optimisé. Si les nœuds de la CFC sont parcourus en suivant les liens du graphe et non les indices des sommets, on augmente les chances d'effectuer un calcul local qui modifie le résultat du nœud, ce qui réduit d'autant le temps de convergence. En outre, partir de ces minima garantit que les premiers calculs locaux ne seront pas inutiles. Pour les grandes CFCs, et suivant la complexité du r-opérateur, le temps de recherche des minima peut s'avérer bénéfique comparé au nombre d'itérations nécessaires à la stabilisation. Notons que cette recherche correspond au calcul d'une `association` (ou d'une `direct-association`) avec le s-opérateur concerné. Signalons aussi que sur une plate-forme multiprocesseurs à mémoire partagée, et avec PM^2 en mode SMP, il pourra être intéressant d'affecter plusieurs *threads* par composantes fortement connexes, chacun partant, par exemple, d'un nœud-minima. À l'instar des itérations asynchrones, la non-coordination des calculs et des communications dans le modèle rendra inutile les synchronisations dans ce type calcul, partant de plusieurs «sources» au sein d'une même composante.

Construction du `rnet`. Le graphe réduit est la structure clé qui permet de s'accomoder de l'irrégularité des données lors de la parallélisation. Sa construction, et l'optimisation des structures de données associées, ont donc été particulièrement soignées. Afin d'économiser de la mémoire, seuls les nœuds ayant un voisin dans le `subnet` se retrouve dans le `rnet` correspondant. Les autres sont donc traités en local (suivant le type d'`association`, ils doivent ou non prendre la valeur de l'élément neutre de l'opérateur). La réduction du sous-réseau est basée sur l'algorithme de Trémeaux-Tarjan (*cf.* § 7.2.4). Il s'agit d'une adaptation aux particularités d'ANET, qui utilise deux passes afin d'optimiser la mémoire.

L'algorithme de Trémeaux-Tarjan repose essentiellement sur la gestion d'une pile de nœuds, d'une liste de listes chaînées de composantes fortement connexes, et d'une liste de liens entre les nœuds appartenant à des composantes distinctes. Durant le calcul, un nœud est caractérisé par son état : *non exploré*, *actif* ou *traité*. Lorsqu'il est actif on connaît à tout instant son index, sa position dans la pile, le `descendant` dans le graphe dont il est issu (on remonte les arcs du graphe dans cet algorithme), son antécédent dans le sous-réseau qui est le plus bas dans la pile (`min`), et le prochain antécédent à explorer. L'algorithme consiste à parcourir l'ensemble des nœuds du graphe et à les traiter selon leur état. Si le nœud courant est *non exploré* et possède au moins un antécédent, il est empilé

(e) La complexité intrinsèque (indépendamment de toute implémentation) de la `direct-association` est étudiée au § 6.1.2 page 105.

et devient actif : on suppose qu'on commence le parcours d'une nouvelle composante connexe. Cet antécédent devient le nœud courant et on boucle tant que la pile n'est pas vide :

Algorithme 30 Réduction d'un `subnet`

pour u décrivant l'ensemble des nœuds **faire**
  si l'état de u n'est pas *traité* et que u a au moins un antécédent dans le graphe **alors**
    Empiler u ; l'état de u est *actif* ; `min[`u`]` $\leftarrow u$.
    tant que il reste un lien entrant non exploré au nœud courant u **faire**
      ▶ *Remontée dans le réseau.*
      $v \leftarrow$ extrémité d'un de ce liens.
      suivant l'état de v **faire**
        actif : on met à jour `min[`u`]` ▶ *v est dans la pile.*
        traité : v appartient à une autre composante fortement connexe. On ajoute donc un lien entre les deux nœuds et on concatène les listes auxquelles appartiennent les deux composantes fortements connexes (si ce n'est pas la même).
        non exploré : v est placé au sommet de la pile et devient le nœud courant ($u \leftarrow v$).
      fin suivant
    fin tant que
    ▶ *Tous les liens entrants du nœud courant ont été explorés.*
    si u est au bas de la pile ou u = `min[`u`]` **alors**
      ▶ *Obtention d'une composante fortement connexe.*
      u «ferme» une composante fortement connexe dont le premier nœud est le haut de la pile ; tous les nœuds empilés après lui sont dépilés et stockés dans la composante fortement connexe courante. Leur état est positionné à *traité*.
    sinon
      ▶ *Descente dans le réseau.*
      `min[`u`]` $\leftarrow$ min (`min[`u`]`, `min[descendant[`u`]]`)
      $u \leftarrow$ `descendant[`u`]`
    fin si
  sinon
    u est une composante fortement connexe a lui tout seul ; on ne le stocke pas dans le graphe réduit.
  fin si
fin pour

Mais la structure issue de ce calcul n'est pas encore satisfaisante. Un second parcours est nécessaire pour compléter les informations des liens inter-composantes, construire la structure définitive du graphe réduit, et éliminer les données temporaires utilisées par la construction. La structure temporaire nécessaire au déroulement de cet algorithme est un tableau alloué en une seule fois pour n'utiliser qu'un seul appel système. Il y a donc sur-consommation temporaire de mémoire, au cas où tous les nœuds ne se retrouvent pas dans le graphe réduit. La structure finale est, quant à elle, optimisée. Elle est composée d'un tableau pour les CC (qui contient uniquement leur taille), d'un tableau pour les composantes fortement connexes (qui contient leur taille et le nombre de liens depuis une CFC en amont), d'un tableau d'indices (`long int`) de nœuds et d'un tableau de liens (indice de la CFC en amont, du nœud de départ du lien dans cette CFC et de celui d'arrivée dans la CFC en aval).

Chaque `association` sur le `subnet` net_G représentant le sous-réseau G requiert au préalable la construction du `rnet` associé. Mais le `rnet` n'est à reconstruire que lorsque le `subnet` a été modifié ; la structure peut donc servir à plusieurs `associations`. À terme, la réduction des `subnets` sera réalisée par un *thread* en tâche de fond[f] . De cette manière le `rnet` sera prêt au moment de l'exécution de l'`association`, sauf à modifier le `subnet` juste avant l'appel de la primitive. L'intérêt est évident en mode SMP ; sans le mode SMP, cette technique pourrait permettre de déporter plus de calcul d'`associations`, le processeur du module maître étant occupé pendant ce temps-là.

Lancement des *threads*. Des *threads* sont créés à chaque calcul d'`association` par un LRPC asynchrone (*cf.* § 8.2.2). Chacun d'eux gère une ou plusieurs composantes connexes, selon le paramètre `ANET_Q`. Ce paramètre permet d'ajuster le nombre de nœuds minimal associé à un *thread*.

(f) C'est un problème technique concernant l'arrêt d'un *thread* en cours de calcul qui nous a empêché de profiter de cette optimisation.

Le paramètre `ANET_N` contrôle la taille des *threads* envoyés à distance. `ANET_P` désigne quant à lui le nombre de modules PM^2 créés. L'algorithme de lancement des *threads* est donné ci-dessous. Le mécanisme général d'attribution des données aux *threads* PM^2 est résumé par la figure 120.

Algorithme 31 Lancement des *threads*

si le `subnet` a été récemment modifié ou n'a jamais été construit **alors**
  Construction du `rnet` (*cf.* algorithme 30 page précédente)
fin si
Initialisation de la pvar résultat.
Affectation des nœuds sans voisins : élément neutre si `exclusive-association`, valeur de la pvar sinon.
Allocation mémoire pour les paramètres d'entrée et de sortie des LRPC ; allocation des structures d'attente des LRPC.
tant que une CC n'a pas été traitée **faire**
  Les paramètres d'entrée du $i^{\text{ème}}$ *thread* sont mis à 0. ▶ *i compte les threads.*
  tant que le nombre total de nœud à traiter par le *thread* courant est inférieur à `ANET_Q` et qu'il reste une CC non traitée **faire**
    Allocation d'un tableau de données pour le *thread.*
    Copie des valeurs de la pvar selon les indices des nœuds indiqués par la composante connexe du graphe réduit (« *pack* ANET »).
  fin tant que
  si le nombre total de nœuds du *thread* est inférieur à `ANET_N` **alors**
    Le *thread* est confié au module maître (`selfprocess`).
  sinon
    Le *thread* est confié au p$^{\text{ème}}$ module esclave.
    $p \leftarrow p + 1 \bmod (\texttt{ANET\_P} - 1)$
  fin si
  $i \leftarrow i + 1$
fin tant que
Lancement des *threads* attribués à des modules esclaves avec des `LRP_CALL`s.
Lancement des *threads* attribués au module maître avec des `LRP_CALL`s.
Attente de la terminaison LRPC à attente différée avec des `LRP_WAIT`s.
Recopie des données de la structure de retour des LRPC dans la pvar résultat (« *unpack* » ANET).

Le paramètre `ANET_N` a été imaginé, dans un premier temps, pour simplifier — et reporter — le travail d'équilibrage de charge : seuls les *threads* les plus importants sont envoyés à distance, car c'est pour eux que la déportation est la plus rentable. À terme, il est clair qu'une véritable politique de répartition de la charge sur les machines présentes, incluant des paramètres caractérisant le réseau et l'opérateur, (*cf.* § 8.4.6), est nécessaire [(g)].

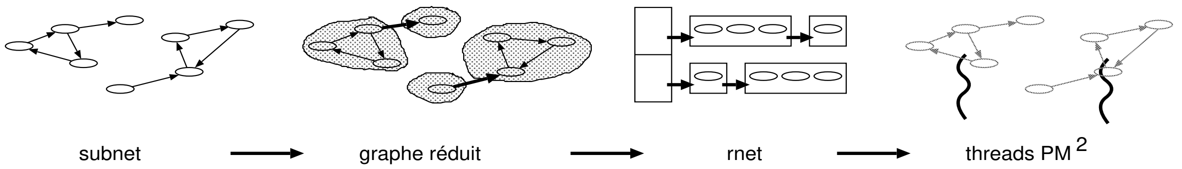


FIG. 120 – *Attribution des données aux* threads *PM^2. Le `subnet` est réduit sous forme d'un graphe réduit, stocké dans la structure `rnet` (liste de composantes connexes, elles-mêmes représentées par une liste de composantes fortement connexes). L'association des* threads *PM^2 aux composantes connexes est contrôlée par le paramètre `ANET_Q`. Le site d'exécution d'un* thread *est contrôlé par le paramètre `ANET_N`.*

(g) En fait, le paramètre `ANET_N` s'est révélé être un tout ou rien trop délicat à contrôler : soit la majeure partie des *threads* partaient à distance, soit la plupart des *threads* restaient en local. La charge n'était donc pas équilibrée, et le manque de pertinence du paramètre `ANET_N` nous a gêné dans les tests d'évaluation d'ANET. Le déséquilibre de la charge aurait été moins gênant si, comme nous le prévoyions initialement, la construction des `rnet`s était préventive, et les `associations` s'exécutaient en parallèle (*cf.* notes **(d)** page 162 et **(f)** page précédente).

8.4 Expérimentations d'ANET

Dans cette section, nous présentons les tests menés avec la bibliothèque ANET, puis nous en analysons les résultats.

La bibliothèque ANET est une généralisation de SAM ; il importait donc de vérifier que notre généralisation n'était pas excessivement coûteuse (*cf.* § 8.4.2). L'envoi d'un *thread* à distance nécessite un formatage des données dans ANET, qui s'ajoute aux *packs* de PM^2 ; il nous fallait donc vérifier que cette couche supplémentaire n'était pas trop pénalisante (*cf.* § 8.4.3). ANET permet d'utiliser les r-opérateurs ; nous avons donc testé l'intérêt de ces opérateurs, notamment sur la segmentation de Voronoï (*cf.* § 8.4.4). La parallélisation des primitives de calcul dans ANET réside dans l'attribution de *threads* aux composantes connexes, et à l'exécution à distance de ces *threads* ; nous avons donc mesuré le coût de l'envoi à distance sur réseaux Ethernet et Myrinet (*cf.* § 8.4.5), ainsi que l'influence de l'opérateur sur l'intérêt de cette parallélisation (*cf.* § 8.4.6). Pour finir, nous discutons du paramétrage de la bibliothèque.

8.4.1 Conditions expérimentales

Environnement matériel. Les tests ont été effectués sur deux environnements différents. Nous avons utilisé le réseau Ethernet 10 Mb/s de l'IEF, ainsi que trois PC sous Linux. L'analyse de nos résultats prend en compte l'hétérogénéité de ces machines : Pentium 133 MHz 32 Mo de RAM EDO, Pentium 166 MHz 32 Mo de RAM EDO, et bi-Pentium II 400 MHz 64 Mo de SDRAM.

Nous avons aussi bénéficié de la plate-forme Myrinet du LRI, dont nous avons utilisé deux processeurs bi-Pentium II 300 MHz 384 Mo de SDRAM, reliés par un connecteur Myrinet équipé de 1 Mo de SDRAM. Utilisé avec la bibliothèque de communication BIP [Prylli et Tourancheau, 1997], le réseau admet une latence de 6 μs et le débit maximum est de 127 Mb/s.

Dans l'analyse des résultats, il faut prendre en compte les conditions réelles des expériences dans un environement qui n'est pas toujours dédié aux tests (malgré nos précautions). Nous n'avons pas multiplié les expériences similaires car nous avons estimé qu'en général les artefacts seraient gommés par la quantité de mesures différentes. Signalons cependant que les temps de calcul trop petits sont difficilement interprétables.

Configurations logicielles. La bibliothèque ANET a été utilisée en *mode séquentiel* (sans PM^2), en *mode local* (avec PM^2 mais sur un seul processeur), et en *mode parallèle* (avec PM^2 et sur plusieurs machines). La version de PM^2 utilisée est *HighPerf* ; le compilateur est `gcc 2.7`.

Nous avons utilisé le programme de la découpe de Voronoï ou de la fusion (dans ce cas, toujours en partant des régions de un pixel, *cf.* 2.1.2 page 20) ; les paramètres de la segmentation ont toujours été les mêmes pour toutes les exécutions, et pour toutes les images. Nous avons sélectionné quatre images, que nous pensons représentatives ; elles sont présentées en annexe (*cf.* B.1 page 191). L'image *forme* est très simple, *icoron* présente des nuances de dégradé très fines, *imuscle* présente un plus grand contraste et doit conduire à de petites régions, tandis que *roks* présente un contraste moyen et devrait conduire à des régions de tailles variables. Ces images ont été utilisées au format 64×64, 128×128 et 256×256 pixels.

8.4.2 Évaluation du coût de la généralisation

But. La bibliothèque ANET a été conçue à partir de SAM, dont elle reprend un certain nombre de concepts. Cependant, ANET est plus générale. En effet, contrairement à SAM qui ne traite que des mailles, la topologie du graphe de base est quelconque et paramétrable (*cf.* § 8.3.1). Aussi un certain nombre d'optimisations dans l'accès et la manipulation des données, qui sont présentes dans SAM, ne peuvent subsister dans ANET. En outre, ANET propose une gestion de l'irrégularité (des données et des traitements à y appliquer) à travers la manipulation du graphe réduit. Celui-ci code les objets de formes diverses, et permet de paralléliser les `associations` en respectant cette irrégularité : la composante connexe est l'unité des données attribuées aux *threads*. La construction de ce graphe pourrait s'avérer coûteuse, malgré les optimisations que nous avons mises en place. En contrepartie, lorsque plusieurs `associations` sont réalisées sur le même sous-graphe, le graphe réduit correspondant est réutilisé dans ANET, tandis que la structure est reconstruite[(h)] dans SAM.

La généralisation de la topologie de base, et le codage de l'irrégularité des données par un graphe réduit sont des caractéristiques essentielles de la bibliothèque. Il est nécessaire de s'assurer qu'elles ne grèvent pas trop ses performances. Nous cherchons à évaluer ce surcoût en comparant ANET à SAM.

Méthode. Les mesures ont été faite sur la fusion puis sur la découpe[(i)] de Voronoï , sur un Pentium II à 400 MHz, avec 64 Mo de mémoire. ANET a été utilisée en mode séquentiel, c'est-à-dire qu'il n'y a ni *thread* ni de communication. Ainsi les différences de temps qui seront constatées proviendront des structures de données différentes (les algorithmes des `associations` sont semblables). Les expérimentations ont été faites sur les quatre images tests, avec trois tailles différentes.

Résultats. Les résultats sont présentés dans les tableaux des figures 121 et 122. Le coefficient indiquant le surcoût d'ANET est calculé en divisant le temps obtenu avec ANET par celui obtenu avec SAM.

type d'image	forme			icoron			imuscle			roks		
taille	64	128	256	64	128	256	64	128	256	64	128	256
SAM (s)	0,308	1,30	5,55	0,934	4,31	19,4	1,98	12,6	64,9	1,60	12,3	63,7
ANET séq. (s)	0,472	2,10	9,24	1,321	6,40	28,0	2,56	12,7	81,3	1,51	18,2	33,6
surcoût	1,53	1,61	1,66	1,41	1,485	1,44	1,29	1,01	1,25	0,94	1,48	0,53

FIG. 121 – *Comparaison de* ANET *et de* SAM *sur la fusion de Voronoï.*

type d'image	forme			icoron			imuscle			roks		
taille	64	128	256	64	128	256	64	128	256	64	128	256
SAM (s)	10,3	56,7	367	8,68	45,0	230	8,60	52,6	277	8,55	53,7	367
ANET séq. (s)	10,5	88,1	455	6,98	41,0	230	8,77	52,3	341	9,78	57,6	306
surcoût	1,02	1,55	1,24	0,80	0,91	1,00	1,02	0,99	1,23	1,14	1,07	0,83

FIG. 122 – *Comparaison de* ANET *et de* SAM *sur la découpe de Voronoï.*

Analyse. La fusion de Voronoï est parfois plus avantageuse avec ANET qu'avec SAM mais, en règle générale, SAM est plus rapide. Cependant le surcoût d'ANET n'est pas excessif : la moyenne des surcoûts des 12 fusions est 1,30 (écart type de 0,3). La comparaison est plus avantageuse pour ANET avec la découpe de Voronoï. Cela s'explique par le fait que, dans cet algorithme, plusieurs `associations` sont réalisées sur le même sous-graphe ; les parcours du graphe (constructions) sont économisés dans ANET. La moyenne des 12 découpes est de 1,06 (écart type de 0,2). En conclusion, malgré le peu de mesures (et une dispersion de l'ordre de 20 %), on peut dire que le surcoût de la généralisation est acceptable.

8.4.3 Évaluation du coût du formatage des données dans ANET

But. Au moment de l'envoi à distance des *threads*, il est nécessaire d'attribuer à chaque *thread* les données dont il a besoin. Il s'agit de passer de la représentation des données sous forme de pvar

(h) Il s'agit d'un parcours du réseau en suivant les liens de chaque nœud. Il n'y a pas, en réalité, de reconstruction de structure. Mais ce parcours correspond à une reconstruction dans le temps du réseau ; le coût en temps est comparable.

(i) Il s'agit de l'algorithme présenté au chapitre 2 et qui n'utilise que des s-opérateurs, car SAM ne permet pas d'utiliser des r-opérateurs.

(pour les valeurs) et de `rnet` (pour la définition des objets au sein de la pvar) à un tableau prêt à l'envoi. Suivant la terminologie de PM^2, nous appelons cette opération le *pack* ANET. Un *pack* ANET correspond essentiellement à une allocation de mémoire, suivie d'une recopie des données. Pour chaque *thread*, un tableau est réservé et les données y sont copiées. Le coût du *pack* ANET dépend donc du nombre de données par *thread*, et du nombre de *threads*. Il dépend donc d'`ANET_Q` (nombre minimum de pixels par *threads*). Notons que la version actuelle d'ANET effectue un *pack* même lorsque les *threads* restent en local, alors que cette opération est inutile puisque les *threads* d'un processus se partagent la mémoire de ce dernier.

Nous cherchons ici à évaluer le coût du *pack* d'ANET, qui s'ajoute à ceux de PM^2 lors des envois à distance.

Méthode. Nous avons comparé les temps d'execution entre ANET en mode séquentiel et ANET en mode parallèle sur la même machine que précédemment. Le programme utilisé est la fusion de Voronoï, appliquée à deux types d'images. En positionnant `ANET_Q` au nombre de pixels (valeur maximale) dans la version parallèle, un seul *thread* est créé à chaque `association` (car on lui adjoint des composantes connexes à hauteur de `ANET_Q` pixels, c'est-à-dire ici toute l'image). Puisque le *pack* ANET est réalisé même lorsque les *threads* restent en local, la différence que nous constaterons entre les deux versions sera essentiellement due à la création d'un *thread* par `association`, et d'un `malloc` pour allouer de la mémoire.

Résultats. Les résultats sont présentés dans la figure 123. Le surcoût est calculé en faisant le rapport du temps ANET parallèle (ANET PM^2) sur ANET séquentiel (ANET séq.).

type d'image	forme			icoron		
taille	64	128	256	64	128	256
ANET seq. (s)	0,467	2,08	9,24	1,32	6,41	28,0
ANET PM^2 (s)	0,539	2,32	10,44	1,56	7,12	30,8
surcoût	1,15	1,11	1,30	1,18	1,11	1,1

FIG. 123 – *Évaluation du pack d'*ANET.

Analyse. Le programme de la fusion de Voronoï conduit à un facteur de ralentissement moyen de 1,16, avec une faible dispersion pour les 6 mesures (écart type de 0,07). Le coût du *pack* ANET est donc acceptable.

Ce bon résultat doit cependant être modéré. En effet, plus `ANET_Q` sera petit, moins il y aura de données à allouer lors de chaque `malloc`, mais plus il y aura de `mallocs`. Le coût global du *pack* ANET aura donc tendance à augmenter avec le nombre de `malloc` à réaliser, plus important lorsqu'`ANET_Q` diminue. Enfin, notons que les valeurs mesurées sont liées au nombre de `direct-associations` dans l'algorithme, et que le coût du *pack* ANET pourrait varier d'un programme à un autre.

8.4.4 Intérêt des r-opérateurs

But. Les r-opérateurs permettent de remplacer plusieurs itérations par un seul calcul asynchrone. Ainsi, les itérations de l'algorithme 2 page 25, nécessaires pour obtenir les régions de Voronoï avec des s-opérateurs, ont été remplacées par une seule minc-`direct-association` dans l'algorithme 26. Si l'intérêt d'une telle transformation est certain au niveau de l'écriture des algorithmes, on peut attendre d'autres avantages de la mise en œuvre. En effet, outre la suppression du contrôle des itérations, le grossissement du grain du calcul data-parallèle permet d'attribuer plus de travail aux *threads*, et donc de mieux rentabiliser leur création (ainsi que les *packs* ANET). De même, l'envoi à distance d'un *thread* pour le calcul d'une `direct-association` utilisant un r-opérateur présentera un meilleur ratio calcul / communication. Nous évaluons ici l'impact des r-opérateurs sur le nombre de création de *threads*. Les autres propriétés sont évaluées au § 8.4.6.

Méthode. Nous avons mesuré le temps d'exécution sur la découpe de Voronoï avec d'une part la bibliothèque SAM (s-opérateurs uniquement), et d'autre part avec ANET, en utilisant les s-opérateurs puis les r-opérateurs. Les deux programmes sont donnés en annexe aux § A.1 page 185 et A.2 page 187. Nous avons utilisé cinq types d'images et, pour chacune d'elles, trois tailles. Les tests ont été réalisés en séquentiel. On ne mesure ici que le remplacement des itérations à base de s-opérateurs par le calcul d'une seule `direct-association` utilisant un r-opérateur.

Résultats. Les résultats sont présentés dans la figure 124. Les deux dernières colonnes donnent le surcoût de SAM et de ANET utilisant les s-opérateurs (et les itérations) par rapport à ANET utilisant les r-opérateurs.

Taille image	**Type image**	**SAM**	**ANET** s-opérateurs	**ANET** r-opérateurs	**SAM** / **ANET** r-opérateurs	**ANET** s-opérateurs / **ANET** r-opérateurs
Images 64 × 64 10 racines	aerial	8,32	9,21	1,09	7,6	8,5
	forme	10,32	10,48	1,55	6,7	6,8
	icoron	8,68	6,98	1,44	6,0	4,9
	imuscle	8,60	8,77	1,16	7,4	7,6
	roks	8,55	9,78	1,23	7,0	8,0
	moyenne	*8,89*	*9,04*	*1,29*	*6,9*	*7,2*
Images 128 × 128 40 racines	aerial	46,78	49,65	5,39	8,7	9,2
	forme	56,75	88,14	7,20	7,9	12,2
	icoron	45,04	41,05	5,53	8,1	7,4
	imuscle	52,57	52,28	5,47	9,6	9,6
	roks	53,69	57,64	5,92	9,1	9,7
	moyenne	*50,96*	*57,75*	*5,9*	*8,7*	*9,6*
Images 256 × 256 160 racines	aerial	268,49	316,96	27,08	9,9	11,7
	forme	367,08	454,82	42,65	8,6	10,7
	icoron	229,82	230,40	28,96	7,9	8,0
	imuscle	277,25	340,77	31,74	8,7	10,7
	roks	366,66	301,50	30,25	12,1	10,0
	moyenne	*301,86*	*328,89*	*32,14*	*9,4*	*10,2*
				moyenne	*8,4*	*9,0*

FIG. 124 – *Intérêt des* r*-opérateurs (les temps sont exprimés en seconde).*

Analyse. Le temps de la découpe de Voronoï utilisant les r-opérateurs est de 8,4 fois inférieur à celui obtenu avec SAM, en moyenne sur 15 mesures (dispersion raisonnable de 17 %). De même, il est de 9 fois inférieur au temps obtenu avec ANET et les s-opérateurs (dispersion raisonnable de 21 %).

Ces résultats peuvent paraître surprenant de prime abord, car la complexité de la `direct-association` utilisant un r-opérateur est en $O(N^2)$ (au pire cas, *cf.* § 8.3.2 page 162), tandis qu'elle est en $O(N)$ pour un s-opérateur (que ce soit pour une `direct-association` ou une `association`). Deux raisons majeures expliquent ces résulats. Premièrement, les régions (*i.e.*, N) ne sont pas suffisament grandes pour que la différence de complexité des calculs devienne primordiale. Deuxièmement, le r-opérateur minc permet de construire les régions de Voronoï en un seul calcul asynchrone (*cf.* algorithme 26 page 144), tandis qu'un grand nombre d'itérations sont nécessaires pour effectuer le même travail avec les s-opérateurs, dans un programme comportant plus de synchronisations (*cf.* chapitre 2 page 19). Nous détaillons maintenant ces deux raisons.

La différence de complexité des deux calculs est certainement exagérée. Il faut obligatoirement N étapes pour mettre à jour N nœuds, et on peut estimer constante la complexité du calcul pour un s-opérateur. Par contre, la complexité en $O(N^2)$ du calcul des r-opérateurs est un pire cas, qui doit assez largement surestimer la complexité moyenne, observée sur le grand nombre de calculs effectués (autant que d'itérations dans la découpe de Voronoï). En outre, les complexités ont évolué en fonction de la taille des régions. Puisqu'`ANET_Q` était constant durant tous les essais, le nombre de *threads* créés à chaque `association` a pu augmenter avec la taille de l'image. Mais une grande image conduit, *a priori*, à de grandes régions, et l'estimation de l'évolution du nombre de régions en fonction de la taille de l'image (à `ANET_Q` constant) n'est pas simple. Cependant les mesures montrent que la combinaison de ces différents facteurs est favorable aux r-opérateurs : on constate, avec les moyennes partielles, calculées à taille d'images constante, que l'intérêt des r-opérateurs est croissant avec le nombre de pixels.

La construction des régions de Voronoï avec des s-opérateurs nécessite des itérations, dont le nombre moyen (noté k par la suite) est difficile à évaluer, et dont chacune contient plusieurs `associations`. Les r-opérateurs grossissent le grain du calcul data-parallèle, et permettent de calculer les régions de Voronoï en une seule `direct-association`, mais de complexité plus importante. On compare donc k itérations en $O(N)$ avec une seule itération en $O(N^2)$ (qui est un pire cas certainement éloigné de la complexité moyenne). Les mesures indiquent que le nombre d'itérations k est suffisament important pour qu'un seul calcul en $O(N^2)$ soit plus avantageux que k calculs en $O(N)$. Notons aussi que la taille du problème (ici les images) ne peut être agrandi indéfiniment, et que la taille des régions (N) ne peut donc croître indéfiniment non-plus. Les images 256×256 sont des images de taille standard. En outre, il est fort probable que le nombre d'itérations k augmente lui aussi avec la taille de l'image.

En conclusion, lorsque chaque instruction data-parallèle (ici des `associations`) conduit à créer des *threads* (ce qui implique un *overhead*), il existe une plage dans laquelle le grain du calcul est intéressant : ni trop faible pour pouvoir remplacer plusieurs itérations (le travail effectué est assez important), ni trop gros pour être plus avantageux que plusieurs instructions data-parallèles simples (la complexité n'est pas trop forte). Les r-opérateurs conduisent à un grain de calcul qui se trouve visiblement dans cette plage : ils permettent d'effectuer un travail important (ici, la détermination des régions de Voronoï), et ils conduisent à un calcul de complexité acceptable (au pire $O(N^2)$). Ils permettent donc de paralléliser plus facilement les programmes data-parallèles, en créant des *threads* à chaque instruction, ce qui permet un contrôle dynamique précis de l'irrégularité (des ajustements sont possibles à chaque instruction). Nous évaluons l'intérêt des r-opérateurs sur l'exécution à distance des *threads* au § 8.4.6.

8.4.5 Évaluation du coût de l'envoi à distance

But. La parallélisation des `associations` dans ANET est effectuée en exécutant des *threads* sur des processeurs différents, ce qui implique des créations de *threads* suivies de communications. Sur machine SMP, cette parallélisation devrait être efficace, car il n'y a pas de communication et seul le coût de création d'un *thread* peut nuire à la parallélisation du programme[j]. Par contre, sur machine à mémoire distribuée, les communications diminuent l'intérêt de la parallélisation, et l'avantage est lié à la proportion du temps passé en calcul par rapport aux communications. Nous cherchons ici à évaluer le coût des communications d'un programme ANET par rapport à son temps d'exécution.

Méthode. La proportion calcul / communication est liée aux performances du réseau ; nous avons donc mené des expériences sur deux types de réseaux : Ethernet (TCP/IP) et Myrinet (BIP). Nous avons utilisé deux processeurs identiques sur Myrinet, mais nous n'avions pas de processeurs semblables sur Ethernet : le processeur du module maître était un Pentium 166 MHz, tandis que celui de

(j) Nous ne disposions par encore de la version SMP de PM^2 ; aussi n'avons nous pas pu mener ces essais.

l'esclave était un Pentium 133 MHz.

Nous avons utilisé la bibliothèque ANET en mode parallèle ; `ANET_Q` a été fixé à 500 (au moins 500 pixels par *threads*). Nous avons utilisé l'algorithme de la fusion de Voronoï avec deux types d'images différentes, et, pour chacune d'elles, trois tailles.

La proportion des communications par rapport aux calculs dans un programme est fortement liée à l'équilibrage de charge entre les différents sites. Actuellement ANET ne dispose pas d'une politique d'équilibrage de charge efficace (*cf.* § 8.3.2) ; nous avons utilisé le paramètre `ANET_N` (qui détermine à partir de combien de pixels le *thread* s'exécute à distance) pour évaluer les communications dans le pire cas.

Le paramètre `ANET_N` a été réglé à 1 (tous les *threads* partent à distance), puis au nombre de pixels de l'image plus 1 (tous les *threads* restent en local). Lorsque tous les *threads* partent à distance au moment de leur création, pour chacun il y a : un *pack* ANET, un *pack* PM^2, la communication, le *unpack* PM^2, le traitement, le *pack* PM^2 du retour, la communication, le *unpack* de PM^2 et le *unpack* d'ANET[(k)]. Lorsque tous les *threads* restent en local, chacun d'eux nécessite un *pack* ANET, le traitement, et le *unpack* d'ANET[(l)]. La différence de temps entre les deux vient donc des *packs* / *unpack* de PM^2, et du temps d'un aller-retour, multiplié par le nombre de communications dans le programme, lui-même lié au nombre d'`associations` (qui dépend de l'image utilisée).

Résultats. Les figures 125 et 126 récapitulent les résultats. Nous avons calculé le surcoût en faisant le rapport entre le temps de la version «tout à distance» (`ANET_N` = nombre de pixels + 1, tous les *threads* partent à distance) et la version «tout en local» (`ANET_N` = 1).

type d'image	forme			icoron			
taille	64	128	256	64	128	256	*moyenne*
tout en local (s)	1,97	7,93	40,15	5,24	23,43	147,9	
tout à distance (s)	6,28	21,56	90,19	16,86	63,77	305,4	
surcoût	3,19	2,72	2,25	3,22	2,72	2,01	*2,68*

FIG. 125 – *Coût de l'envoi à distance sur réseau Ethernet, fusion de Voronoï.*

type d'image	forme			icoron			
taille	64	128	256	64	128	256	*moyenne*
tout en local (s)	0,712	3,29	16,88	2,07	10,07	50,49	
tout à distance (s)	1,000	3,45	17,53	2,34	10,29	52,07	
surcoût	1,4	1,05	1,04	1,13	1,02	1,03	*1,11*

FIG. 126 – *Coût de l'envoi à distance sur réseau Myrinet, fusion de Voronoï.*

Analyse. La régression des surcoûts ne semble pas pertinente, d'autant que les faibles temps observés pour les petites images sont plus facilement faussés par des aléas de mesure. Nous analysons donc les moyennes. Les 6 mesures donnent un surcoût moyen du réseau Ethernet de 2,7 (dispersion raisonnable de 16 %). La moyenne est de 1,11 pour le réseau Myrinet (dispersion correcte de 13 %). On constate que les performances du réseau Myrinet donnent une proportion de communications par rapport aux calculs de 11 %, tandis que cette proportion est de 168 % sur réseau Ethernet.

En conclusion, le réseau Ethernet est désavantageux pour la parallélisation que nous avons envisagée, qui nécessite (actuellement) des communications à chaque `association`. Cette expérience a permi de vérifier l'intérêt du réseau Myrinet pour notre projet, et de quantifier la différence entre les deux réseaux.

(k) Il n'y a pas de *unpack* ANET à l'arrivée sur un processeur esclave car les données ne sont pas réorganisées en pvars et `rnets` sur ces processeurs. Il n'y a donc pas non plus de `pack` ANET au départ d'un processeur esclave.

(l) En effet, pour l'instant, ANET effectue des *packs* même lorsque les *threads* restent en local.

8.4.6 Influence de l'opérateur sur la parallélisation

But. L'envoi d'un *thread* à distance n'est pas rentable si le module maître attend davantage à cause des communications qu'il ne gagne de temps en confiant du travail au module esclave. Mais, à l'influence du réseau que nous avons évalué précédement, il faut ajouter celle du grain de la primitive data-parallèle. En effet, l'envoi d'un *thread* sur un processeur distant donne du travail au processeur du module maître à cause des *packs* ANET et PM^2. Aussi, indépendamment des performances du réseau et de l'attente des communications, l'exécution à distance d'un *thread* n'est rentable que si elle soulage plus le processeur maître qu'elle ne le surcharge. Il faut en effet que le calcul déporté (qui ne sera pas exécuté par le processeur du module maître), soit plus important que le travail nécessité par l'envoi (qui est effectué par le processus maître). Dans le cas contraire, le processus maître n'aura pas intérêt à déporter le calcul. La parallélisation des `associations` utilisant des opérateurs qui impliquent trop peu de calcul risque ainsi d'être pénalisante. Il est donc important de déterminer quels sont les opérateurs qui peuvent bénéficier de cette parallélisation.

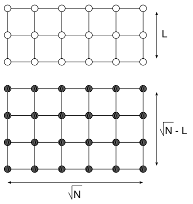


FIG. 127 – Partage de la grille en deux régions ; les nœuds grisés sont traités à distance.

Méthode. Nous avons utilisé un programme qui exécute un grand nombre d'`associations` avec différents opérateurs. Les opérateurs testés sont le s-opérateur logique or, le s-opérateur numérique min, et le r-opérateur minc.

Les `associations` ont été appliquées sur un `subnet` représentant une grille coupée en deux parties, contenant L lignes pour la première et $\sqrt{N} - L$ pour la seconde (N est le nombre de nœuds de la grille). Nous avons fait varier L de 1 à $\sqrt{N}/2 - 1$, et N a pris les valeurs 64×64, 128×128 et 256×256. `ANET_Q` était toujours inférieur à $L \times \sqrt{N}$ (le nombre de nœuds de la plus petite région). Il y a donc toujours eu deux *threads*. En positionnant `ANET_N` à $(\sqrt{N} - L) \times \sqrt{N} - 1$ (nombre de nœuds de la plus grosse composante moins un), le *thread* attribué à la plus grande des régions est toujours parti à distance.

La mesure de la variation de la proportion des données traitées en local par rapport à celles traitées à distance permet d'évaluer le gain de la parallélisation même lorsque la charge est déséquilibrée. Le processeur maître qui, dans cette expérience, est toujours le moins chargé, attend le processeur esclave, et les résultats vont être influencés par le réseau. Si le grain du calcul data-parallèle qu'implique l'opérateur est suffisamment important, le temps d'exécution avec $L \simeq \sqrt{N}/2$ devrait être inférieur avec deux processeurs plutôt qu'un.

Nous avons mené les tests sur réseau Ethernet et Myrinet. Pour comparer la parallélisation des `associations`, nous avons effectué les tests avec un, puis deux processeurs. Puisque nous ne disposions pas de machines équivalentes sous Ethernet, nous avons utilisé d'une part une machine esclave plus performante que la machine maître, et d'autre part une machine esclave moins performante.

Résultats. Les résultats sont donnés sous forme de courbes en annexe (temps d'exécution en fonction d'`ANET_N`). La figure 134 page 192 donne les temps de calcul pour l'opérateur or sur les réseaux Ethernet et Myrinet en fonction d'`ANET_N`. La figure 135 page 193 présente les courbes similaires pour l'opérateur min, et la figure 136 page 194 celles concernant l'opérateur minc.

Analyse. Les courbes des figures 134 et 135 indiquent que les `associations` utilisant les s-opérateurs logiques de type or, et numériques de type min, ne présentent aucun intérêt à être exécutées sur deux processeurs distants. Cela s'explique par le fait que ces opérateurs sont trop simples, et que la recopie des données en vue de l'exécution à distance est comparable, voire plus coûteuse, que l'exécution en local du calcul. À charge égale, les temps sont comparables sur Myrinet, mais fort différents sur Ethernet. Celà est dû au fait que le module maître attend le retour du calcul du module esclave, et que cette attente est d'autant plus longue que le réseau est lent.

Les résultats sont tout autres pour l'opérateur minc (figure 136 page 194). Même sur réseau Ethernet, et avec un processeur esclave moins bon, la parallélisation de la minc-`direct-association` présente un léger intérêt lorsque la charge n'est pas trop déséquilibrée. Cet avantage croît avec la taille des données. Il est beaucoup plus net, et quel que soit le déséquilibre de charge, lorsque le processeur esclave est meilleur que le processeur maître ; mais il faut noter que les deux processeurs ont une puissance de calcul bien différente.

Sur réseau Myrinet, l'intérêt de la parallélisation apparaît même lorsque la charge est fortement déséquilibrée. Puisque les processeurs utilisés sont ici équivalents, nous avons calculé les rapports des temps de calcul séquentiels et parallèles, pour les trois tailles de grille. Les résultats sont donnés par les courbes de la figure 128 ; elles ont été normalisées en fonction de la proportion de travail effectuée en local. La forme et la superposition des courbes peuvent, à première vue, surprendre. En fait, si on suppose que la communication est négligeable, les temps de calcul sont essentiellement dus à la `minc-direct-association`. On peut dans ce cas assimiler le temps de la version à deux processeurs au temps de calcul de la `direct-association` sur la grande région. Ce temps est donné par $[(\sqrt{N} - L) \times \sqrt{N}]^2$ (le calcul a une complexité de $O(N^2)$, où N représente le nombre de pixels de la région). Le temps séquentiel est donné par la somme des temps nécessaires pour les deux régions : $[(\sqrt{N} - L) \times \sqrt{N}]^2 + [L \times \sqrt{N}]^2$. Le rapport des temps séquentiel sur parallèle est donc égal à $1 + L^2/(\sqrt{N} - L)^2$. Or la proportion des données traitées en local est donnée par $x = L/(\sqrt{N} - L)$. On retrouve donc la forme des courbes de la figure 128.

La superposition des courbes indique que la quantité de données envoyée n'influence pas le coût du réseau. Le rapport des temps presque optimal à $x = 1/2$ montre que le coût du réseau est négligeable sur Myrinet pour la `minc-direct-association`.

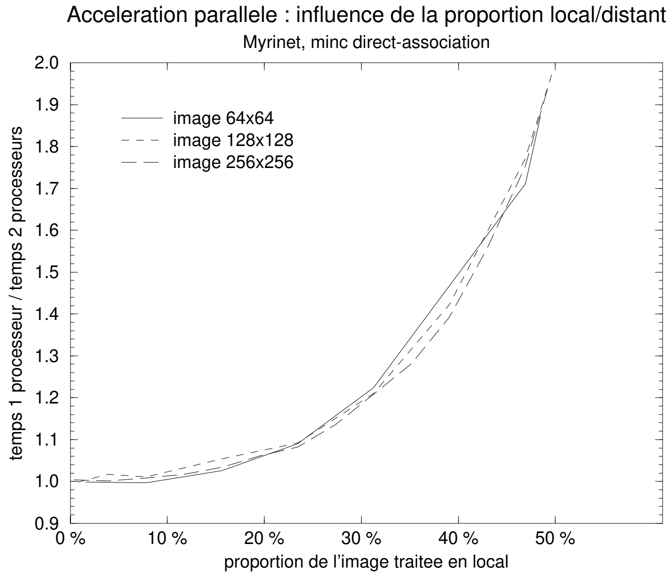


FIG. 128 – *Avantage de la parallélisation pour la* `direct-association` *et le* r-*opérateur* minc. *Réseau Myrinet, PM^2 High Perf, BIP.*

En conclusion, seules les `direct-associations` basées sur des r-opérateurs, dont le calcul est en $O(N^2)$, peuvent bénéficier de la parallélisation sur machine à mémoire distribuée que nous avons mise en place dans ANET. Il est intéressant de noter que c'est aussi le cas, dans une moindre mesure, sur réseau Ethernet, lorsque la charge est à peu près équilibrée. Le résultat est très net sur réseau Myrinet. Plus le grain du calcul data-parallèle est important, plus la parallélisation par attribution des composantes connexes aux *threads* est intéressante. Outre l'avantage signalé au § 8.4.4, les r-opérateurs offrent un grain de calcul qui autorise une parallélisation par création de *threads* à distance pour chaque primitive data-parallèle. Cela permet d'obtenir une bonne parallélisation, mais aussi d'adapter la répartition des données aux unités de calcul à chaque appel de primitive. Il est donc possible de s'adapter à des programmes très irréguliers, tant dans la forme de leurs données (régions évoluant sans cesse, comme dans la segmentation de Voronoï), que dans leurs traitements, en ajustant dynamiquement et régulièrement la charge des machines.

8.4.7 Paramétrages de la bibliothèque

Nous avons mené diverses expériences pour trouver les valeurs des paramètres ANET_Q et ANET_N qui apportent les meilleures performances [Ducourthial et Sicard, 1998]. Il ressort de ces expériences que le paramètre ANET_Q idéal est lié à la taille des données (évolution linéaire). Ce résultat, *a priori* étonnant, est lié au fait que le nombre de *threads* est borné par $N/$ANET_Q. À ANET_Q fixé, plus N est grand, plus il y aura de création de *threads*, qui se traduisent par autant de mallocs. Le paramètre ANET_N, qui devait donner une heuristique grossière d'équilibrage de charge, s'est révélé peu pertinent. Sauf dans des cas précis (*cf.* § 8.4.6), les *threads* restaient tous en local, ou, au contraire, partaient tous à distance. Ceci est dû à l'homogénéité moyenne des nombres de nœuds confiés à chaque *thread* (ce qui ne signifie pas que les composantes connexes sont, elles, homogènes en taille).

8.5 Conclusion

Résumé. Dans ce chapitre, nous avons décrit la bibliothèque ANET. La validation du bien fondé du modèle théorique pour la programmation data-parallèle nécessitait un tel développement. En effet, le chapitre précédent a validé l'intérêt de la direct-association et des r-opérateurs en algorithmique ; il restait donc à montrer que ces algorithmes pouvaient être mis en œuvre efficacement.

ANET est une bibliothèque de programmation parallèle, écrite en C++, au dessus de l'environnement à processus légers PM^2. Elle peut être enrichie par de nouvelles classes et de nouveaux opérateurs, au gré des besoins. Elle peut s'exécuter sur toutes les plate-formes acceptant PM^2. La bibliothèque permet de programmer avec la direct-association, et les r-opérateurs. Elle propose une machine virtuelle d'instanciation paramétrable, mais autorise tout type d'objets irréguliers sur cette topologie.

Bilan. Le développement d'ANET et les tests que nous avons menés permettent de tirer des conclusions sur quatre points.

Bien que figée, la machine virtuelle permet, lorsqu'on choisit une maille, de travailler avec toutes sortes de topologies, codées sous forme d'un sous-réseau. Il semble donc que les modèles data-parallèles à machines d'instanciations figées — plus simples à réaliser — peuvent manipuler des données irrégulières ; en outre, le codage des données par des sous-réseaux apparaît intéressant.

Les primitives data-parallèles du modèle sont parallélisées en associant des *threads* PM^2 aux composantes connexes du graphe réduit, obtenu à partir du sous-réseau codant l'irrégularité des objets. Ces *threads* peuvent s'exécuter sur des processeurs distants, mais ils nécessitent de passer du sous-réseau (subnet) au graphe réduit (rnet), qui est un tableau de CC, elles-mêmes tableaux de CFC, elles-mêmes tableaux de nœuds. La gestion des structures de données dans ANET (topologie quelconque et graphe réduit) conduisent à un surcoût très raisonnable par rapport à SAM. De même, le *pack* ANET n'est pas très coûteux. La parallélisation des primitives data-parallèles basée sur le graphe réduit, qui respecte l'irrégularité des données, semblent donc être une solution intéressante.

Les r-opérateurs permettent de remplacer, dans un programme tel que la découpe d'une image en régions de Voronoï, les itérations composées d'associations utilisant des s-opérateurs par une seule direct-association utilisant un r-opérateur. Bien que cette dernière soit plus longue, le gain obtenu est très sensible, puisqu'il avoisine un ordre de grandeur (environ 9 fois plus rapide sur un panel d'images assez large). De plus, le grain du calcul étant plus important, l'exécution à distance des primitives data-parallèles utilisant de tels opérateurs est avantageuse, tandis qu'elle ne l'est pas pour des grains plus faibles, obtenus avec des opérateurs classiques tels que min, or, plus, *etc.* Le grossissement du grain du calcul par l'introduction de primitives asynchrones dans le data-parallélisme semble donc être une idée intéressante.

Enfin, ANET a permis de valider le modèle de calcul théorique des réseaux associatifs, puisque celui-ci peut être mis en œuvre sous la forme d'une bibliothèque de programmation parallèle. ANET est un premier développement expérimental, utilisable en l'état, mais qui laisse aussi espérer de nombreuses améliorations. Elle propose une parallélisation différente de celles des autres études portant sur l'utilisation des *threads* en data-parallélisme [Perez et Namyst, 1997] [Bougé et Perez, 1998] [Verbièse et al., 1998]. Elle tire parti du grain de l'opérateur, et incite donc à programmer en utilisant les primitives asynchrones et les r-opérateurs.

Travaux futurs. Si la parallélisation offerte par ANET peut conduire à de bonnes performances sur réseau Myrinet, elles sont cependant faibles sur réseau Ethernet 10 Mb/s. Le déséquilibre de la charge nuit grandement aux performances. La solution temporaire que nous avions imaginée n'est pas satisfaisante. Elle ne nous a pas permis de tester l'avantage d'ANET sur un cas réel, plutôt que sur un programme-test. Les développements futurs devront porter sur un mécanisme d'équilibrage de la charge, qui placera les *threads* en local ou à distance sur des processeurs peu chargés, et lorsque le ratio calcul / communication est favorable.

ANET peut être optimisée sur différents points, tels que l'exécution de plusieurs `associations` à la fois, ou la construction préventive du `rnet`. Une autre piste prometteuse consiste à laisser les données à distance tant que le `subnet` n'est pas modifié, et à les rappatrier sur le processeur central uniquement en cas de nécessité (*redistribute*). Cela limitera grandement les communications, et devrait permettre de paralléliser plus de primitives.

Enfin, il est possible d'obtenir de meilleurs performances avec une solution délibérément distribuée, dans laquelle il n'y aurait plus de processeur maître. Il est difficile de se prononcer dés maintenant sur cette idée, car l'avantage lié à la diminution des envois de données pourrait être gaché par un trop lourd protocole de cohérence des nœuds frontières, dont les voisins sont détenus par des processeurs distants.

Des expérimentations sont donc encore nécessaires, que ce soit pour étudier l'intérêt de ces optimisations, ou pour évaluer le comportement de notre parallélisation lors d'un «passage à l'échelle» (augmentation du nombre de processeurs).

CONCLUSION

Dans la première partie de ce document, nous nous sommes intéressés à la définition du modèle de calcul des RÉSEAUX ASSOCIATIFS. *Dans la seconde partie, nous avons étudié les propriétés intrinsèques de ce modèle. Enfin, dans la troisième partie, nous avons abordé ses applications, sous l'angle de l'algorithmique, et de la mise en œuvre.*

Nous présentons maintenant le bilan de notre travail, et des idées avancées, avant d'envisager les travaux futurs.

Sujet de l'étude

Le modèle de calcul parallèle des Réseaux Associatifs a d'abord été proposé pour le traitement d'images sur machine spécialisée. L'originalité du modèle réside dans ses calculs globaux sur des graphes, qui définissent des zones irrégulières, et qui sont manipulés dynamiquement.

Notre travail a concerné l'étude des propriétés intrinsèques du modèle, et de ses aptitudes au calcul parallèle. Il s'agissait d'envisager les réseaux associatifs en tant que modèle de programmation général. L'introduction d'une nouvelle primitive, réalisant un calcul global asynchrone, a précisé les orientations de notre travail. Celles-ci ont porté sur l'étude de la terminaison, des applications algorithmiques, et de la mise en œuvre logicielle de la cette primitive.

Travail effectué

Segmentation de Voronoï. Après avoir rappelé la première définition du modèle au chapitre 1, nous avons éprouvé ses propriétés en algorithmique d'images, et plus particulièrement avec l'algorithme non trivial de la segmentation de Voronoï (chapitre 2).

La segmentation d'images présente des évolutions irrégulières des données, et des traitements, imprévisibles car dépendant entièrement de l'image traitée. La segmentation de Voronoï conduit à des objets aux contours très irréguliers, contrairement aux segmentations à motifs réguliers. Le codage, et le maniement, de ces objets irréguliers et fortement dynamiques n'est pas aisé en parallélisme de données.

Il ressort de cette étude que le modèle des réseaux associatifs permet sans difficulté de coder, et de manipuler en parallèle, les objets irréguliers, grâce aux *pvars de masques*. Il permet un passage souple du codage de l'image par les pixels au codage par les régions, grâce aux *graphes virtuels*. La primitive `association`, qui s'applique aux zones connexes de l'image, se révèle bien adaptée à ce type de problème.

Nouvelle définition. Dans le troisième chapitre, nous avons mis en avant des différences significatives entre le modèle tel que défini en 1992 et la Maille Associative. Elles sont liées à la difficulté de mettre en œuvre une primitive de haut niveau (`association`) travaillant sur des ensembles dans un modèle d'exécution peu contraint. Nous avons montré que le modèle d'exécution de la Maille et les réseaux associatifs coïncident sur les *sous-réseaux préfixes* pour tout type d'opérateurs, et sur tous les réseaux pour les s-*opérateurs* (associatifs, commutatifs et idempotents). L'idempotence est apparue comme une notion centrale pour les calculs sur réseaux.

Partant de ces constatations, nous avons introduit une nouvelle primitive de calcul, la `direct-association`. Celle-ci exécute un calcul global asynchrone qui est défini par la donnée d'un calcul local pour chaque nœud, répété jusqu'à complète stabilisation du réseau. Cette primitive n'est définie que si elle se termine. La différence est essentielle avec l'`association`. En effet, les calculs ne sont plus spécifiés par une indication sémantique directe (*e.g.*, «calculer le maximum d'une zone»). Ils sont maintenant caractérisés par le calcul local effectué par chaque nœud. La signification de l'instruction data-parallèle vient après, si le calcul termine, s'il est déterministe, et si l'on sait reconnaître son résultat. En contrepartie, la définition du modèle est peu contraignante. Il n'est pas supposé de coordinations entre les calculs locaux et les communications. Cette définition est suffisamment générale pour permettre des mises en œuvres variées.

Étude matricielle. La définition du modèle étant précisée, il nous fallait étudier les conditions rendant utilisable la `direct-association`.

Étant donné l'importance de l'idempotence dans notre travail, nous nous sommes intéressés à l'algèbre max-plus. Celle-ci permet une modélisation matricielle du calcul de la `direct-association` : les valeurs des nœuds sont notées dans un vecteur ; leurs évolutions sont données par une multiplication matricielle, utilisant la matrice de précédence correspondant au réseau. L'étude de la terminaison de la `direct-association` devient un point fixe matriciel. Mais le calcul matriciel implique des phases de calculs globales, correspondant aux mises à jour successives du vecteur. Les nœuds se trouvent donc synchronisés.

Nous avons donc cherché à désynchroniser le calcul matriciel, grâce aux itérations asynchrones. Ces techniques ont été développées dans le cadre du calcul matriciel parallèle, pour limiter les synchronisations. Sous certaines conditions, elles autorisent la mise à jour des composantes du vecteur indépendamment les unes des autres. Nous avons montré que le calcul de la matrice $\mathbf{A}^*$, capitale dans la résolution des équations matricielles max-plus (points fixes, algèbre de chemins), peut être calculée avec une itération asynchrone. De même, le calcul de la `direct-association` peut être modélisé, et étudié, par une itération asynchrone : la convergence vers la stabilité du réseau asynchrone devient recherche d'un point fixe matriciel.

La modélisation matricielle, même asynchrone, des propriétés calculatoires du réseau ne peut cependant se suffire à elle-même. En effet, elle souffre de cet aspect macroscopique, qui incite à penser les algorithmes sous la forme d'un calcul matriciel global. Ce produit matriciel ne peut qu'être conçu de façon synchrone, et le programmeur ne peut qu'espérer une traduction dans un modèle d'exécution asynchrone plus performant. Au contraire, nous pensons que le programmeur a intérêt à prendre en compte l'asynchronisme le plus tôt possible, en amont de la programmation, dès la conception des algorithmes. L'asynchronisme n'est plus un moyen de mise en œuvre efficace, ni un artefact subi de la programmation parallèle, mais devient un outil algorithmique à part entière.

Algèbre déformée. Ces considérations nous ont conduit à étudier la terminaison de la `direct-association` sous l'angle microscopique de l'opérateur utilisé par chaque nœud du réseau. Nous avons utilisé pour cela une méthode de réduction des expressions. Les s-opérateurs réduisent les expressions, grâce à l'associativité, la commutativité et l'idempotence. Mais la réduction des expressions, et donc la terminaison du calcul global, peut être obtenue avec des hypothèses plus faibles que celles des s-opérateurs.

Nous avons ainsi défini, et étudié, les r-algèbres, obtenues en déformant l'algèbre idempotente par une r-fonction. Ainsi, au s-opérateur associatif, commutatif et idempotent, correspond le r-*opérateur*, r-*associatif*, r-*commutatif* et r-*idempotent*.

Cette algèbre déformée admet d'intéressantes propriétés algébriques. En particulier, les r-opérateurs idempotents réduisent les expressions, et conduisent à la terminaison déterministe de la `direct-association`. Nous avons étendu ces résultats aux r-*opérateurs* n-*aires*, et *généralisés*.

Nous avons aussi précisé les liens existant entre un r-opérateur et une matrice d'algèbre de chemins généralisée.

Les r-opérateurs étendent très largement le résultat de terminaison de la `direct-association` que nous avions établis pour les s-opérateurs. Les propriétés calculatoires du réseau étant mieux cernées, nous nous sommes intéressés aux propriétés de contrôle (au sens large).

Résistance aux pannes. Nous avons expliqué au chapitre 6 que dans le cas d'une mise en œuvre par passage de messages, les propriétés du modèle, et des s- et r-opérateurs, autorisaient des hypothèses très faibles sur les liens de communications. Nous nous sommes aussi intéressés à la complexité de la `direct-association`. Celle-ci dépend de l'opérateur utilisé. Mais pour les s- et les r-opérateurs binaires, elle est proportionnelle au diamètre du réseau. Nous avons également étudié la détection de la terminaison de la `direct-association`. Nous préconisons, pour le calcul parallèle, une détection centralisée. Cependant, nous avons proposé une modification de cette primitive pour en faire un protocole distribué (*vague*) qui inclut sa propre détection de terminaison.

Enfin, nous avons envisagé la résistance aux pannes transitoires de la `direct-association`. Nous avons donné un algorithme à base de règles pour cette primitive ; il respecte les spécificités de la *lecture-écriture atomique*. Nous avons montré que lorsque les r-opérateurs sont basés sur un s-opérateur définissant un ordre total, alors cet algorithme est auto-stabilisant si et seulement si le r-opérateur est *strictement idempotent* (*i.e.*, la r-fonction est strictement croissante). Nous avons aussi donné une preuve matricielle pour cela, grâce à l'algèbre max-plus et aux itérations asynchrones. Cette preuve n'utilise pas l'hypothèse de l'ordre total, mais elle n'est valable qu'en atomicité composite.

Outre les aspects de la résistance aux pannes, l'étude de l'auto-stabilisation de la `direct-association` aura aussi permis de mieux caractériser les r-opérateurs (*cf.* figure 73 page 119).

Algorithmique. Le modèle étant mieux établi dans sa définition, et les fondements théoriques de la `direct-association` étant mieux connus, nous nous sommes intéressés à ses applications algorithmiques.

Contrairement à l'`association`, qui est toujours définie, l'algorithmique de la `direct-association` nécessite l'emploi d'un opérateur idempotent, ou d'un sous-réseau préfixe. Par contre, elle permet d'utiliser les r-opérateurs. Ces derniers fournissent un résultat qui inclut une information relative à la distance ou aux chemins du réseau, tandis que les s-opérateurs n'exploitent que l'information relative à la forte connexité.

Nous avons illustré l'intérêt de la `direct-association` en algorithmique de graphes (détection de DAG, de composantes connexes, construction d'arborescences des plus courts chemins, de poids minimum, *etc.*). Les r-opérateurs permettent d'adapter, grâce aux relations que nous avons soulignées, des algorithmes de l'algèbre de chemins à la programmation sur réseau. À l'inverse, de nouveaux algorithmes définis pour les réseaux associatifs peuvent s'appliquer au calcul matriciel de l'algèbre de chemin. Notons que plusieurs de ces algorithmes bénéficient de la preuve de l'auto-stabilisation, que nous avons établie pour toute une famille de r-opérateurs.

Nous avons ensuite étudié les nombreuses applications de la `direct-association`, et des r-opérateurs à l'algorithmique géométrique, et au traitement d'images. En particulier, une seule `direct-association` permet de calculer une transformée en distance, ou d'obtenir les régions de Voronoï. Ici, l'asynchronisme agit comme un procédé de relaxation : chaque nœud adapte localement son résultat à la meilleure solution qu'il rencontre. L'ensemble de ces modifications locales non coordonnées se stabilise lorsqu'une solution globale optimale est trouvée. Le programmeur a tout intérêt à compter sur cette relaxation pour résoudre ses problèmes algorithmiques. Ainsi, les itérations imbriquées utilisant des s-opérateurs de la première version de la segmentation de Voronoï (chapitre 2) sont avantageusement remplacées par une seule instruction data-parallèle asynchrone.

Mise en œuvre. Après l'étude du modèle, et de son algorithmique, il nous restait sa mise en œuvre. Nous avons présenté au chapitre 8 une bibliothèque de programmation parallèle : ANET.

Cette bibliothèque reprend les concepts des réseaux associatifs. En particulier, elle gère les objets irréguliers dans une machine virtuelle d'instanciation fixe, mais paramétrable. Les primitives de calcul sont parallélisées en associant des *threads* PM^2 aux composantes connexes. Cette technique souple permet de respecter l'irrégularité et la dynamicité des objets. Elle autorise aussi l'équilibrage dynamique de la charge des processeurs. La bibliothèque ANET est programmée en C++. Le programmeur peut se définir ses propres objets, et ses propres opérateurs pour les manipuler.

Nous avons procédé à des tests sur réseaux classique (Ethernet 10 Mb/s) et rapide (Myrinet, 1 Gb/s) pour valider notre parallélisation. Deux résultats principaux ressortent de ces expériences. Le premier, fort prévisible, concerne l'équilibrage de la charge. Le prototype ANET propose actuellement une répartition de charge beaucoup trop simple pour qu'un algorithme aussi irrégulier que la segmentation de Voronoï puisse être parallélisé efficacement. Mais la parallélisation par *threads* PM^2 associés aux composantes connexes ouvre la voie à un contrôle de la charge précis, soit au moment de la création des *threads*, soit ultérieurement, par migration.

La seconde observation était, elle aussi, prévisible, mais nous l'avons quantifiée. Elle concerne le rapport entre grain du calcul data-parallèle et efficacité de la parallélisation, sur machine à mémoire distribuée. Les r-opérateurs conduisent à un grain de calcul plus gros, ce qui est appréciable dans un modèle data-parallèle, où la finesse du grain est souvent un handicap. En quelque sorte, les *threads* sont créés et envoyés à distance pour une charge de travail suffisante, rentabilisant création et communication. Si la parallélisation des instructions data-parallèles utilisant des r-opérateurs est intéressante sur réseau Ethernet, elle est réellement significative sur réseau Myrinet.

Bilan

Nous faisons maintenant le bilan des idées que nous avons avancées dans notre travail.

Irrégularité. La gestion de l'irrégularité est encore un sujet de recherche en parallélisme de données. Notre contribution a porté à la fois sur le codage des données irrégulières, et sur la prise en compte des traitements irréguliers.

Les sous-réseaux d'un réseau de base pourtant fixé permettent de coder toute sorte d'objets. Ils permettent de s'adapter dynamiquement, et aussi rapidement qu'une affectation de variable parallèle, aux objets très irréguliers de l'image. Le modèle des réseaux associatifs permet ainsi de programmer l'algorithme de la segmentation de Voronoï, dont les régions sont irrégulières, et sans cesse modifiées. De plus, une topologie en maille permet de plonger tout type de graphes, sous des hypothèses en accord avec les primitives de calcul du modèle. Cela permet, par exemple, de travailler sur des hypercubes tout en gardant une machine virtuelle simple.

Dans la bibliothèque ANET, cette machine virtuelle est placée sur les processeurs d'une machine réelle en utilisant les *threads* PM^2. Ces *threads* sont associés aux composantes connexes du sous-réseau, ce qui permet de respecter l'irrégularité des objets, mais aussi les spécificités des primitives du modèle, qui travaillent en parallèle, et indépendamment, sur les composantes connexes.

L'autre avantage de cette technique réside dans sa souplesse, qui permet de s'adapter dynamiquement à l'évolution imprévisible des traitements irréguliers. L'exécution à distance d'un *thread*, ou sa migration, nécessitent des délais raisonnables. Le placement des *threads* exécutant les `associations` peut être effectué en fonction d'un rapide audit des charges des processeurs. Divers politiques de régulation de charges, étudiées ailleurs, pourront ainsi être adaptées dans ANET.

`direct-association`. Comme nous l'avons rappelé, la `direct-association` spécifie un traitement data-parallèle par la donnée d'un calcul local, et non par une indication sémantique directe. Cette particularité est loin d'être un handicap. En effet, la plupart des opérations data-parallèles classiques peuvent être spécifiées indifféremment au niveau local ou au niveau global. Par exemple, pour calculer le minimum des valeurs d'une pvar p, on peut directement spécifier $\beta\mathsf{min}\ p$. Mais on peut aussi indiquer à chaque nœud de calculer un minimum jusqu'à stabilité. Les deux formulations sont équivalentes dans un réseau fortement connexe, avec les s-opérateurs. De même, les *scans*, exécutés sur des lignes (ou tout type de graphe acyclique) peuvent indifféremment être spécifiés par un calcul local ou global.

L'avantage de la spécification par le calcul local permet cependant d'utiliser le réseau sous-jacent. Si son intérêt n'apparaît pas de façon claire avec un s-opérateur, il n'en est pas de même avec un r-opérateur. Un tel opérateur, de par sa dissymétrie, traite différemment la donnée locale et celles reçues des voisins. Cette particularité lui permet d'exploiter la nature du réseau. Ainsi, les r-opérateurs binaires construisent un résultat qui intègre la distance, plus riche que la forte connexité, et les r-opérateurs n-aires utilisent les chemins, plus riches encore que la distance.

Enfin, la `direct-association` permet d'utiliser des mécanismes de calcul asynchrones au sein d'une programmation à parallélisme de données.

Asynchronisme. L'intérêt de l'asynchronisme en parallélisme de données est de deux ordres. Le premier concerne l'écriture des algorithmes, et le second leur exécution.

La conception asynchrone d'un algorithme consiste à exploiter le calcul global asynchrone de la `direct-association` dès l'écriture d'un algorithme. L'asynchronisme n'apparaît plus uniquement dans la mise en œuvre, dans laquelle on aurait supprimé des synchronisations, mais dès la conception des algorithmes.

Comme nous l'avons expliqué, l'asynchronisme peut agir comme une méthode de relaxation, qui permet d'obtenir un résultat sans le spécifier totalement. Par exemple, pour obtenir les régions de Voronoï, une première solution consiste à programmer les itérations nécessaires à la recherche de la meilleure graine pour chaque pixel. C'est ce que nous avons présenté au chapitre 2. La seconde solution consiste à programmer un calcul global asynchrone, basé sur le calcul local utilisant le r-opérateur minc, qui se chargera de la recherche d'une solution.

Si l'exemple de Voronoï est le plus frappant, c'est la même technique qui est utilisée pour les forêts des chemins les plus fiables, ou bien encore pour l'arborescence en profondeur. Par contre, l'asynchronisme passe inaperçu avec un s-opérateur (exemple du minimum ci-dessus).

Le second avantage de l'asynchronisme est une conséquence directe du premier. Le remplacement des itérations, composées de plusieurs instructions data-parallèles, par une seule instruction plus conséquente grossit le grain du calcul. L'intérêt est non négligeable dans la suppression des contrôles du programme (boucles, mais aussi lancement de *threads* dans notre cas). Il est également indéniable dans la parallélisation sur environnement à mémoire distribuée, car l'exécution à distance d'une instruction data-parallèle est d'autant plus rentable que son grain est gros.

Connexions. Notre travail a aussi souligné des points communs entre disciplines aussi diverses que l'algèbre max-plus, les itérations asynchrones ou les systèmes distribués.

Ainsi, nous avons étudié sous l'angle macroscopique du calcul matriciel et sous l'angle microscopique des systèmes distribués les propriétés de la `direct-association`. La modélisation matricielle d'un phénomène asynchrone n'est possible qu'avec l'apport des itérations asynchrones. Bien que réducteur sur certains aspects (que nous avons rappelé précédemment), cette modélisation conduit à des preuves concises.

De même, il est frappant de constater certaines similitudes entre des concepts définis dans des domaines *a priori* très différents. Ainsi, la notion d'opérateur contractant des itérations asynchrones est assez voisine de l'absence de circuit absorbant en algèbre de chemins, et de l'idempotence des r-opérateurs. Notre travail a également mis en avant la relation moins évidente entre la propriété algébrique d'idempotence et le calcul asynchrone sur réseau.

Le formalisme des r-opérateurs a facilité les rapprochements — fructueux — entre ces domaines. Il semble que d'autres avantages puissent encore en être tirés.

Conclusion et perspectives

Nous signalions en introduction deux axes de recherche privilégiés dans notre travail : prise en compte de l'irrégularité et de l'asynchronisme dans le data-parallélisme. Notre étude a montré que le modèle des réseaux associatifs s'accommode de l'irrégularité des données et des traitements. Nous avons aussi montré que l'utilisation de l'asynchronisme en parallélisme de données était possible, et très profitable, que ce soit dans l'écriture des algorithmes ou dans l'exécution des programmes. La primitive `direct-association` que nous avons proposée permet une algorithmique variée ; elle peut aussi être mise en œuvre efficacement dans une bibliothèque de programmation parallèle à processus légers.

Plusieurs suites peuvent, d'ores et déjà, être envisagées.

Tout d'abord, certains points n'ont pas été complètement résolus. Citons la reconnaissance des résultats donnés par les r-opérateurs généralisés ou la recherche des conditions nécessaires à la terminaison de la `direct-association`, que nous avons seulement cernées.

En ce qui concerne l'auto-stabilisation de la `direct-association`, la relation entre l'ordre total et l'hypothèse d'atomicité des règles mérite d'être étudiée. En particulier, si nous avons utilisé l'ordre total pour la lecture-écriture atomique, nous n'avons cependant pas montré que cela était nécessaire. De même, il semble intéressant d'approfondir les liens entre l'algèbre max-plus et les systèmes distribués. Sous certaines hypothèses, certains problèmes de systèmes distribués pourraient bénéficier du formalisme unificateur de l'algèbre max-plus, qui trouverait là un domaine d'application supplémentaire.

L'algorithmique du modèle a été significativement étoffée. Mais de nombreuses voies restent à explorer. L'algorithmique géométrique et le traitement d'images fournissent de nombreux sujets d'études, qui devraient tirer parti de la `direct-association`. D'autres domaines pourraient aussi être abordés, tels que les réseaux de neurones par exemple. De même, à l'instar de la programmation dynamique, ou de la «relaxation» offerte par l'asynchronisme, il nous semble que notre modèle pourrait aussi trouver des applications en optimisation combinatoire.

Du côté des expérimentations, un certain nombre d'algorithmes étudiés restent à tester, de façon à évaluer les complexités moyennes, ou à calibrer les heuristiques (par exemple pour les segmentations d'images). La bibliothèque ANET est un outil de programmation très intéressant, qui devrait permettre de fructueuses études algorithmiques pratiques.

Enfin, de nombreuses améliorations peuvent être apportées à ANET. Citons notamment un mécanisme d'équilibrage de charge dynamique, prenant en compte l'état des processeurs et du réseau. L'étude d'un tel mécanisme devrait préciser s'il ne faut pas envisager un contrôle des calculs plus réparti. Par exemple, la redistribution des données sur les processeurs est actuellement effectuée à chaque `association`. Si cela est adapté aux algorithmes très irréguliers de segmentations, pour lesquels la forme des objets évoluent à chaque étape, ce n'est certainement pas la solution la plus efficace pour les programmes présentant une irrégularité moyenne. La mise en place d'une redistribution des données activée uniquement en cas de modification des formes des objets semble souhaitable. D'autres améliorations concernant l'exécution d'`associations` simultanément, ou la construction préventive de la structure gérant l'irrégularité (graphe réduit), pourraient aussi être apportées.

L'asynchronisme est généralement subi en programmation parallèle, et ce, dés que les unités de calculs ne bénéficient pas d'une synchronisation. Mais il est parfois recherché pour améliorer les performances (électronique, itérations asynchrones). Il semble que l'on puisse attendre beaucoup plus de l'absence de synchronisations entre les éléments calculants, en terme de performance intrinsèque, d'efficacité de parallélisation dans les bibliothèques de calcul, mais aussi d'écriture d'algorithmes (*programmation asynchrone*). Cette voie est incomplètement explorée. Son étude, basée sur la confrontation des résultats et des méthodes de différents domaines (algèbre, systèmes distribués, itérations asynchrones notamment), semble prometteuse.

Quatrième partie

Annexes, *etc*.

Annexe A

Segmentation de Voronoï

A.1 Programme de la découpe de Voronoï, à base de s-opérateurs

```
//A partir de IRacine, on fabrique mgRegion, le graphe des regions de voronoi.
void Voronoi(Image_int& IRacine, Subnet& mgRegion)
{
  // INITIALISATIONS
  Subnet mgT;                   // subnet de la la forêt couvrante, enraciné en IRacine.
  Subnet mgMesh = 0xff ;
  AssocMesh::MaskTorus(mgMesh); // mgMesh est le subnet de la maille non torique.
  Subnet mgTemp;                // subnet pour la mise a jour de mgFront.
  Image_int IRegion_new = IRegion;  // contient les changements de region des pixels.

  Subnet mgFront;               // subnet frontière.
  Image_int IDistRacine = -1;   // distance a la racine.
  Image_int IRVoisine = -1;     // numéro de la région voisine traitée.
  Image_int IDistRVoisine = -1; // distance a la racine de la région voisine choisie.
  Image_int IFin;               // pour la détection de la stabilité.
  Image_uchar IPossedeVoisin;   // vrai si la région a des voisins non examinés.

  mgRegion = 0;                 // au départ, pas de région.

  // FABRICATION DES PREMIERES REGIONS APPROXIMATIVES
  Image_int IRacineBis = 0;     // pour ne pas perdre de racine
  WHERE(IRacine != 0 ){         // lors de la converison en caractere.
    IRacineBis |= 1;
  }ENDWHERE;
  SpanningTree(mgT, mgMesh, IRacineBis, 1); // mgT est un arbre couvrant.
  Subnet mgTsym = mgT;
  mgTsym.Symmetrize();           //mgTsym est la forêt mgT symetrisée.
  Image_int IRegion = -1;
  WHERE(IRacine != 0)
    IRegion = IRacine;
  ENDWHERE;
  IRegion.MaxAssoc(mgTsym, IRegion); // numéro de la région du pixel.

  do{
    //FABRICATION DU GRAPHE DES REGIONS
    mgRegion.LinkWithEquals(IRegion, IRegion);
    AssocMesh::MaskTorus(mgRegion);

    //FABRICATION DU GRAPHE FRONTIERE
    mgFront = ~mgRegion;
    AssocMesh::MaskTorus(mgFront);

    //DISTANCE A LA RACINE DE LA REGION
    Image_int p, s;
    p.SelfX(); s.SelfY();
    p -= ((IRegion - 1) % IRacine.XSize());
    s -= ((IRegion - 1) / IRacine.YSize());
    p.Map(abs); s.Map(abs);
    IDistRacine = p+s;

    Image_int Itemp = 0;

    do{
      IPossedeVoisin.OrAssoc(mgRegion, mgFront); // recherche des régions voisines
```

```
      WHERE(IPossedeVoisin != 0){
        //CHOIX D'UNE SEULE REGION VOISINE PAR REGION
        IRVoisine = -1;
        //IRVoisine = num des regions voisines sur les bords :
        IRVoisine.MaxAssocStepEx(mgFront, IRegion);
        //IRVoisine = num region voisine choisie (une seule par region) :
        IRVoisine.MaxAssoc(mgRegion, IRVoisine );

        //DISTANCE A LA RACINE DE LA REGION CHOISIE
        Image_int p, s;
        p.SelfX(); s.SelfY();
        p -= ((IRVoisine - 1) % IRacine.XSize());
        s -= ((IRVoisine - 1) / IRacine.YSize());
        p.Map(abs); s.Map(abs);
        IDistRVoisine = p+s;

        //CHOIX DE LA RACINE LA PLUS PROCHE
        //(des pixels changent de region)
        WHERE((IDistRacine > IDistRVoisine)  && (IRacine == 0)){
          IRegion_new = IRVoisine;
        }ENDWHERE;

        // MISE A JOUR Du GRAPHE DES FRONTIERES.
        mgTemp.LinkWithEquals(IRVoisine, IRegion);
        AssocMesh::MaskTorus(mgTemp);
        mgFront &= ~mgTemp;
        mgFront.SetModified();
      }
      ENDWHERE;
    }
    while( mgFront.GlobalOr() != 0 );

    WHERE( IRegion != IRegion_new ) // Contrôle de la stabilité.
      IFin = 1;
    ELSEWHERE
      IFin = 0;
    ENDWHERE;

    //MISE A JOUR DES REGIONS
    IRegion = IRegion_new;

  }while( IFin.GlobalOr() != 0);     // iteration jusqu'à stabilité
}
```

A.2 Programme à base de r-opérateurs

```
void Voronoi2(Image_int& IRacine, Subnet& mgRegion)//passage par reference
{
  Subnet mgMesh(0xff);
  AssocMesh::MaskTorus(mgMesh); // subnet de la maille non torique.

  // CALCUL DE LA DISTANCE AVEC LE MINC
  Image_int ITmp;               // pour le calcul du Minc.
  WHERE(IRacine != 0)
    {
      ITmp = 0;                  // 0 sur les racines
    }
  ELSEWHERE
    {
      ITmp = ITmp.Size();       // infini ailleurs
    }
  ENDWHERE;
  ITmp.MincDirectAssoc(mgMesh, ITmp); // ITmp = distance à la racine la plus proche.

  // FABRICATION D'UN DAG DES PLUS PETITS ANTECEDENTS (DAG COUVRANT)
  Image_int ITmp2;
  ITmp2.MinAssocStep(mgMesh, ITmp); // Recherche des antécédents directs plus petits.
  Subnet mgTmp;
  mgTmp.LinkWithEquals(ITmp2, ITmp, mgMesh);
  WHERE( ITmp == 0 ){ // pas d'antécédents pour les racines
    mgTmp = 0;
  } ENDWHERE;

  // FABRICATION D'UN ARBRE PLANAIRE A PARTIR DU DAG (ARBRE COUVRANT)
  for (unsigned i = 0; i < AssocMesh::degree; i+=2){
 // Choix en priorité des  voisins horizontaux ou verticaux
    WHERE( (mgTmp & Mask(1 << i)) != 0 )
      {
        mgTmp = Mask(1 << i);
      }
    ENDWHERE;
  }
 // en second choix, les voisins atteints par une diagonale.
  for (unsigned i = 1; i < AssocMesh::degree; i+=2){
    WHERE( (mgTmp & Mask(1 << i)) != 0 )
      {
        mgTmp = Mask(1 << i);
      }
    ENDWHERE;
  }

  // FABRICATION DES REGIONS
  // ELles ne se chevauchent pas grâce à l'arbre couvrant planaire.
  ITmp2.MaxAssoc(mgTmp,IRacine);  // diffusion de l'adresse de la racine.
  mgRegion.LinkWithEquals(ITmp2, ITmp2, mgMesh); // connexion complete des régions.
}
```

A.3 Exemples de résultats

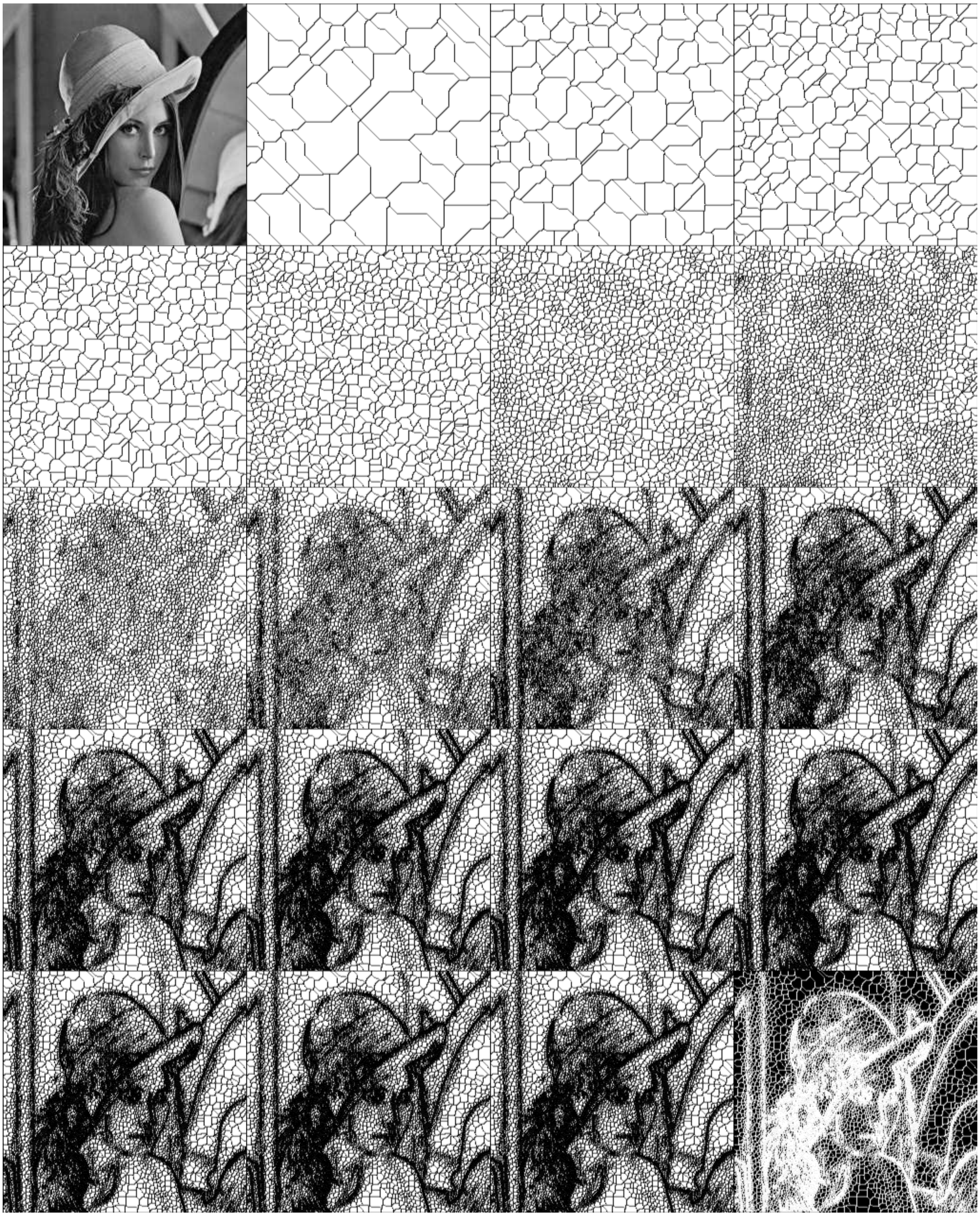


FIG. 129 – *Découpe de Voronoï sur l'image* lena.

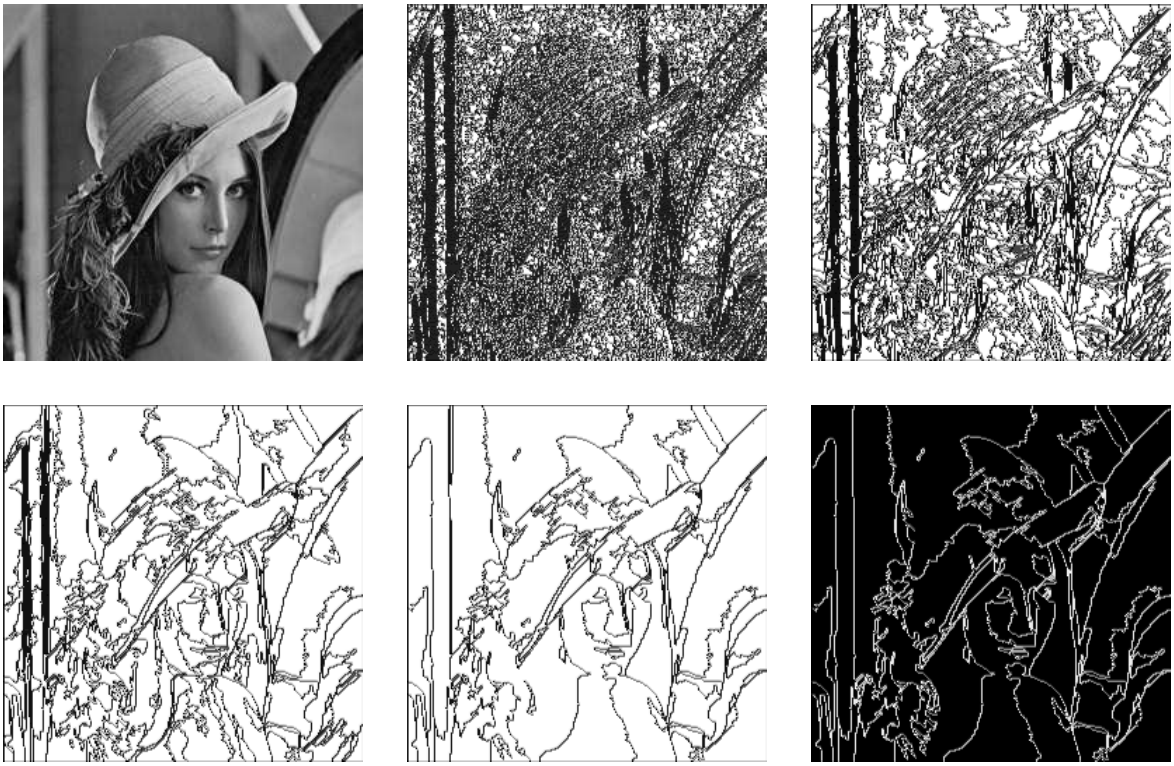


FIG. 130 – *Fusion de Voronoï sur l'image* lena.

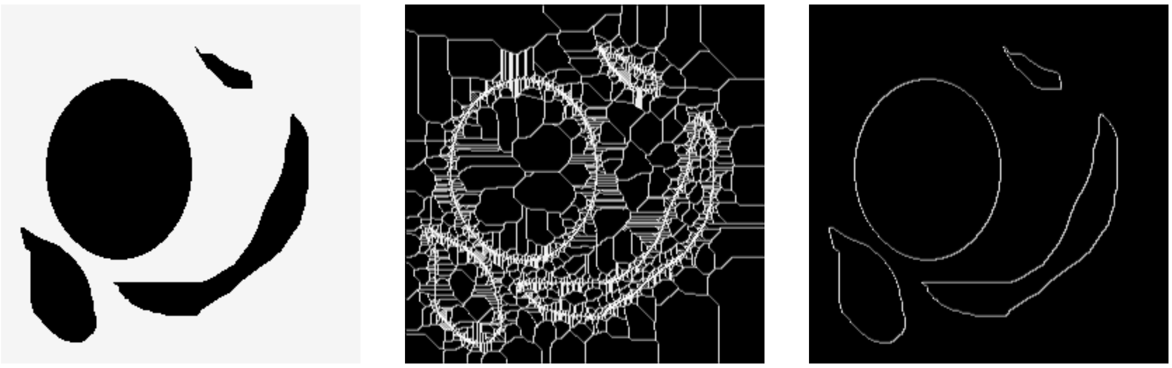


FIG. 131 – *Algorithme de découpe et fusion de Voronoï appliqué à l'image* forme.

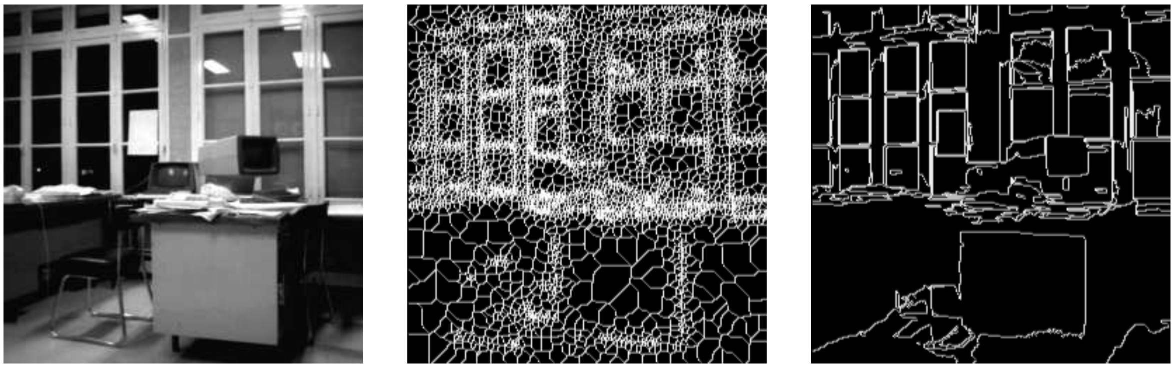


FIG. 132 – *Algorithme de découpe et fusion de Voronoï appliqué à l'image* bureau.

Annexe B

Données expérimentales

B.1 Images tests

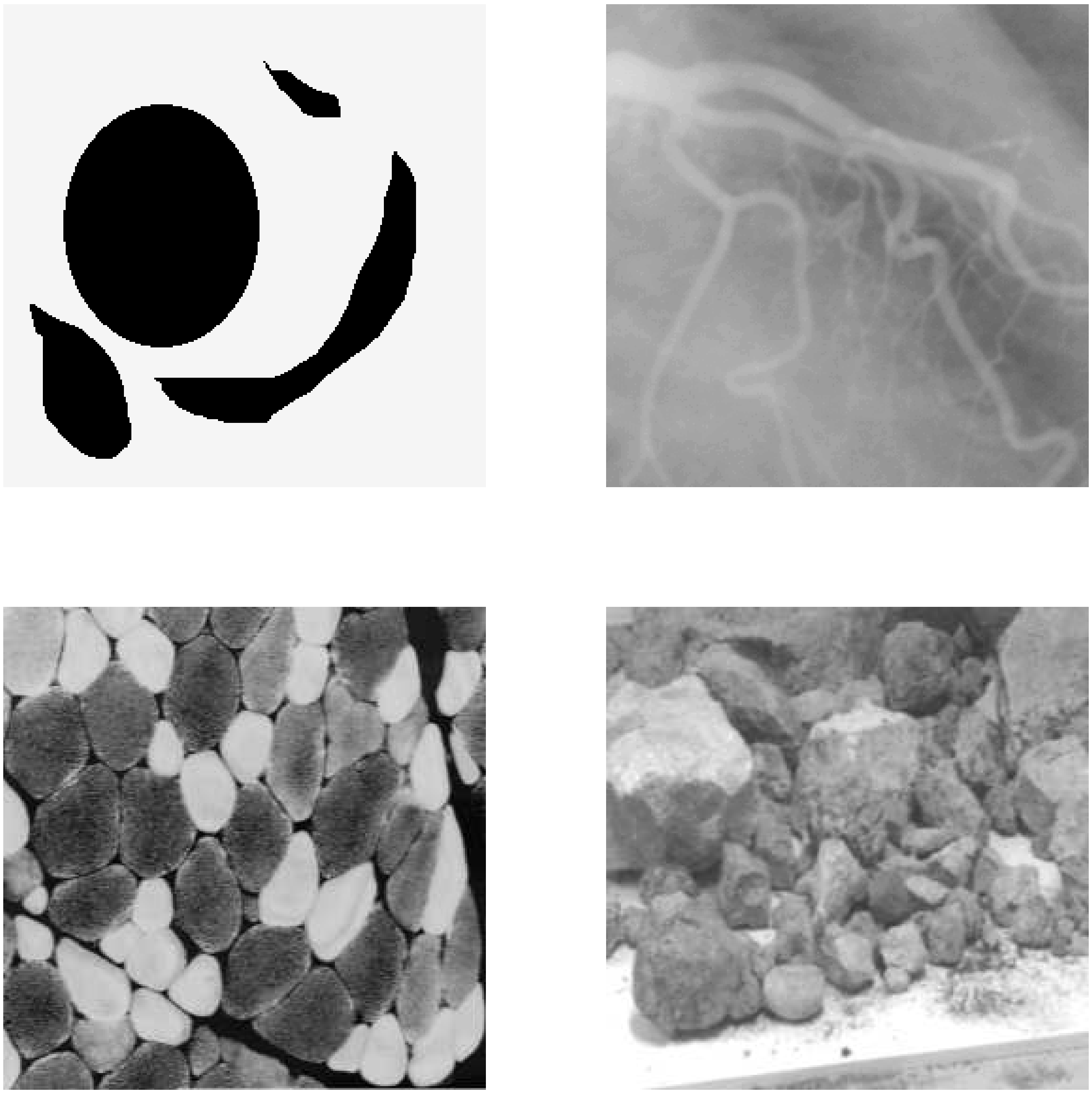


FIG. 133 – *Images utilisées pour les tests de la bibliothèque* ANET*, avec le programme de découpe et fusion de Voronoï. De gauche à droite et de bas en haut :* forme*,* icoron*,* imuscle *et* roks*.*

B.2 Influence du rapport calcul / communication

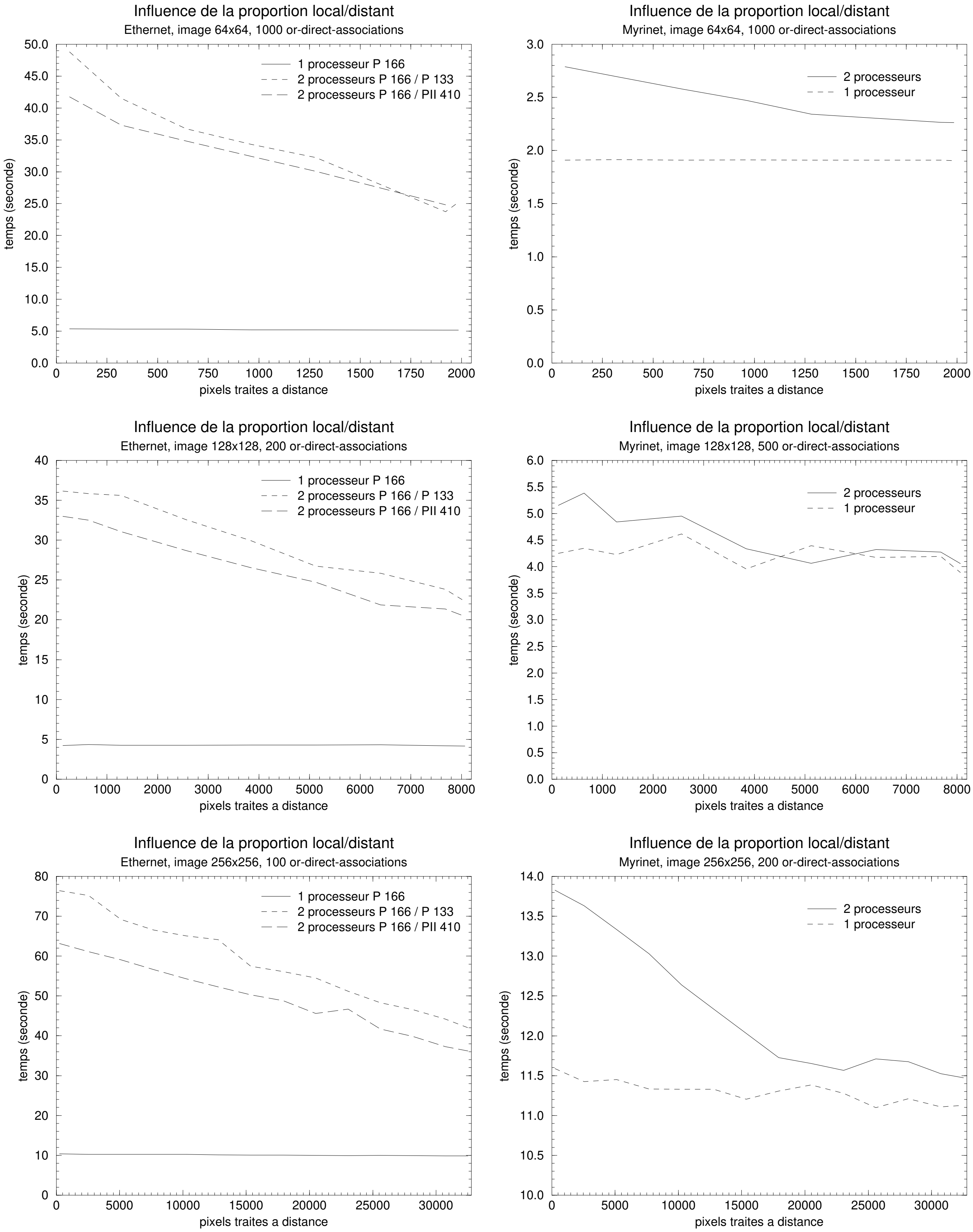


FIG. 134 – *Rapport calcul / communication. Opérateur* or.

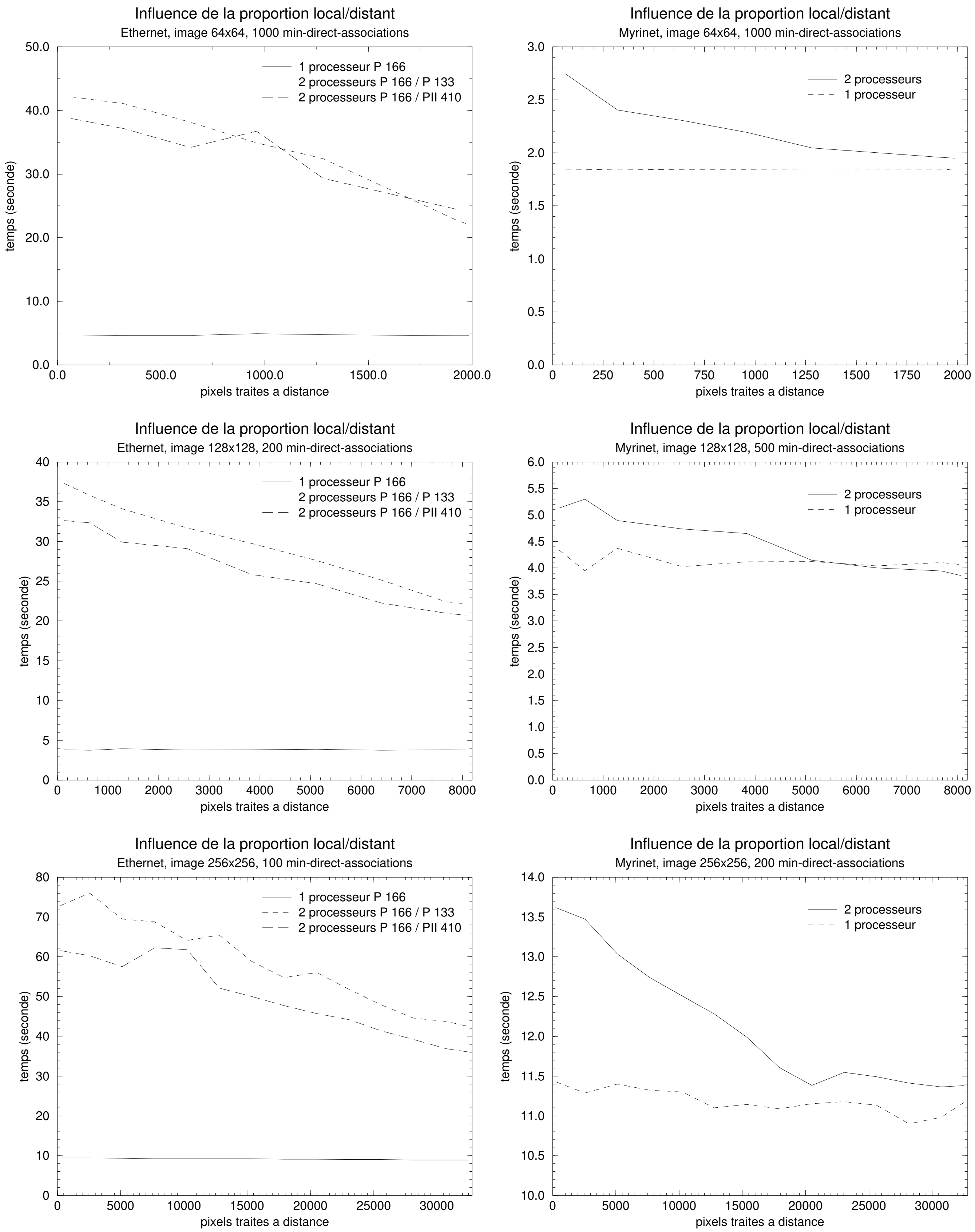


FIG. 135 – *Rapport calcul / communication. Opérateur* min.

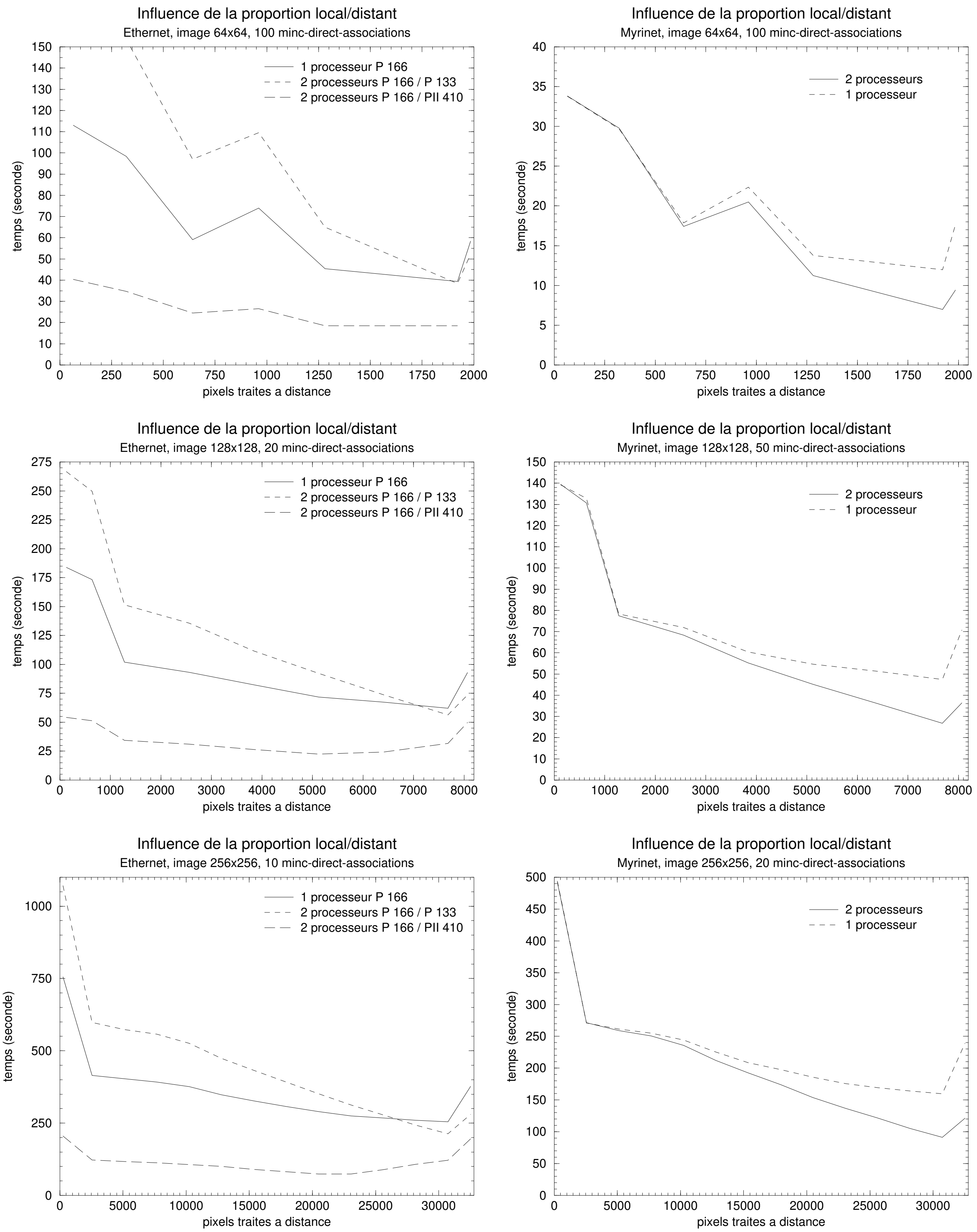


FIG. 136 – *Rapport calcul / communication. Opérateur* minc.

Annexe C

Étude du graphe maille

La définition du modèle des réseaux associatifs fait intervenir un graphe de base, dans lequel, grâce aux masques, il est possible de définir des sous-graphes. L'évaluation de la portée de la technique des masques semblait donc importante.

Dans cette annexe, nous résumons cette étude[(a)], que nous avons centrée sur le graphe maille, la topologie de la Maille Associative. Nous commençons par définir le *plongement un-vers-ensemble-connexe*, conforme aux exigences du modèle des réseaux associatifs. Puis nous donnons une construction indiquant que tout graphe peut ainsi être plongé dans une maille de taille suffisante. Nous améliorons ensuite la construction des plongements de quelques graphes usuels : hypercube, graphe de la transformée de Fourier, et arbres binaires.

Cette étude montre qu'au moins une famille de graphes de base permet d'envisager toute sorte d'objets, codés sous forme de sous-graphes.

C.1 Définitions

C.1.1 Graphe maille

Nous commençons par définir le graphe *maille*, qui est la topologie de la Maille Associative (*cf.* § 1.2.2 page 13).

Définition 108 *Une maille $n \times n$ est un graphe $G(V, E)$ tel que les sommets peuvent être assimilés aux éléments de l'ensemble $\{1, \dots, n\}^2$, et tel qu'il existe un arc du sommet (i, j) au sommet (i', j') si et seulement si* ***(a)*** *$(i, j) \neq (i', j')$,* ***(b)*** *$i' \in \{i - 1 \mod n, i, i + 1 \mod n\}$ et* ***(c)*** *$j' \in \{j - 1 \mod n, j, j + 1 \mod n\}$.*

Le graphe maille peut aussi être défini en tant que graphe de Cayley. Un *graphe de Cayley* orienté est défini à partir d'un groupe $(\mathbb{S}, \diamond)$ et d'un ensemble de générateurs $\mathcal{G}$ de $\mathbb{S}$, de telle sorte que chaque sommet correspond à un élément du groupe, et qu'il existe un arc entre les sommets x et y si et seulement s'il existe un générateur $g \in \mathcal{G}$ tel que $y = x \diamond g$ [Heydemann et Ducourthial, 1997]. Une maille $n \times n$ est un graphe de Cayley défini avec le groupe $(\mathbb{Z}/n\mathbb{Z})^2$ et les générateurs $\mathcal{G} = \{(1, 0), (1, -1), (0, -1), (-1, -1), (-1, 0), (-1, 1), (0, 1), (1, 1)\}$.

Un graphe de Cayley n'a «pas de bord», et cette propriété a des applications dans l'algorithmique et dans les mises en œuvres, logicielles ou matérielles. Néanmoins, certains problèmes algorithmiques admettent par nature des bords, qu'il faut alors prendre en compte. C'est le cas de la segmentation d'images par exemple (*cf.* chapitre 2), où les objets ne doivent pas inclure les connexions toriques, sous peine d'être invalides. Pour ces problèmes, on est donc amené à travailler sur la maille non torique, que nous définissons comme suit.

(a) Les preuves manquantes, des compléments et un état de l'art incluant des références bibliographiques peuvent être trouvés dans [Ducourthial et Mérigot, 1996].

Définition 109 *Une maille non torique $n \times n$ est un graphe orienté symétrique $G(V, E)$ tel que les sommets peuvent être assimilés aux éléments de l'ensemble $\{1, \ldots, n\}^2$, et tel qu'il existe un arc du sommet (i, j) au sommet (i', j') si et seulement si **(a)** $(i, j) \neq (i', j')$, **(b)** $i' \in \{i-1, i, i+1\} \cap \{1, \ldots, n\}$ et **(c)** $j' \in \{j-1, j, j+1\} \cap \{1, \ldots, n\}$.*

La définition d'une maille rectangulaire se déduit aisément de ces définitions. Une maille non torique non orientée est appelée *grille en 8-connexité*, ou encore *Grille étendue*.

C.1.2 Plongement un-vers-ensemble-connexe

La définition d'un plongement de type un-vers-ensemble-connexe est motivée par les caractéristiques du modèle des réseaux associatifs. Il serait en effet préjudiciable de découper en plusieurs phases une `association` sur un graphe plongé dans le graphe de base à cause d'un quelconque «chevauchement» des images des sommets ou des arcs. On préfère donc définir un plongement sans chevauchement, permettant de calculer les `associations` en une seule phase.

Étant donné un graphe $G(V, E)$, l'ensemble des sommets $\mathcal{V} = \{v_1, \ldots, v_k\} \subset V$ est dit *connexe* s'il existe au moins un chemin reliant tout sommet $v_i \in \mathcal{V}$ à tout autre sommet $v_j \in \mathcal{V}$, passant exclusivement par des sommets de $\mathcal{V}$. On note $\mathcal{V}(G)$ l'ensemble des ensembles de sommets connexes d'un graphe G.

Définition 110 *Un plongement un-vers-ensemble-connexe du graphe $G(V_G, E_G)$ dans le graphe $H(V_H, E_H)$ est défini par la donnée de deux applications f_V et f_E telles que :*
- *f_V est une application de V_G dans $\mathcal{V}(H)$ (l'image d'un sommet est un ensemble connexe),*
- *$f_V(u) \cap f_V(v) = \emptyset$ pour tout u et v dans V_G (les ensembles connexes images de sommets distincts ne se chevauchent pas),*
- *f_E est une application de E_G dans E_H (l'image d'un arc est un arc),*
- *pour tout arc (u, v) de E_G, si $f_E(\,(u, v)\,) = (u', v')$, alors $u' \in f_V(u)$ et $v' \in f_V(v)$ (l'image d'un arc relie les images de ses extrémités).*

Le graphe $G(V_G, E_G)$ est appelé *graphe logique*, tandis que le graphe $H(V_H, E_H)$ est appelé *graphe physique*. On parle aussi de sommets, d'arcs et de chemins logiques ou physiques. La figure 133 donne un exemple de plongement de type un-vers-ensemble-connexe : le graphe logique est représenté sur la partie (a) et son plongement dans une maille non torique 3×5 sur la partie (b).

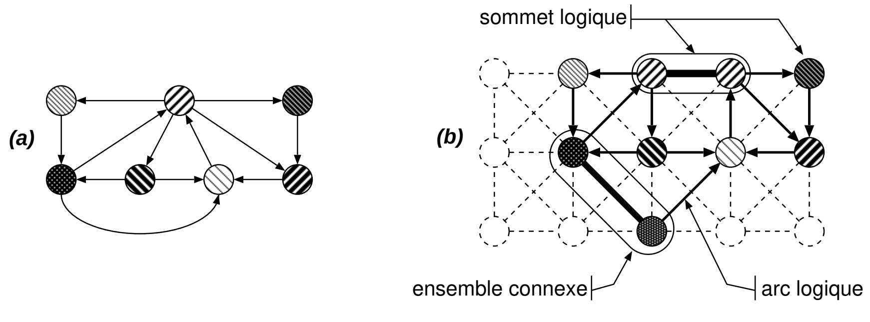


FIG. 133 – *Exemple de plongement un-vers-ensemble-connexe dans une maille non torique* 3×5.

131. Expansion : rapport entre le nombre de sommets du graphe plongé et le nombre de sommets du graphe de base.

Dans [Ducourthial et Mérigot, 1996] [Ducourthial et Mérigot, 1997], nous avons défini le plongement un-vers-ensemble-connexe de telle sorte que l'image d'un arc pouvait être un ensemble connexe, faisant office de chemin. Les deux notions sont semblables. Par contre, les graphes plongés sont plus faciles à manipuler en programmation lorsque les arcs logiques correspondent à des arcs physiques, et non à des ensembles connexes. On retrouve dans ce cas les graphes virtuels (*cf.* § 2.2.1 page 22).

Par définition, les sommets du graphe physique utilisés pour le plongement sont dédiés soit à un seul arc, soit à un seul sommet du graphe logique. Cela permet de calculer une `direct-association` en une seule étape, et ainsi de bénéficier de la propagation asynchrone. En contre-partie, l'expansion[131] du plongement est assez forte.

C.2 Existence de plongements un-vers-ensemble-connexe

On montre que pour tout graphe $G(V, E)$, il existe une maille non-torique de taille suffisante et un plongement un-vers-ensemble-connexe de G dans cette maille. La preuve est basée sur le routage d'une permutation dans la maille.

C.2.1 Routage d'une permutation d'une ligne vers une autre

Nous définissons le problème du routage d'une permutation de n valeurs d'une ligne vers une autre dans une maille non torique $n \times m$ comme suit (le problème est défini de la même façon pour une grille en 8-connexité). On positionne les n valeurs sur l'un des bords de taille n de la maille. Puis on procède (si besoin) à des échanges de valeurs voisines, en utilisant les connexions diagonales de la maille, de façon à obtenir (si possible) la permutation identité $1 \cdots n$ sur le bord opposé (*cf.* figure 134).

Ce problème admet une solution pour toutes les permutations lorsque $m \geq n+1$. Cette borne est atteinte pour certaines permutations (*cf.* figure 135). Pour montrer cela, on utilise les résultats du tri pair-impair de n valeurs [Knuth, 1969].

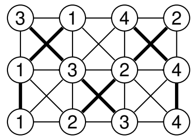


FIG. 134 – Routage d'une permutation de 4 valeurs dans une maille 3×4.

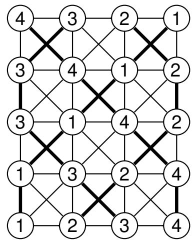


FIG. 135 – Certaines permutations de n valeurs nécessitent une maille de taille $n \times (n+1)$ pour être routées.

C.2.2 Théorème d'existence

Théorème 111 *Pour tout graphe orienté $G(V_G, E_G)$, il existe un plongement un-vers-ensemble-connexe dans une maille $n \times n$ notée $H(V_H, E_H)$, avec $n = \sum_{v \in V_G} (\delta_v^- + 1)$.*

On montre ce théorème en exhibant un plongement sous forme du routage d'une permutation. Pour cela, on associe à chaque sommet v de V_G un *ensemble de départ* $D(v)$ de $(\delta^+(v)+1)$ sommets de V_H, situés consécutivement sur l'un des bords — que l'on appellera *ligne de départ* D — de la maille H. De même, on associe à tout sommet v un *ensemble d'arrivée* $A(v)$ de $(\delta_v^- + 1)$ sommets sur le bord opposé — appelé *ligne d'arrivée* A — de la maille. Puisque $n = \sum_{v \in V_G} (\delta_v^- + 1) = \sum_{v \in V_G} (\delta^+(v) + 1)$, il est possible d'affecter des ensembles disjoints à chaque sommet, que ce soit sur la ligne de départ ou sur la ligne d'arrivée (*cf.* figure 136 *a* et *b*).

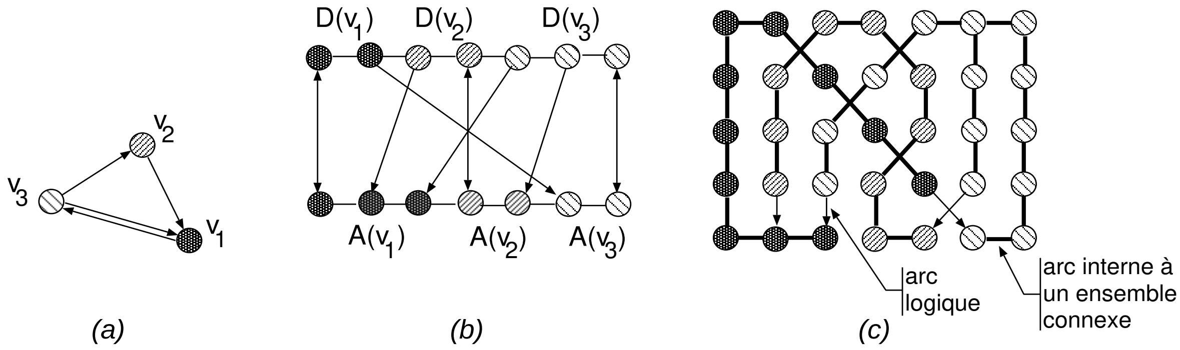


FIG. 136 – *Construction du plongement du théorème 111.*

Par construction, il y a assez de sommets dans les ensembles de départ et d'arrivée pour qu'il puisse partir, de chaque ensemble $D(v)$, un chemin vers l'ensemble $A(v)$, et un chemin vers chaque ensemble $A(w)$ tel que l'arc (v, w) soit dans E_G, et ce, sans que deux chemins ne se partagent la même extrémité (*cf.* figure 136 *b*). Pour obtenir le plongement, il faut réussir à router tous ces chemins dans la maille, depuis la ligne de départ vers la ligne d'arrivée, sans qu'ils ne se chevauchent. Ces chemins déterminent alors des ensembles connexes, images des sommets logiques, ainsi que les arcs logiques les reliant (*cf.* figure 136 *c*).

Les ensembles situés sur les extrémités des lignes de départ et d'arrivée peuvent être reliés en ligne droite (sur la figure 136 *c*, il s'agit des ensembles correspondant aux sommets v_1 et v_3). Le problème revient donc au routage d'une permutation de $n-2$ valeurs, dans une maille $n \times n$, ce qui est toujours possible. Cela donne le théorème.

La construction donnée par la preuve du théorème 111 conduit à un plongement de forte expansion. Nous donnons de meilleurs plongements pour quelques graphes usuels dans les sections suivantes. Par la suite, et par soucis de simplicité, nous considérons une maille non torique non orientée, c'est-à-dire une grille en 8-connexité. Les plongements concernent donc des graphes non orientés, et les représentations s'en trouvent simplifiées. Ces plongements peuvent aisément être adaptés aux graphes orientés.

C.3 Plongement de l'hypercube

La construction que nous proposons pour l'hypercube est basée sur la technique des faisceaux ; nous commençons par la définir.

C.3.1 Technique des faisceaux

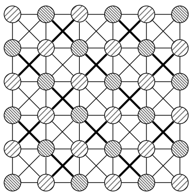


FIG. 137 – Deux faisceaux de diagonales se croisent sans se rencontrer.

Étant donné une grille en 8-connexité, on définit un *faisceau de diagonales* en prenant une diagonale sur deux dans une direction donnée. Lorsque deux faisceaux de directions différentes ne contiennent pas de sommets communs, ils se croisent sans se rencontrer (*cf.* figure 137).

Étant donné deux faisceaux se croisant sans se rencontrer, un plongement un-vers-ensemble-connexe d'un graphe biparti dans la grille en 8-connexité est obtenu en «attribuant» les sommets de la première partition aux diagonales du premier faisceau (l'image d'un sommet logique est une diagonale d'un faisceau), et ceux de la deuxième partition à celles du deuxième faisceau. Puisqu'il est possible d'établir des connexions entre toute paire de diagonales appartenant à des faisceaux différents avec un seul arc, les arcs logiques sont aisément obtenus dans le graphe physique. On peut montrer ainsi que tout graphe biparti, dont la première partition contient p sommets et la seconde q, admet un plongement de type un-vers-ensemble-connexe dans une grille en 8-connexité de côté $(2 \times \mathsf{max}(p, q) - 2)$.

De plus, la technique des faisceaux permet d'obtenir une construction itérative pour les plongements de graphes admettant eux-mêmes une construction itérative. La figure 138 indique comment compléter les faisceaux de diagonales lorsque le graphe logique est complété par deux sommets à chaque étape (un dans chaque partition). Les diagonales existantes sont allongées pour autoriser des connexions avec les nouveaux sommets logiques.

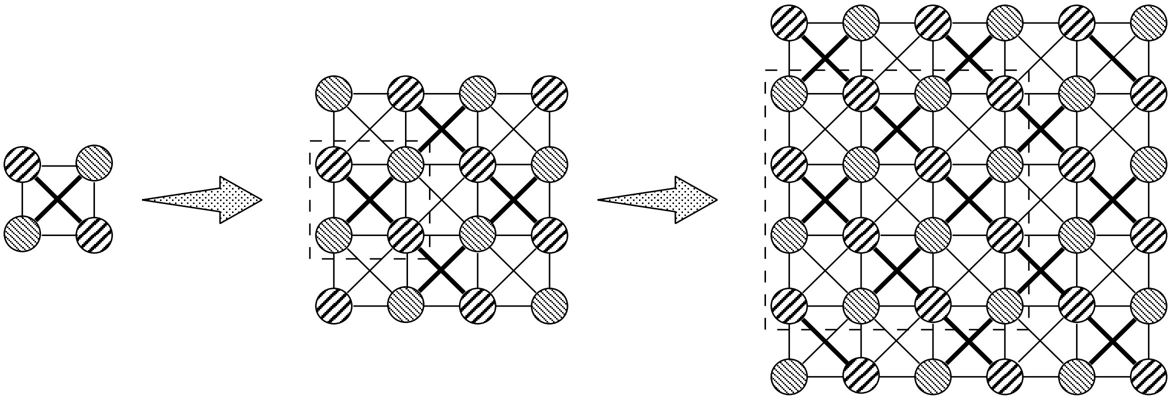


FIG. 138 – *Construction récursive de plongements avec les faisceaux.*

C.3.2 Application à l'hypercube

L'hypercube $H(n)$ de dimension n est tel que les sommets peuvent être assimilés aux mots binaires de n bits. Une arête relie un sommet à un autre si les mots binaires correspondants diffèrent d'un seul bit[b]. C'est un graphe biparti et on peut donc utiliser la technique des faisceaux pour

(b) L'hypercube $H(n)$ peut aussi être défini comme le graphe de Cayley sur le groupe $(\mathbb{Z}/2\mathbb{Z})^n$, avec les n générateurs $(0, \ldots, 0, 1, 0, \ldots, 0)$.

obtenir son plongement.

On montre ainsi que $H(n)$ peut être plongé dans une grille en 8-connexité de côté $2^n - 2$ (*cf.* figure 139).

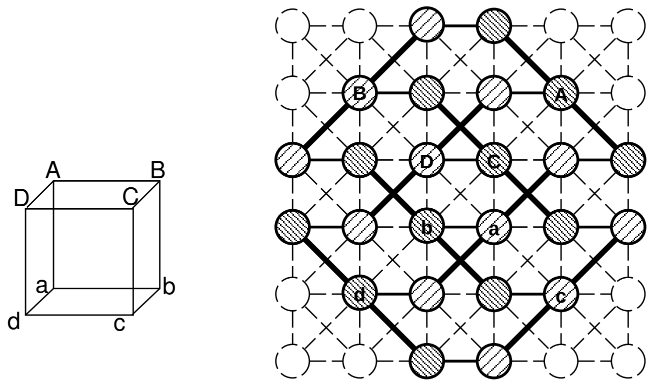


FIG. 139 – *Plongement un-vers-ensemble-connexe de $H(3)$.*

C.4 Plongement du graphe de la transformée de Fourier rapide

Le graphe de la transformée de Fourier rapide[132] $FFT(n)$ est un graphe multi-étages, dont les sommets sont les couples $(l, x_{n-1} \cdots x_0)$, avec $l \in \{0, \ldots, n\}$, et $x_i \in \{0, 1\}$. Il y a un arc entre le sommet $(l, x_{n-1} \cdots x_0)$ et les sommets $(l+1, x_{n-1} \cdots x_0)$ et $(l+1, x_{n-1} \cdots \overline{x_l} \cdots x_0)$ $(l < n)$. Ce graphe correspond au graphe de précédence de l'algorithme de la transformée de Fourier rapide (*cf.* § 7.3 page 136). Il est composé d'*étages*, que relient les arcs (*cf.* figure 140 gauche).

132. En anglais, *Fast Fourier Transform*, ou *FFT*.

La construction d'un plongement un-vers-ensemble-connexe que nous proposons pour $FFT(n)$, et qui est illustrée sur la figure 140 droite, utilise les routages de permutations. En effet, le passage d'un étage à un autre correspond au routage d'une permutation particulière. Le plongement d'un étage revient alors au routage de la permutation dans une grille en 8-connexité. Sur la figure 140 droite, les étages sont signalés par les sommets physiques gras (les images de certains sommets logiques sont coloriées). La nature régulière des étages, et la permutation particulière qu'il faut router pour chacun d'eux, assurent une expansion optimale du plongement de $FFT(n)$ en étages. Une grille en 8-connexité $2^{n+1} \times (2^{n+1} - n)$ convient.

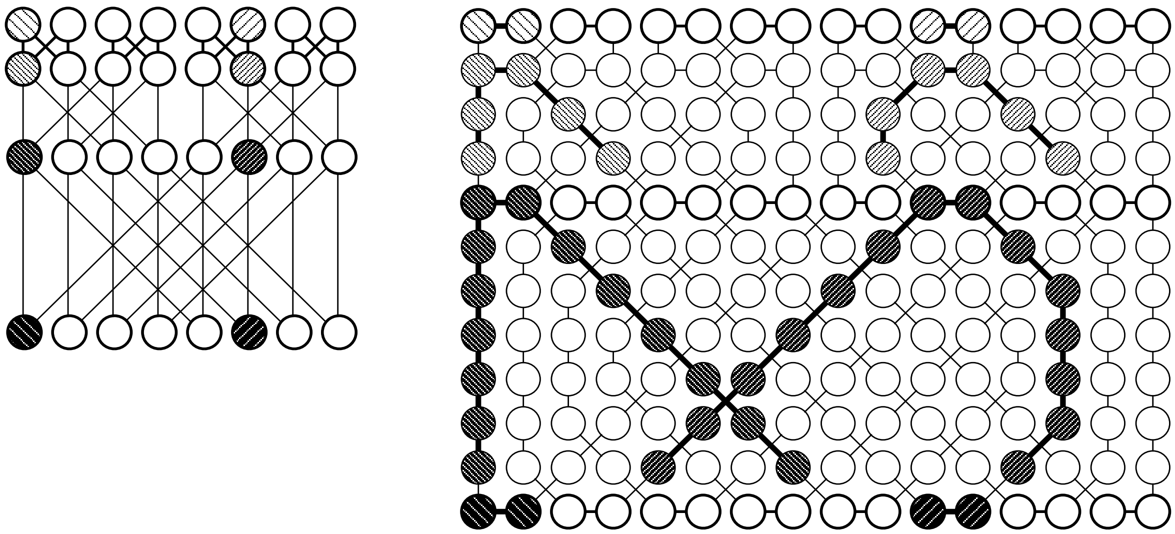


FIG. 140 – *Plongement un-vers-ensemble-connexe de $FFT(3)$.*

C.5 Plongement des arbres binaires

L'arbre binaire complet T_n de hauteur n est défini récursivement comme suit. T_0 est réduit à un seul sommet, appelé *racine*. Pour $n \geq 1$, T_n est obtenu à partir de deux arbres T_{n-1}, en connectant leurs racines à un nouveau sommet, racine de T_n.

Le plongement un-vers-ensemble-connexe d'un arbre binaire complet dans une grille peut être réalisé avec un pavage de la grille par des tuiles. Une *tuile* est une «petite» grille, contenant le plongement d'un «petit» arbre binaire. Les plongements d'arbres binaires plus grands sont obtenus par assemblage de plusieurs tuiles, comme illustré sur la figure 141. Les constructions les plus simples utilisent des tuiles carrées ; nous proposons aussi un plongement basé sur des tuiles rectangles.

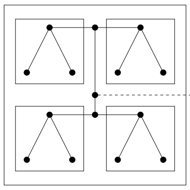


FIG. 141 – Plongements d'arbres binaires avec un pavage récursif carré.

C.5.1 Tuiles carrées

L'assemblage de quatre tuiles contenant l'arbre T_k donne l'arbre T_{k+2}, composé de quatre arbres T_k correctement connectés (*cf.* figure 141). Pour obtenir tous les arbres binaires, il est donc nécessaire de disposer à la fois d'une tuile contenant un arbre de hauteur paire T_p (p est pair), et d'un arbre de hauteur impaire T_q (q est impair). Les arbres de hauteur paire T_{2k} seront obtenus par assemblage de tuiles contenant T_p, tandis que les arbres de hauteur impaire T_{2k+1} seront obtenus par assemblage de tuiles contenant T_q. Par ailleurs, il est nécessaire de laisser un espace entre les tuiles pour plonger les racines sans emprunter des ensembles connexes déjà utilisés. En supposant que les tuiles initiales sont optimales, plus celles-ci seront grandes, moins il y aura d'espace perdu, et meilleure sera l'expansion finale du plongement. Néanmoins, une construction à partir de tuiles contenant l'arbre T_k ne permet pas d'obtenir T_l, pour $l < k$, et les cas particuliers sont plus nombreux lorsque la tuile initiale est grande.

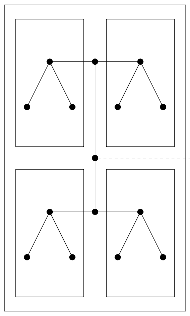


FIG. 142 – Construction récursive des tuiles rectangles.

Soit m le côté de la tuile initiale contenant l'arbre T_p et m' celui contenant l'arbre $T_{p'}$, avec p impair et p' pair. On montre que le côté $c(n)$ de la grille contenant $T(n)$ est donné par :

$$c(n) = \begin{cases} 2^{\frac{n-p}{2}} \times (m+1) - 1 & \text{si } n \text{ est impair et si } n \geq p \\ 2^{\frac{n-p'}{2}} \times (m'+1) - 1 & \text{si } n \text{ est pair et } n \geq p' \end{cases} \tag{C.1}$$

C.5.2 Tuiles rectangles

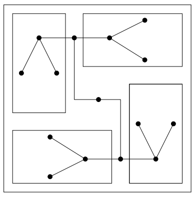


FIG. 143 – Étape finale de la construction à base de tuiles rectangles ; la grille est carrée.

Pour plonger l'arbre T_k, la technique des tuiles rectangles consiste à appliquer le procédé d'assemblage de la figure 142, jusqu'à obtenir un plongement de l'arbre T_{k-2} dans un assemblage rectangle de tuiles rectangles. Pour que le plongement de T_k soit obtenu dans une grille carrée (ce qui est généralement souhaité), on assemble alors les tuiles contenant T_{k-2} selon le schéma de la figure 143.

Supposons que l'arbre T_p, de hauteur p impaire, soit contenu dans la tuile $m_1 \times m_2$, tandis que l'arbre T'_p, de hauteur p' paire soit contenu dans la tuile $m'_1 \times m'_2$. On montre alors que le côté $c(n)$ de la grille contenant l'arbre T_n est donné par :

$$c(n) = \begin{cases} 2^{\frac{n-p}{2}} \times \left(\frac{m_1+m_2}{2} + 1\right) - 1 & \text{si } n \text{ est impair et } n \geq p+2 \\ 2^{\frac{n-p'}{2}} \times \left(\frac{m'_1+m'_2}{2} + 1\right) - 1 & \text{si } n \text{ est pair et } n \geq p'+2 \end{cases} \tag{C.2}$$

Ainsi, lorsque le plongement de l'arbre T_p d'une tuile carrée peut être obtenu dans une tuile rectangle de périmètre inférieur (*i.e.*, $m_1 + m_2 < 2m$), l'expansion du plongement à base de cette tuile rectangle sera meilleure que celle du plongement utilisant la tuile carrée.

C.5.3 Application à la grille en 4-connexité

La figure 144 page ci-contre donne des tuiles carrées pour la grille en 4-connexité (sans diagonale). Le plongement récursif utilisant la tuile paire (contenant T_2) est appelé méthode des H-*trees* [Ullman, 1984]. Cette méthode est notamment utilisée pour distribuer une horloge dans les circuits électroniques, grâce à sa «régularité» (symétries dans les motifs). Ces tuiles carrées conduisent à un plongement de T_n dans une grille en 4-connexité de côté $c(n)$ donné par :

$$c(n) = \begin{cases} 3 \times 2^{\frac{n-1}{2}} - 1 & \text{si } n \text{ est impair et si } n \geq 1 \\ 2 \times 2^{\frac{n}{2}} - 1 & \text{si } n \text{ est pair et si } n \geq 2 \end{cases} \tag{C.3}$$

D'une manière générale, plus les tuiles sont grandes, et plus on peut espérer tirer un bénéfice en les réarrangeant. Ici, pour obtenir des tuiles rectangles de périmètre plus avantageux que les tuiles carrées, il est nécessaire de commencer la construction à partir des arbres T_3 et T_4. La figure 145 donne les tuiles correspondantes. Ces tuiles rectangles donnent les valeurs suivantes de $c(n)$:

$$c(n) = \begin{cases} \frac{11}{4} \times 2^{\frac{n-1}{2}} - 1 & \text{si } n \text{ est impair et si } n \geq 5 \\ \frac{15}{8} \times 2^{\frac{n}{2}} - 1 & \text{si } n \text{ est pair et si } n \geq 6 \end{cases} \tag{C.4}$$

En particulier, la méthode du pavage récursif rectangle améliore la technique des *H-trees* : le côté de la grille nécessaire au plongement est diminué d'environ 7 %.

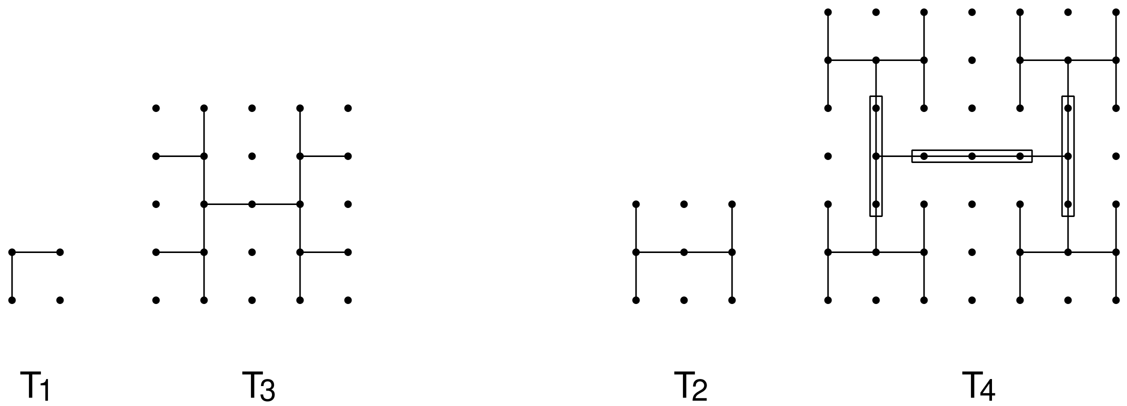


FIG. 144 – *Tuiles carrées pour la grille en 4-connexité, et premier pas de récurrence (T_1 et T_2).*

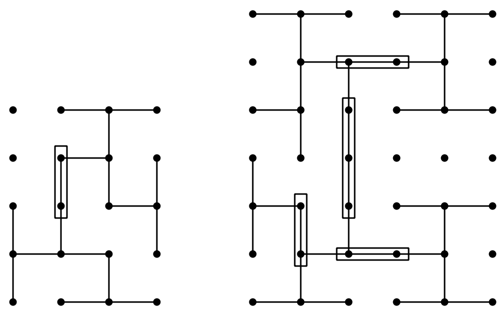


FIG. 145 – *Tuiles rectangles pour la grille en 4-connexité (T_3 et T_6).*

C.5.4 Application à la grille en 8-connexité

Pour que les diagonales puissent être exploitées dans les tuiles de base, il est nécessaire de commencer la construction à pavage carré à partir des arbres T_3 et T_4 (aucun bénéfice n'est tiré des diagonales pour des tuiles plus petites). La figure 146 page suivante donne les tuiles correspondantes[c]. La valeur de $c(n)$ correspondante est donnée par :

$$c(n) = \begin{cases} \frac{5}{2} \times 2^{\frac{n-1}{2}} - 1 & \text{si } n \text{ est impair et si } n \geq 3 \\ \frac{7}{4} \times 2^{\frac{n}{2}} - 1 & \text{si } n \text{ est pair et si } n \geq 4 \end{cases} \tag{C.5}$$

(c) Ces tuiles ont été trouvées à l'aide d'un programme.

Le côté de la grille est diminué d'environ 14 % par rapport à la méthode des H-*trees*.

Pour obtenir des tuiles rectangles de périmètre plus avantageux, il est nécessaire de commencer la récursion à partir des tuiles T_5 et T_6. Les tuiles correspondantes sont données par la figure 147. La valeur de $c(n)$ correspondante est donnée par :

$$c(n) = \begin{cases} \frac{19}{8} \times 2^{\frac{n-1}{2}} - 1 & \text{si } n \text{ est impair et si } n \geq 7 \\ \frac{27}{16} \times 2^{\frac{n}{2}} - 1 & \text{si } n \text{ est pair et si } n \geq 8 \end{cases} \tag{C.6}$$

Le côté de la grille est alors diminué d'environ 18 %.

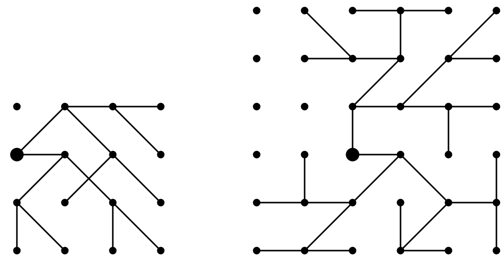


FIG. 146 – *Tuiles carrées pour la grille en 8-connexité (T_3 et T_4).*

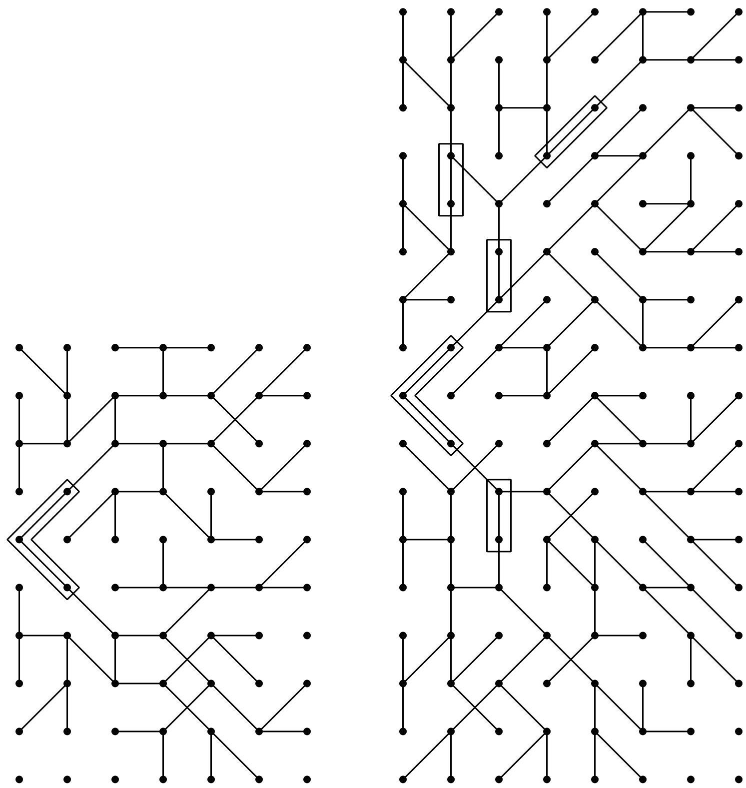


FIG. 147 – *Tuiles rectangles pour la grille en 8-connexité (T_5 et T_6).*

C.6 Conclusion

Le modèle des réseaux associatifs autorise la définition de nombreux objets dans le graphe de base, grâce aux masques. Cependant, il induit des contraintes, et conduit à la définition des plongements un-vers-ensemble-connexe.

Le graphe maille est un exemple de topologie s'affranchissant de ces contraintes, et permettant de plonger tout type de graphes.

Bibliographie

AFEK, T., KUTTEN, S., et YUNG, M. (1990). « Memory-efficient self-stabilization on general networks ». Dans *Proceedings of Workshop on Distributed Algorithms*, pages 15–28.

AHO, A., HOPCROPFT, J., et ULLMAN, J. (1974). *The Design and Analysis of Computer Algorithms*. Series in Computer Science and Information Processing. Addison-Wesley Publishing, Reading, Ma, USA.

AHO, A. V., HOPCROPFT, J. E., et ULLMAN, J. D. (1983). *Data Structures and Algorithms*. Addison-Wesley Publishing, Reading, Ma, USA.

AHUJA, N., AN, B., et SHACHTE, B. (1985). « Image Representation Using Voronoï Tesselation ». *CVGIP*, 29:286–295.

AKIAN, M., COHEN, G., GAUBERT, S., VIOT, M., et QUADRAT, J.-P. (1994). « Max-Plus algebra and applications to system theory and optimal control ». Dans *Proceedings of the International Congress of Mathematicians*, Zurich, Switzerland. Août 1994. (*cf.* `http://amadeus.inria.fr/TROPICAL/`).

ALIMA, L. O., BEAUQUIER, J., DATTA, A. K., et TIXEUIL, S. (1998). « Self-stabilization with Global Rooted Synchronizers ». Dans *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*.

ALSTEIN, D., HOEPMAN, J.-H., OLIVIER, B., et VAN DER PUT, P. (1994). « Self-stabilizing mutual exclusion on directed graphs ». Dans BACKER, E., éditeur, *Proceedings of Computer Science in the Netherlands, Utrech, The Netherlands*, pages 42–53, Stichting Mathematisch Centrum, Amsterdam.

ARORA, R. K., RANA, S. P., et GUPTA, M. N. (1986). « Distributed Termination Detection Algorithm For Distributed Computations ». *Information Processing Letters*, 22:311–314. Mai 1986.

AWERBUCH, B. (1985). « A new distributed depth-first-search algorithm ». *Information Processing Letters*, 20:147–150. Avril 1985.

AXELROD, T. (1986). « Effects of synchronisation barriers on multiprocessor performance ». *Parallel Computing*, 3:129–140.

BACCELLI, F., COHEN, G., OLSDER, G., et QUADRAT, J.-P. (1992). *Synchronisation and Linearity, an algebra for discrete event systems*. Series in Probability and Mathematical Statistics. Wiley, Chichester, UK.

BAUDET, G. M. (1978). « Asynchronous iterative methods for multiprocessors ». *Journal of the ACM*, 25(2):226–244. Avril 1978.

BEAUQUIER, J. et BÉRARD, B. (1991). *Systèmes d'Exploitation : Concepts et Algorithmes*. Collection Informatique. Mc Graw-Hill, Paris, France.

BILARDI, G. et PREPARATA, F. (1990). « Characterization of associative operations with prefix circuits of constant depth and linear size ». *SIAM Journal of Computing*, 19(2):246–255. Avril 1990.

BLELLOCH, G. (1990a). « Prefix sums and their applications ». Rapport Technique CMU-CS-90-190, School of Computer Science, Carnegie-Mellon University, Pittsburg, PA, USA. Novembre 1990. *(aussi dans Synthesis of Parallel Algorithms, J.H. Reif Éditeur).*

BLELLOCH, G. (1990b). *Vector Models for Data-Parallel Computing*. Artificial Intelligences Series. The MIT Press, Cambridge, Ma, USA.

BORGEFORS, G. (1986). « Distance transformations in digital images ». *Computer vision, graphics, and image processing*, 34:344–371.

BOUGÉ, L. (1996). « *The Data Parallel Programming Model* », Chapitre The data-parallel programming model: a semantic perspective, pages 4–26. Lecture Notes on Computer Science Tutorial Series (LNCS-TS). Springer Verlag. G.-R. Perrin et A. Darte éditeurs.

BOUGÉ, L., CACHERA, D., LE GUYADEC, Y., UTARD, G., et VIROT, B. (1997). « Formal validation of data-parallel programs: a two-component assertional proof system for a simple language ». *Theoretical Computer Science*, 189(1-2):71–107. Décembre 1997.

BOUGÉ, L. et LEVAIRE, J.-L. (1992). « Control structures for data-parallel SIMD languages: semantic and implementation ». *Future Generation Computer Systems*, 8:363–378.

BOUGÉ, L., MÉHAUT, J.-F., et NAMYST, R. (1998). « Madeleine : An efficient and portable communication interface for RPC-based multithreaded environments ». Rapport Technique 98-26, LIP, École Normale Supérieure, Lyon, France.

BOUGÉ, L. et PEREZ, C. (1998). « Production de code à base de processus légers migrables pour le compilateur à parallélisme de données HPF Adaptor ». Dans MÉRY, D. et PERRIN, G.-R., éditeurs, *Actes des Dixièmes Rencontres Francophones du Parallélisme (RenPar 10)*, pages 7–10, Starsbourg, France. Juin 1998.

BOURBAKI, N. (1964). *Algèbre*. Éléments de Mathématiques. Hermann, Paris, France, seconde edition. Fascicule IV, Livre II.

BOURGON, B., DAS, S., DATTA, A., et NATARAJAN, V. (1995). « A Self-Stabilizing Shortest Path Algorithm in a DAG ». Dans *Proceedings Fourteenth IEEE International Phoenix Conference on Computers and Communications, Scottsdale, Ar. USA*, pages 341–345. Mars 1995.

BRÉGIER, F. (1998). « Proposition de gestion de l'irrégularité avec HPF2 ». Dans MÉRY, D. et PERRIN, G.-R., éditeurs, *Actes des Dixièmes Rencontres Francophones du Parallélisme (RenPar 10)*, pages 11–14, Starsbourg, France. Juin 1998.

BRIAT, J., GEIB, J.-M., et MÉHAUT, J.-F. (1997a). Technologie et objectifs des supports d'exécution pour la régularisation. Dans BARTH, D., Chassin de KERGOMMEAUX, J., ROCH, J.-L., et ROMAN, J., éditeurs, *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICARE'97)*, Chapitre 6, pages 125–165. Décembre 1997. École CNRS PRC PRS (Capa, Rumeur, Exec).

BRIAT, J., GINZBURG, I., PASIN, M., et PLATEAU, B. (1997b). « Athapascan Runtime: Efficiency for Irregular Problems ». Dans LENGAUER, C., GRIELL, M., et GORLATCH, S., éditeurs, *Proceedings of Euro-Par'97, Third International Parallel Processing Conference*, volume 1300 de *Lecture Notes on Computer Science (LNCS)*, pages 591–600, Passau, Germany. Août 1997. Springer Verlag.

BRIGGS, P. (1997). « *The Tera MTA* ». Tera Computer Company, 5, Villa Alexandrine, 92100 Boulogne, France.

BROOKES, S. (1993). « Historical introduction to "Concrete domains" by G. Kahn and G.D. Plotkin ». *Theoretical Computer Science*, 121(1-2):179–186.

CALVIN, C. et COLOMBET, L. (1994). « Introduction au Modèle de Programmation Par Processus Communicants : deux exemples PVM et MPI ». Rapport APACHE 12, IMAG, Grenoble, France. Juillet 1994.

CAPPELLO, F. et RICHARD, O. (1998). « Architectures parallèles à partir de réseaux de stations de travail : réalités, opportunités et enjeux ». *Calculateurs Parallèles*, 10(1):9–34. Février 1998.

CHAJAKIS, E. et ZENIOS, S. (1991). « Synchronous and asynchronous implementations of relaxation algorithms for non linear network optimisation ». *Parallel Computing*, 17:873–894.

CHANG, E. J. H. (1982). « Echo algorithms: depth parallel operations on general graphs ». *IEEE Transactions on Software Engineering*, SE 8(4):391–401. Juillet 1982.

CHAROT, F. (1993). « Architectures parallèles spécialisées pour le traitement d'images ». Publication Interne 722, IRISA, Campus de Beaulieu, Rennes, France. Avril 1993.

CHEUNG, T.-Y. (1983). « Graph traversal techniques and the maximum flow problem in distributed computation ». *IEEE Transactions on Software Engineering*, 9(4):504–512. Juillet 1983.

CHUNG, K.-L. (1995). « Prefix computations on a generalized mesh-connected computer with multiples buses ». *IEEE Transactions on Parallel and Distributed Systems*, 6(2):196–199. Février 1995.

CIARLET, P. G. (1982). *Introduction à l'Analyse Numérique Matricielle et à l'Optimisation*. Collection Mathématiques Appliquées pour la Maîtrise. Masson, Paris, France.

CIDON, I. (1988). « Yet another distributed deph-first-search algorithm ». *Information Processing Letters*, 26:301–305. Janvier 1988.

COCHET-TERRASSON, J., GAUBERT, S., et GUNAWARDENA, J. (1998). Dynamics of min-max functions. Dans *Algèbre Max-Plus et applications en informatique et automatique*, Support de cours, pages 459–496. INRIA, Rocquencourt, France. Mai 1998. (26$^{\text{ème}}$ École de printemps d'informatique théorique, Ile de Noirmoutier, France).

COHEN, G., MOLLER, P., QUADRAT, J.-P., et VIOT, M. (1985). « A linear system-theoretic view of discrete event processes and its use for performance evaluation in manufacturing ». *IEEE Transactions on Automatic Control*, 30:210–220.

COHN, E. (1990). « Implementing the multiprefix operation efficiently ». *Journal of Parallel and Distributed Computing (JPDC)*, 10:29–34.

CORMEN, T., LEISERSON, C., et RIVEST, R. (1994). *Introduction à l'algorithmique*. Dunod, Paris, France.

de la TORRE, P. et KRUSKAL, C. (1995). « Fast parallel algorithms for all-sources lexicographic search and path-algebra problems ». *Journal of the Algorithms*, 19:1–24.

de RUMEUR, J. (1994). *Communication dans les réseaux de processeurs*. Masson, Paris, France.

DEHNE, F. et KLEIN, R. (1994). « "The Big Sweep": On the power of beach line approach for Voronoï diagrams ». Dans PRIVARA, éditeur, *Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science (MFCS '94)*, volume 841 de *Lecture Notes on Computer Science (LNCS)*, pages 296–305. Springer Verlag.

DEKEYSER, J.-L., KOKOSZKO, B., LEVAIRE, J.-L., et MARQUET, P. (1996). « Irregular data-parallel objects in C++ ». Dans *Proceedings of the Second International Meeting on Vector and Parallel Processing, Systems and Applications (VECPAR'96)*, Porto, Portugal. Septembre 1996.

DEKEYSER, J.-L., LAZURE, D., et MARQUET, P. (1994). « A geometrical data-parallel language ». *ACM SigPlan Notices*, 29(4):31–40. Avril 1994.

DEKEYSER, J.-L. et MARQUET, P. (1996). « *The Data Paralel Programming Model* », Chapitre Supporting irregular and dynamic computations in data-parallel languages, pages 197–219. Lecture Notes on Computer Science Tutorial Series (LNCS-TS). Springer Verlag. G.-R. Perrin et A. Darte éditeurs.

DELAËT, S. (1995). « *Auto-stabilisation : Modèle et Applications à l'Exclusion Mutuelle* ». Thèse de Doctorat, Laboratoire de Recherche en Informatique, bât. 490, Université Paris-Sud, 91405 Orsay, France. Décembre 1995.

DEMEURE, I. et MÉHAUT, J.-F. (1997). Supports d'exécution : paradigmes. Dans BARTH, D., Chassin de KERGOMMEAUX, J., ROCH, J.-L., et ROMAN, J., éditeurs, *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICARE'97)*, Chapitre 1, pages 3–25. Décembre 1997. École CNRS PRC PRS (Capa, Rumeur, Exec).

DENNEULIN, Y. (1997). « Programmation parallèle avec priorités et régulation de charge ». Dans SCHIPER, A. et TRYSTRAM, D., éditeurs, *Actes des Neuvièmes Rencontres Francophones du Parallélisme (RenPar 9)*, pages 171–174, EPFL, Lausanne, Suisse. Mai 1997.

DENNEULIN, Y. et MAUTOR, T. (1997). Techniques de régulation de charge – Applications en optimisation combinatoire. Dans BARTH, D., Chassin de KERGOMMEAUX, J., ROCH, J.-L., et ROMAN, J., éditeurs, *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICARE'97)*, Chapitre 10, pages 215–228. Décembre 1997. École CNRS PRC PRS (Capa, Rumeur, Exec).

DIJKSTRA, E. W. (1974). « Self stabilizing systems in spite of distributed control ». *Communication of the ACM*, 17:643–644.

DIJKSTRA, E. W. et SCHOLTEN, C. S. (1980). « Termination Detection For Diffusing Computations ». *Information Processing Letters*, 11(1):1–4. Août 1980.

DINNING, A. (1989). « A survey of synchronisation methods for parallel computers ». *IEEE Computer*, pages 66–77. Juillet 1989.

DION, E. (1997). *Invitation à la théorie de l'information*. Numéro S118 dans Points Sciences. Seuil, Paris, France. Mars 1997.

DOLEV, S., GOUDA, M. G., et SCHNEIDER, M. (1996). « Memory requirements for silent stabilization ». Dans *15th annual ACM Symposium on the Principles Of Distributed Computing (PODC'96)"*, pages 27–34.

DOLEV, S., ISRAELI, A., et MORAN, S. (1993). « Self-stabilization of dynamic systems assuming only read/write atomicity ». *Distributed Computing*, 7:3–16.

DU, C. (1998). « A note on finding nearest neighbours and constructing Delaunay triangulation in the plane ». *Communications in numerical methods in engineering*, 14:871–877.

DUCOURTHIAL, B. (1995). « Etude du Modèle Réseau Associatif ». Mémoire de DEA Architecture Parallèle, Laboratoire de Recherche en Informatique, bât. 490, Université Paris-Sud, Orsay, France. Juin 1995.

DUCOURTHIAL, B. (1997). « Properties of the r-operators to insure termination of distributed computations ». Rapport Technique 1140, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France. Décembre 1997.

DUCOURTHIAL, B. (1998a). « Demi-vagues utilisant des r-opérateurs : applications et terminaison ». Dans MÉRY, D. et PERRIN, G.-R., éditeurs, *Actes des Dixièmes Rencontres Francophones du Parallélisme (RenPar 10)*, pages 173–176, Strasbourg, France. Juin 1998.

DUCOURTHIAL, B. (1998b). « Generalization of the r-operators ». Rapport Technique 1181, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France. Juin 1998.

DUCOURTHIAL, B. (1998c). « New operators for computing with *associative nets* ». Dans *Proceedings of the Fifth International Colloquium on Structural Information and Communication Complexity (SIROCCO'98), Amalfi, Italie*. Juin 1998. Carleton Scientific. (à paraître).

DUCOURTHIAL, B. (1998d). « Properties of the r-operators to insure termination of distributed computations ». Dans *17th annual ACM Symposium on the Principles Of Distributed Computing (PODC'98)"*, page 325, Puerto Vallarta, Mexique. Juin 1998. ACM, ACM Press. *(communication courte)*.

DUCOURTHIAL, B., CONSTANTINESCU, G., et MÉRIGOT, A. (1998). « Implementing Image Analysis with a Graph Based Parallel Computing Model ». *Computing Supplement*, 12:111–121.

DUCOURTHIAL, B. et MÉRIGOT, A. (1996). « Graph Embedding in the Associative Mesh Model ». Rapport Technique 96-02, Institut d'Electronique Fondamentale, Bât. 220. Université Paris-Sud, 91405 Orsay, France. Décembre 1996. *(soumis à IEEE Transaction on Computer)*.

DUCOURTHIAL, B. et MÉRIGOT, A. (1997). « Plongement de graphes dans la Maille Associative ». Dans SCHIPER, A. et TRYSTRAM, D., éditeurs, *Actes des Neuvièmes Rencontres Francophones du Parallélisme (RenPar 9)*, pages 131–134, EPFL, Lausanne, Suisse. Mai 1997.

DUCOURTHIAL, B. et MÉRIGOT, A. (1998). « Image analysis with r-operators ». Rapport Technique 98-03, Institut d'Electronique Fondamentale, Bât. 220. Université Paris-Sud, 91405 Orsay, France. Août 1998. *(soumis à The Computer Journal)*.

DUCOURTHIAL, B. et SICARD, N. (1998). « ANET : une bibliothèque de programmation data-parallèle à primitives de calcul asynchrones multi-threadée ». Rapport Technique 98-03, Institut d'Electronique Fondamentale, bât. 220, Université Paris-Sud, 91405 Orsay, France.

DUCOURTHIAL, B. et TIXEUIL, S. (1998a). « Self-stabilizing global computations with r-operators ». Dans *Proceedings of the Second International Conference On Principles Of Distributed Systems OPODIS'98, Amiens, France*, page . Décembre 1998. (*à paraître*).

DUCOURTHIAL, B. et TIXEUIL, S. (1998b). « Self-Stabilizing Global Computations with r-operators ». Rapport Technique 1182, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France. Juillet 1998.

DULAC, D. (1996). « *Contribution au parallélisme massif en analyse d'images : une architecture SIMD fondée sur la reconfigurabilité et l'asynchronisme* ». Thèse de Doctorat, Institut d'Electronique Fondamentale, bât. 220, Université Paris-Sud, 91405 Orsay, France. Janvier 1996.

DULAC, D., MÉRIGOT, A., et MOHAMMADI, S. (1993). « Associative Meshes: a New Parallel Architecture for Image Analysis Applications ». Dans *Proceedings of the 1993 Workshop on Computer Architecture for Machine Perception (CAMP'93)*, pages 393–399. Décembre 1993.

DULAC, D., MOHAMMADI, S., et MÉRIGOT, A. (1995). « Implementation and Evaluation of a Parallel Architecture Using Asynchronous Communications ». Dans CANTONI, V., LOMBARDI, L., et OTHERS, éditeurs, *Proceedings of the 1995 Workshop on Computer Architecture for Machine Perception (CAMP'95)*, pages 106–111, Pavia, Italia. Septembre 1995. IEEE Press.

DUNCAN, R. (1990). « A survey of parallel computer architectures ». *IEEE Computer*, pages 5–16. Février 1990.

ECO, U. (1981). *Le nom de la rose*. Numéro 5859 dans Le Livre de Poche. Grasset.

EL BAZ, D., SPITERI, P., MIELLOU, J.-C., et GAZEN, D. (1996). « Asynchronous Iterative Algorithms with Flexible Communications for Non-Linear Network Flow Problems ». *Journal of Parallel and Distributed Computing (JPDC)*, 38:1–15.

FERREIRA, A. et FRAIGNIAUD, P. (1997). Impact du modèle sur l'algorithmique. Dans BARTH, D., Chassin de KERGOMMEAUX, J., ROCH, J.-L., et ROMAN, J., éditeurs, *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICARE'97)*, Chapitre 16, pages 383–403. Décembre 1997. École CNRS PRC PRS (Capa, Rumeur, Exec).

FOSTER, I., KESSELMAN, C., et TUECKE, S. (1996). « The Nexus Approach to Integrating Multithreading and Communication ». *Journal of Parallel and Distributed Computing (JPDC)*, 37:70–82.

GALILÉE, F. et ROCH, J.-L. (1997). Langages pour l'expression dynamique de parallélisme et graphes de tâches. Dans BARTH, D., Chassin de KERGOMMEAUX, J., ROCH, J.-L., et ROMAN, J., éditeurs, *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICARE'97)*, Chapitre 9, pages 199–213. Décembre 1997. École CNRS PRC PRS (Capa, Rumeur, Exec).

GAUBERT, S. (1998). Exotic semirings: examples and general results. Dans *Algèbre Max-Plus et applications en informatique et automatique*, Support de cours, pages 105–136. INRIA, Rocquencourt, France. Mai 1998. (26$^{\text{ème}}$ École de printemps d'informatique théorique, Ile de Noirmoutier, France).

GAUBERT, S. et LOISEAU, J.-J. (1997). « Quelques problèmes d'actualités sur les algèbres tropicales ». (*cf.* `http://amadeus.inria.fr/TROPICAL/`). Mai 1997.

GAUBERT, S. et PLUS, M. (1997). « Methods and Applications of $(\max, +)$ Linear Algebra (survey) ». Dans REISCHUK, R. et MORVAN, M., éditeurs, *Proc. STACS'97, Lübeck*, volume 1200 de *Lecture Notes on Computer Science (LNCS)*. Springer Verlag. (aussi dans le rapport INRIA 3088).

GAUTIER, T., ROCH, J.-L., et VILLARD, G. (1995). « Regular versus irregular problems and algorithms ». Dans *Proceedings Irregular'95*, volume 980 de *Lecture Notes on Computer Science (LNCS)*, pages 1–26. Springer Verlag.

GONDRAN, M. (1975). « Algèbre linéaire et cheminement dans les graphes ». *Revue Française d'Automatique, d'Informatique et Recherche Opérationnelle (RAIRO)*, 9(1):77–99. Janvier 1975.

GONDRAN, M. (1996a). « Analyse MINMAX ». *Comptes-rendus de l'Académie des Sciences (CRAS)*, Série I(323):1249–1252.

GONDRAN, M. (1996b). « Analyse MINPLUS ». *Comptes-rendus de l'Académie des Sciences (CRAS)*, Série I(323):371–375.

GONDRAN, M. et MINOUX, M. (1977). « Valeurs propres et vecteurs propres dans les dioïdes et leur interprétation en théorie des graphes ». *EDF Bulletin de la Direction des Études et Recherches. Série C. Mathématiques Informatique*, 2:25–41.

GONDRAN, M. et MINOUX, M. (1978). « L'indépendance linéaire dans les dioïdes ». *EDF Bulletin de la Direction des Études et Recherches. Série C. Mathématiques Informatique*, 1:67–90.

GONDRAN, M. et MINOUX, M. (1979). *Graphes et Algorithmes*. Numéro 37 dans Collection de la DER de EDF. Eyrolles, Paris, France.

GUEZGUEZ, S., DULAC, D., et MÉRIGOT, A. (1998). « Parallélisation d'une segmentation par ligne de partage des eaux sur la Maille Associative d'Orsay ». Dans *Rencontres Francophones de l'Intelligence Artificielle RFIA'98*.

GUNAWARDENA, J., éditeur (1998). *Idempotency*. Numéro 11 dans Publications of the Newton Institute. Cambridge University Press, Cambridge, UK.

GUYAUX, P., MONTEIL, T., et GARCIA, J.-M. (1997). Fonctionnalités des bibliothèques de communication. Dans BARTH, D., Chassin de KERGOMMEAUX, J., ROCH, J.-L., et ROMAN, J., éditeurs, *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICARE'97)*, Chapitre 4, pages 75–91. Décembre 1997. École CNRS PRC PRS (Capa, Rumeur, Exec).

HAMBRUSCH, S. E. (1996). « Models for Parallel Computation ». Dans *Proceedings of Workshop on Challenge for Parallel Processing. International Conference on Parallel Processing*. (*cf.* `http://www.cs.purdue.edu/people/seh`).

HAZARI, C. et ZEDAN, H. (1987). « A Distributed Algorithm For Distributed Termination ». *Information Processing Letters*, 24:293–297. Mars 1987.

HEYDEMANN, M.-C. et DUCOURTHIAL, B. (1997). « Cayley graphs and interconnection networks ». Rapport Technique 1122, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France. Septembre 1997.

HILLIS, W. D. (1985). *The Connection Machine*. The MIT Press, Cambridge, Ma, USA.

HOARE, C. (1978). « Communicating Sequential Processes ». *Communication of the ACM*, 21(8):666–677. Août 1978.

HOARE, R., DIETZ, H., MATTOX, T., et KIM, S. (1996). « Bitwise Aggregate Networks ». (*cf.* `http://garage.ecn.purdue.edu/ papers/Arch/`).

HOHBERG, W. (1990). « How to find Biconnected Components in Distributed Networks ». *Journal of Parallel and Distributed Computing (JPDC)*, 9:374–386.

HOROWITZ, S. et PAVLIDIS, T. (1976). « Picture segmentation by a tree traversal algorithm ». *Journal of the ACM*, 23(2):368–388.

HOUZET, D. et FATNI, A. (1998). « Modèle à mémoire partagée sur un réseau de PCs ». *Calculateurs Parallèles*, 10(1):85–106. Février 1998.

JAKOBY, A., REISCHUK, R., et SCHINDELHAUER, C. (1995). « The average case complexity of the parallel prefix problem ». Rapport Technique, Technische Hochschule, Institut für Theoretische Informatik, Darmstadt, GE. Mars 1995. (*cf.* `http://www.itheoi.mu-luebeck.de/Publikationen.html`).

JIANG, B. (1993). « I/O- and CPU-optimal Recognition of Strongly Connected Components ». *Information Processing Letters*, 45(3):111–115. Mars 1993.

JOLION, J. (1998). « Cours de Vision Artificielle ». (*cf.* `http://rfv.insa-lyon.fr/~telesun/welcome.html`).

JOLY, P. (1990). *Mise en œuvre de la méthode des éléments finis*, volume 2. Ellipses. Société de Mathématiques Appliquées et Industrielles.

KAHN, G. et PLOTKIN, G. (1993). « Concrete Domains ». *Theoretical Computer Science*, 121(1-2):187–277.

KNUTH, E., D. (1969). *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Addison Wesley Publishing, Reading, Ma, USA.

KOKOSZKO, B. (1996). « Intégration du modèle data-parallèle irrégulier IDOLE dans C++ ». Dans *Actes des Huitièmes Rencontres Francophones du Parallélisme (RenPar 8)*, Bordeaux, France. Mai 1996.

KUTRIB, M., VOLLMAR, R., et WORSCH, T. (1997). « Introduction to the special issue on cellular automata ». *Parallel Computing*, 23:1567–1576.

LADNER, R. et FISCHER, M. (1980). « Parallel Prefix Computation ». *Journal of the ACM*, 27(4):831–838. Octobre 1980.

LAKSHMIVARAHAN, S. et DHALL, K. (1994). *Parallel Computing Using the Prefix Problem*. Oxford University Press, New-York, USA.

LAMPORT, L. et LYNCH, N. (1990). « *Formal Models and Semantics* », volume B de *Handbook of Theoretical Computer Science*, Chapitre Distributed Computing: Models and Methods, pages 1159–1199. Elsevier Science Publishers. J. van Leeuwen, éditeur.

LEE, B. et HURSON, A. (1994). « Dataflow architectures and multithreading ». *IEEE Computer*, pages 27–39. Août 1994.

LEIGHTON, Thomson, F. (1992). *Introduction to Parallel Algorithms and Architectures : Arrays, Trees, Hypercubes*. Morgan-Kaufmann, San Mateo, Ca, USA.

LI, H. et STOUT, Q., éditeurs (1991). *Reconfigurable Massively Parallel Computers*. Prentice Hall, Englewood Cliffs, NJ, USA.

LYNCH, N. (1996). *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, Ca, USA.

MAGGS, B., MATHESON, L., et TARJAN, R. (1995). « Models of parallel computation: a survey and synthesis ». Dans *Proceeding of the 28th Hawaï International Conference on Systems Science (HICSS'95)*, volume 2, pages 61–70. Janvier 1995.

MAKKI, S. et HAVAS, G. (1996). « Distributed Algorithms for Depth-first Search ». *Information Processing Letters*, 60:7–12.

MARQUET, P. (1993). « Langages et expression du parallélisme de données ». *Technique et Science Informatique (TSI)*, 12(6):685–714.

MARQUET, P. (1997). Gestion de l'irrégularité dans HPF. Dans BARTH, D., Chassin de KERGOMMEAUX, J., ROCH, J.-L., et ROMAN, J., éditeurs, *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICARE'97)*, Chapitre 11, pages 231–250. Décembre 1997. École CNRS PRC PRS (Capa, Rumeur, Exec).

MATTERN, F. (1987). « Experience With a New Distributed Termination Detection Algorithm ». Dans VAN LEEUWEN, J., éditeur, *Distributed Algorithm. Proceedings of the Second Internationnal Workshop*, volume 312 de *Lecture Notes on Computer Science (LNCS)*, pages 127–143, Amsterdam, The Netherlands. Juillet 1987. Springer Verlag.

MÉRIGOT, A. (1992). « Associative Nets: A New Parallel Computing Model ». Rapport Technique 92-02, Institut d'Electronique Fondamentale, bât. 220, Université Paris-Sud, 91405 Orsay, France.

MÉRIGOT, A. (1997a). « Associative Nets: A Graph-Based Parallel Computing Model ». *IEEE Transactions on Computer*, 46(5):558–571. Mai 1997.

MÉRIGOT, A. (1997b). « SAM, la bibliothèque de simulation de la Maille Associative. Mode d'emploi ». Institut d'Electronique Fondamentale, bât. 220, Université Paris-Sud, 91405 Orsay, France. Mai 1997.

MERLIN, J. (1994). Data-Parallelism in Fortran 90 and High Performance Fortran. Notes de cours, Dept. of Electronics and Computer Science, University of Southampton, UK. Mai 1994. (*cf.* `http://www.vcpc.univie.ac.at/activities/tutorials/HPF/lectures/html/`).

MIELLOU, J. (1975). « Algorithmes de relaxation chaotique à retards ». *Revue Française d'Automatique, d'Informatique et Recherche Opérationnelle (RAIRO)*, pages 55–82. Avril 1975.

MOHAMMADI, S. (1996). « *Techniques asynchrones pour la réalisation d'une machine massivement parallèle reconfigurable* ». Thèse de Doctorat, Institut d'Electronique Fondamentale, bât. 220, Université Paris-Sud, 91405 Orsay, France. Juin 1996.

MOHAMMADI, S., DULAC, D., et MÉRIGOT, A. (1994). « Techniques asynchrones mises en œuvre pour une machine massivement parallèle: la Maille Associative d'Orsay ». Dans *Technologie et Architectures, Réseaux d'interconnexion – Journées PRC ANM et RD ASM*, pages 47–56. Mars 1994.

MOREAU, L., DE ROURE, D., et FOSTER, I. (1997). « Nexeme: a distributed Scheme based on Nexus ». Dans LENGAUER, C., GRIELL, M., et GORLATCH, S., éditeurs, *Proceedings of Euro-Par'97, Third International Parallel Processing Conference*, volume 1300 de *Lecture Notes on Computer Science (LNCS)*, pages 581–590, Passau, Germany. Août 1997. Springer Verlag.

NAMYST, R. (1997). « *PM2 : un environnement pour une conception portable et une exécution efficace des applications parallèles irrégulières* ». Thèse de Doctorat, LIFL, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France. Janvier 1997.

NAMYST, R. et MÉHAUT, J.-F. (1995a). « *Marcel : Une bibliothèque de processus légers* ». LIFL, Université des Sciences et Technologies de Lille.

NAMYST, R. et MÉHAUT, J.-F. (1995b). « *PM2 (Parallel Multithreaded Machine) : Guide d'utilisation* ». LIFL, Univ. Sciences et Techn. Lille.

NUUTILA, E. et SOISALON-SOININEN, E. (1994). « On Finding the Strongly Connected Components in a Directed Graph ». *Information Processing Letters*, 49:9–14. Janvier 1994.

PAVLIDIS, T. (1982). *Graphics and Image Processing*. Springer Verlag.

PEREZ, C. et NAMYST, R. (1997). « On the compilation of data-parallel languages on a distributed memory multithreaded environment with thread migration ». Rapport Technique 97-20, LIP, Ecole Normale de Lyon, France. Juillet 1997.

PLANET, P., PRIVAT, G., et RENAUDIN, M. (1993). « Asynchronous relaxation of locally-coupled automata networks, with application to parallel VLSI implementation of iterative image prrocessing algorithms ». Dans *Proceedings of the International Conference on Application Specific Array Processors*, pages 156–159. IEEE. (poster session).

PREPARATA, F. et SHAMOS, M. (1985). *Computational geometry*. Texts and monographs in Computer Science. Springer Verlag, New-York, USA.

PRIOL, T. (1997). Conception et réalisation de mémoires virtuelles partagées. Dans BARTH, D., Chassin de KERGOMMEAUX, J., ROCH, J.-L., et ROMAN, J., éditeurs, *Conception et mise en œuvre d'applications parallèles irrégulières de grande taille (ICARE'97)*, Chapitre 16, pages 325–355. Décembre 1997. École CNRS PRC PRS (Capa, Rumeur, Exec).

PRIVAT, G., ROBIN, F., RENAUDIN, M., et EL HASSAN, B. (1995). « A fine-grain asynchronous VLSI cellular array processor architecture ». Dans *Proceedings of the International Symposium on Circuits and Systems*, pages 1041–1044, Seattle, USA. Avril 1995. IEEE.

PRYLLI, L. et TOURANCHEAU, B. (1997). « Protocol design for high performance networking: a Myrinet experience ». Rapport Technique 97-22, LIP, École Normale Supérieure, Lyon, France.

QUADRAT, J.-P. (1998). Min-plus probability calculus. Dans *Algèbre Max-Plus et applications en informatique et automatique*, Support de cours, pages 393–411. INRIA, Rocquencourt, France. Mai 1998. (26$^{\text{ème}}$ École de printemps d'informatique théorique, Ile de Noirmoutier, France).

RAMARAO, K. V. S. et VENKATESAN, S. (1992). « On finding and updating shortest paths distributively ». *Journal of Algorithms*, 13:235–257.

RAYNAL, M. et HÉLARY, J.-M. (1988). *Synchronisation et contrôle des systèmes et des programmes répartis*. Eyrolles, Paris, France.

REIF, J. H. (1985). « Depth First Search is Inherently Sequential ». *Information Processing Letters*, 20:229–234. Juin 1985.

REINHARDT, F. et SOEDER, H. (1997). *Atlas des Mathématiques*. Encyclopédies d'Aujourd'hui, La Pochotèque. Le Livre de Poche. Édition française.

ROBIN, F. (1997). « *Étude d'architectures VLSI numériques parallèles asynchrones pour la mise en œuvre de nouveaux algorithmes d'analyse et rendu d'images* ». Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, Grenoble, France. Octobre 1997.

ROBIN, F., PRIVAT, G., et RENAUDIN, M. (1997). « Asynchronous relaxation of morphological operators: a joint algorithm-architecture perspective ». *International Journal of Pattern Recognition*, 11(7):1085–1094.

ROH, L., NAJJAR, W. A., SHANKAR, B., et WIM BÔHM, A. (1996). « Generation, optimization, and evaluation of multithreaded code ». *Journal of Parallel and Distributed Computing (JPDC)*, 32:188–204.

ROUCAIROL, C. (1996). « STRATAGÈME : une méthodologie de programmation parallèle pour les problèmes non structurés ». Rapport final, PRISM, Université de Versailles, France.

SABOT, G. (1988). *The Paralation Model*. Artificial Intelligences Series. The MIT Press, Cambridge, Ma, USA.

SALEMBIER, P. et PARDÀS, M. (1994). « Hierarchical morphological segmentation for image sequence coding ». *IEEE Transactions on Image Processing*, 3(5):639–651. Septembre 1994.

SALEMBIER, P. et SERRA, J. (1995). « Flat zones filtering, connected operators and filters by reconstruction ». *IEEE Transactions on Image Processing*, 4(8):1153–1160. Août 1995.

SANSONNET, J. (1994). Architecture des Machines Parallèles. Polycopié du cours du DEA Architecture Parallèle, LRI, Université Paris-Sud, Orsay, France. 400 pages.

SEDGEWICK, R. (1983). *Algorithms*. Addison Wesley Publishing, Reading, Ma, USA.

SEGALL, A. (1983). « Distributed Network Protocols ». *IEEE Transactions on Information Theory*, IT 29(1):23–35. Janvier 1983.

SHARMA, M., IYENGAR, S., et MANDYAN, N. (1989). « An efficient distributed depth-first-search algorithm ». *Information Processing Letters*, 32:183–186. Septembre 1989.

SICARD, N. (1998a). « ANET Manuel d'Utilisation ». Rapport Technique 98-02, Institut d'Electronique Fondamentale, bât. 220, Université Paris-Sud, 91405 Orsay, France. Septembre 1998.

SICARD, N. (1998b). « Mise en œuvre des réseaux associatifs ». Rapport de stage, Magistère d'Informatique Appliquée de l'Ile de France, Institut d'Electronique Fondamentale, bât. 220, Université Paris-Sud, 91405 Orsay, France. Septembre 1998.

SIMON, I. (1998). The tropical semiring and the theory of automata. Dans *Algèbre Max-Plus et applications en informatique et automatique*, Support de cours, pages 373–389. INRIA, Rocquencourt, France. Mai 1998. (26$^{\text{ème}}$ École de printemps d'informatique théorique, Ile de Noirmoutier, France).

SNIR, M. (1986). « Depth-size trade-offs for parallel prefix computation ». *Journal of the Algorithms*, 7:185–201.

TAN, R., TEL, G., et VAN LEEUWEN, J. (1986). « Comments on Distributed Termination Detection Algorithm for Distributed Computations ». *Information Processing Letters*, 23:349. Juillet 1986.

TEL, G. (1987). « Directed Network Protocols ». Dans van LEEUWEN, J., éditeur, *Distributed Algorithm. Second International Workshop*, volume 312 de *Lecture Notes on Computer Science (LNCS)*, pages 13–29, Amsterdam, The Netherlands. Juillet 1987. Springer Verlag.

TEL, G. (1991). *Topics in Distributed Algorithms*, volume 1 de *Cambridge International Series on Parallel Computation*. Cambridge University Press, Cambridge, UK.

TEL, G. (1994). *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, UK.

TEL, G. et VAN LEEUWEN, J. (1987). « Comments on A Distributed Algorithm For Distributed Termination ». *Information Processing Letters*, 25:349. Juillet 1987.

Thinking Machine CORPORATION (1990). « *Connection Machine Model CM-2 Technical Summary* ». Thinking Machines Corporation, Cambridge, Ma, USA. Novembre 1990. Version 6.0.

TUCERYAN, M. et JAIN, A. (1990). « Texture Segmentation Using Voronoï Polygons ». *IEEE Transactions on PAMI*, 13:211–216.

UJALDON, M., ZAPATA, E., CHAPMAN, B., et ZIMA, H. (1997). « Vienna-Fortran / HPF extensions for sparse and irregular problems and their compilation ». *IEEE Transactions on Parallel and Distributed Systems*, 8(10):1068–1083. Octobre 1997.

ULLMAN, J. D. (1984). *Computational Aspects of VLSI*. Computer Science Press, Stanford University, USA.

ÜRESIN, A. et DUBOIS, M. (1989). « Sufficient conditions for the convergence of asynchronous iterations ». *Parallel Computing*, 10:83–92.

ÜRESIN, A. et DUBOIS, M. (1990). « Parallel asynchronous algorithms ». *Journal of the ACM*, 37(3):588–606. Juillet 1990.

ÜRESIN, A. et DUBOIS, M. (1996). « Effects of asynchronism on the convergence rate of iterative algorithms ». *Journal of Parallel and Distributed Computing (JPDC)*, 34:66–81.

VERBIÈSE, L., DEVESA, N., et LECOUFFE, P. (1998). « ACADA : Aide à la Conception et à l'exécution d'Applications Distribuées Adaptatives ». Dans MÉRY, D. et PERRIN, G.-R., éditeurs, *Actes des Dixièmes Rencontres Francophones du Parallélisme (RenPar 10)*, pages 99–102, Starsbourg, France. Juin 1998.

XUONG, N. H. (1992). *Mathématiques discrètes et informatique*. Masson, Paris, France.

ZHU, Y. et CHEUNG, T. Y. (1987). « A new distributed bread-first-search algorithm ». *Information Processing Letters*, 25:329–333.

ZIELONKA, W. (1987). « Notes on finite asynchronous automata ». *RAIRO Informatique Théorique et Applications*, 21(2):99–135.

Index

Conventions de notation

Symboles

$\sqcup$	union disjointe
$\cup$	union ensembliste, ou concaténation de listes
$\equiv$	équivalence, égalité d'applications
$\vee$	ou logique
$\wedge$	et logique
$\diamond$	opérateur binaire
$\Diamond$	opérateur n-aire
$\triangleleft$	r-opérateur binaire
∇	r-opérateur n-aire
$\oplus$	s-opérateur
$e_\diamond$	élément neutre (à droite) de l'opérateur $\diamond$
$\preccurlyeq_\diamond$	relation d'ordre partielle «plus petit que» induite par l'opérateur $\diamond$
$\succcurlyeq_\diamond$	relation d'ordre partielle «plus grand que» induite par l'opérateur $\diamond$

Opérateurs particuliers

plus	addition
mod	modulo
or	ou logique
xor	ou exclusif
and	et logique
$\rho(\mathbf{A})$	rayon spectral de la matrice $\mathbf{A}$
$\mathrm{Ker}(\mathsf{f})$	noyau de l'homomorphisme f
Id	application identité

Ensembles

$\{x_1, x_2, x_3\}$	ensemble contenant les éléments x_1, x_2 et x_3 (ordre quelconque)
(x_1, x_2, x_3)	liste contenant l'élément x_1, puis x_2, puis x_3
$\mathbb{S}$	ensemble de définition des opérateurs
$\mathbb{H}$	ensemble des endomorphismes de $\mathbb{S}$
$\mathbb{M}$	moduloïde
$\mathbb{E}$	Banach
$\mathbb{K}$	espace vectoriel
$\mathbb{N}$; $\overline{\mathbb{N}}$	entiers naturels ; $\mathbb{N} \cup \{+\infty\}$
$\mathbb{Z}$; $\overline{\mathbb{Z}}$	entiers relatifs ; $\mathbb{Z} \cup \{+\infty\}$
$\mathbb{R}_{\max}$	demi-corps max-plus
$\mathbb{Z}_{\min}$	demi-corps tropicale
Σ ; Σ^*	alphabet ; ensemble des mots sur Σ

Systèmes distribués

$\mathcal{C}$	ensemble des configurations
c_i ; c_1 ; c_t	configuration ; configuration initiale ; configuration terminale
e	exécution
$\mathcal{E}_{c_1}$	ensemble des exécutions commençant par c_1
$R\vert\,\Diamond$	règle paramétrée par l'opérateur $\Diamond$
$\{garde\} \longrightarrow \{action\}$	action gardée

Graphes

G, ou $G(V,E)$	graphe, ensemble de sommets V, d'arcs E
G^{-1}	graphe inverse (inversion des arcs)
$\widetilde{G}$	graphe réduit (concentration des composantes fortement connexes)
$\mathrm{Diam}\,(G)$	diamètre
$\mathrm{d}_G(u,v)$	distance de u à v dans le graphe G
δ_v^-	degré entrant du sommet v
$\delta^+(v)$	degré sortant du sommet v
δ_G^-	degré entrant du graphe G
δ_G^+	degré sortant du graphe G
$\mathbf{i}_v(u)$	indice de l'arc en provenance de u sur le sommet v
$\mathbf{i}_v^{-1}(i)$	sommet à l'origine de l'arc entrant numéro i du sommet v
$\Gamma_G^-(v)$	antécédents de v dans G
$\overline{\Gamma_G^-}(v)$	antécédents de v dans G, v inclus
$\Gamma_G^{-1}(v)$	antécédents directs de v dans G
$\overline{\Gamma_G^{-1}}(v)$	antécédents directs de v dans G, v inclus
$\Gamma_G^+(v)$	descendants de v dans G
$\overline{\Gamma_G^+}(v)$	descendants de v dans G, v inclus
$\Gamma_G^{+1}(v)$	descendants directs de v dans G
$\overline{\Gamma_G^{+1}}(v)$	descendants directs de v dans G, v inclus
C	circuit

Réseaux associatifs

G_b	graphe de base
$b_{\text{in}}^v[i]$	tampon d'entrée numéro i du nœud v
$b_{\text{out}}[v]$	tampon de sortie du nœud v
pvar	variable parallèle
pvar$\langle$ **masque** $\rangle$	pvar de masques
net$_G$	pvar de masques définissant le sous-réseau G
net$_G^{-1}$, *net*$_{G^{-1}}$	pvar de masques définissant le sous-réseau G^{-1}
`association`	primitive de la première définition du modèle (calculs sur $\overline{\Gamma_G^-}(v)$)
`direct-association`	primitive de la nouvelle définition (calculs répétés sur $\overline{\Gamma_G^{-1}}(v)$)
`prefix-`	indique que le graphe est préfixe
`step-`	indique qu'un seul calcul local est exécuté par chaque nœud
`exclusive-`	indique que la valeur localle est exclue du calcul (calculs sur $\Gamma_G^-(v)$ ou $\Gamma_G^{-1}(v)$)

Conventions typographiques

$\mathbb{E}$	ensemble
$\mathbf{A}$	matrice
B	vecteurs
F, f, r	fonction
mod	opérateur prédéfini

Table des matières