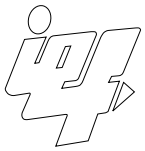

**INSTITUT D'ELECTRONIQUE FONDAMENTALE**
**Université Paris Sud**

# Image analysis with r-operators

Bertrand Ducourthial et Alain Mérigot
ducourth@ief.u-psud.fr am@ief.u-psud.fr

Rapport Interne IEF 98-03

**Abstract**

Associative nets is a well suited parallel computing model for image analysis. It allows to perform asynchronous computations an irregular data. Some efficient implementations have been designed, either in hardware or software.

Asynchronism can largely improve the algorithm speeds, provided these algorithms can be written with operators that allow the asynchronous implementation. Classical operators, such as min, max, $And$, or, $\cup$, $\cap$, **lcm**, **gcd**, ... , are known for their ability to stabilize asynchronous computations. We defined a larger class of operators, called the $r$-operators, that still insure the termination of the asynchronous computations.

After presenting our model, the interest of the asynchronism and the $r$-operators, we investigate in this paper the applications of the $r$-operators in image analysis. We show that they are useful for distance transformation, contour closing and image segmentation.

# Chapter 1

# Introduction

## 1.1 Image analysis

Several properties characterize image processing. First, an image represents a large amount of data, and depending on the applications, it must be processed under strict speed constraints. Second, the elementary data is very small (a pixel); in most cases, it only requires a single byte to be properly coded. Third, at least in the early processing stages, a rigid local geographical relationship exists between these data, which are arranged according to a 2D mesh.

According to these characteristics, classical parallel architectures, that are mainly tuned towards the flexibility and the computing power, required by numerical analysis techniques, are not perfectly suited for image processing analysis, and a scheme based on a large number of small processing elements could probably be more efficient.

Indeed, according to these considerations, there have been several parallel computer architectures proposed for image processing. Thanks to the geographic local organization of the pixels in an image as a two dimensional mesh, and to the regularity of most low-level computations, mesh-based parallel architectures are quite popular for image analysis applications [12, 16]. Indeed they allow efficient implementations of basic neighbor-based primitives (convolution, filtering, etc.).

However, the situation is more complex when dealing with higher level image analysis algorithms. Their objective is mainly to find structural information in an image, to ease higher level recognition phases. In that case, most information is rather given by the complete inter-pixel relationship, and not only by one pixel neighborhood. For instance, interesting information elements may be segmented objects, contour segments, etc. While this kind of information needs more time to be used, it is required for many applications to exploit relational informations beyond the neighborhood of the nodes. The main problem lies in the irregular and data dependent nature of this kind of information, which cannot be efficiently exploited either by compilers or by most parallel architectures.

## 1.2 Associative nets

To overcome these problems, we have proposed the associative net model [15], a graph based computing model well designed for image analysis [8]. Its main purpose is to

provide simultaneously an expressive high level programming model and a potentially efficient hardware implementation.

This model can efficiently exploit structured information within an image, by representing it by means of a graph that interconnects subsets of adjacent pixels. It works with two kinds of basic objects: parallel variables for the pixel-value information, and spanning subgraphs of the mesh for the relational information. These spanning subgraphs are locally coded within every pixel by defining the subset of the eight potential neighbors that are connected to the pixel and that can be manipulated by means of data parallel primitives. These graphs are exploited by computing primitives by means of the *direct-associations* (see Section 1.3 and 2).

The associative nets model is quite simple in its definition. It allows expressiveness in software [3] and powerful implementations in hardware. Hence, the *associative mesh* is an SIMD multicomputer based on asynchronous communications, which has been designed for the associative nets model [10, 15]. In the following sections, we point out two main characteristics of this model.

## 1.3 Asynchronous computing primitive

A *direct-association* consists in several local-to-the-node computations, until no data changes in the network. A local computation is performed on each node $v$ by *associating* the data of the direct ancestors of $v$ using a given operator. Each node then builds a new data, which will be used by the direct descendant nodes for their own local computations, and so on, until stability of the whole network is reached.

During a global computation, the local-to-the-node computations are not synchronized and thus a direct-association is an asynchronous computation over the network of computing nodes. The asynchronism has many advantages, that we present in detail in Chapter 3, after the model have been more precisely defined. In brief, one can say that asynchronism decreases the overhead of control due to synchronization and thus improves the performance of algorithms.

The asynchronism also has some drawbacks. Its main inconvenient is that not all the computations can be performed asynchronously; some of them would never stabilize. However, based on this consideration, we developed a superset of operators known for their ability to stabilize the asynchronous computations: the *r-operators* [7, 6]. In this paper, we investigate their applications in image processing.

## 1.4 Dealing with irregularity

The associative nets model is a superset of the mesh based scheme as direct-associations allow to efficiently implement any kind of neighbor based computations. If the subgraph is regular, it is possible to perform for instance an image shift, but the local manipulation of the communication graph gives an extra flexibility.

Indeed, the main purpose of the model is to provide the ability to perform global computations over any connected sets of the subgraph. For instance, if the considered graph has connected sets that represent regions of an image, it is possible to compute a global measure of the regions, whatever their shapes, and to get informations like area,

average grey level, bounding box and so on by considering appropriate operators and data. Global operations are performed on any spanning subgraphs of the mesh, including the whole mesh itself, graphs with regular connected sets (e.g. chess-boards), graphs with massive connected sets (that may represent regions) or in the shape of wires (for contours).

## 1.5 Aims

The first operators that were used in the associative nets were the associative, commutative and idempotent binary operators such as max, min, and, or, and so on (called below $s$-operators, see Chapter 4). There are a lot of programming applications of $\diamond$-direct-associations where $\diamond$ is an $s$-operator, especially in image analysis [15, 8]. Associative nets have also been used with any kind of operators on directed acyclic graphs, and especially with oriented trees. Since there is no cycle in these digraphs, all operators can be used, even if they are not idempotent (see Chapter 4). For example it is possible to perform a plus-direct-association on integers. This was used, in conjunction with a previous tree embedding, to authorize the use of non-idempotent operators on any graph.

In [7, 6], we proposed to extend the use of associative nets to a superset of the $s$-operators: idempotent $r$-operators (binary, $n$-ary and generalized). These operators have interesting properties that insure convergence of direct-associations on any graph. In this paper, we explore the use of binary and $n$-ary $r$-operators in image analysis and we show that they allow to deal with information coded by the underlying relational graph in images.

In Chapter 2, we present in detail our computational model. We discuss about asynchronism in Chapter 3. In Chapter 4, we recall basic results about the $s$-operators and their use in image analysis. Next, in Chapter 5, we present the $r$-operators. Finally, in the last chapters, some applications are developed: distance computation (Chapters 6, 6.4), contour (Chapter 7) and image segmentation (Chapter 8).

# Chapter 2

# Computing Model

## 2.1 Topology

The associative nets model relies on a network of computing nodes. Any topology is available. For image analysis purpose, the two-dimensional mesh is efficient and we consider such a topology in the following, while a hexagonal grid could be considered too without any formal modification.

The $N \times N$ *mesh* is a symmetric digraph[1] $G(V, E)$ of $|V| = N^2$ nodes, in the form of a two dimensional toroidal grid, with diagonal connection. Each node $v_{i,j}$ in $V$ has eight neighbors $v_{i+\varepsilon \bmod N, j+\varepsilon' \bmod N}$ with $\varepsilon, \varepsilon' \in \{-1, 0, 1\}$ (eight-connectivity). Such a topology has many advantages. It is a Cayley digraph [13] so that it is fully symmetric, and it allows to emulate any other topology [9] embedded in it.

A *node* has a private memory to store some local variables. In particular, each node has a buffer $b_0$ to store the *current local variable*, eight incoming buffers $b_i$, $1 \leq i \leq 8$, to store the data from the neighbors, and a buffer $b_{\text{out}}$ to store the *current result*.

## 2.2 Pvar and mgraphs

A $|V|$-uplet composed of one local variable on each node of the mesh is a *parallel variable* or *pvar*. An $N \times N$ image is a pvar; one pixel is stored on each node.

A special variable of eight bits, called the *mask*, codes on each node a subset of the neighbors that must be taken into account for the local computation: the current result of the $i^{\text{th}}$ neighbor will be copied into $b_i$, before the local computation, only if the $i^{\text{th}}$ bit of the mask is equal to 1. The pvar composed of all the masks is called the *current mgraph*. Such a pvar allows to design any spanning subgraph $H(V, E_H)$ with $E_H \subset E$.

Mgraphs can be used to code any spanning subgraphs, such as graphs that represent regions in the image or contours of objects, or any other topology, such as spanning tree. Mgraphs allow to dynamically deal with irregularity of the images. Any digraph can be embedded into the mesh using a couple of mgraphs (one for the edges, one for the nodes) [9].

[1] A directed graph (or digraph) $G(V, E)$ is defined with a set of vertices (or nodes) $V$ and a set of edges $E$ that interconnect the nodes. A symmetric digraph is a directed graph where for each edge $(u, v)$, there exists the edge $(v, u)$.

## 2.3 Local computation

A *local computation* is performed by a node in associating, with a given operator $\diamond$, its current local variable stored into $b_0$ and the data of the incoming buffers (which can be either identity of $\diamond$ or current result of the neighbors, depending of the current mask).

More formally, a local computation on $v$ based on the binary operator $\diamond$ is defined by the concurrent execution of the two following rules:

1. **if** mask$[i] = 0$, $b_i(v) \leftarrow e_\diamond$, where $e_\diamond$ is the right identity element of the operator $\diamond$.
   **else** $b_i(v) \leftarrow b_{\text{out}}(u)$, where $u$ is the $i^{\text{th}}$ neighbor of $v \quad (1 \leq i \leq 8)$.

2. $b_{\text{out}}(v) \leftarrow b_0(v) \diamond b_1(v) \diamond \cdots \diamond b_8(v)$

If the operator is an $n$-ary one denoted by $\diamondsuit$, the calculation (**2**) is replaced by

$$b_{\text{out}}(v) \leftarrow \diamondsuit\,(b_0(v), b_1(v), \ldots, b_8(v))$$

## 2.4 Global computation

The nodes are not synchronized. A *global computation* consists in concurrent execution of the local computations until stability of the entire network (*i.e.*, all the buffers $b_{\text{out}}$ will no longer change). Such a global computation is called a direct-association because the data of the direct ancestors of each node $v$ in the digraph coded by the current mgraph are associated to build the new result of $v$.

Programs in the associative nets paradigm are composed of synchronous steps. A step is either a purely local computation (without communication, *element-wise*), or a direct-association, which is an asynchronous global computation, including both communications and local computations. A parallel multithreaded programming library called `ANET` has been designed for the Associative `NET`s.

# Chapter 3

# Asynchronous computations

Mgraphs allow to deal with any irregular and dynamic topology. The direct-associations allow to asynchronously apply some operators on connected sets coded by the digraphs. In this chapter, we discuss about the use of the asynchronism in the computations.

## 3.1 Computing on the regions

For computing a result on a region of the image, such as the maximum of the grey levels, a first method consists in scanning all the pixels of the region, and then performing a max-scan [4], provided, for example, that the pixels belonging to the same region are correctly reorganized.

A second method is based on synchronous iterations of local-to-the-node computations of the maximum value of the neighborhood. If the diameter of the region is $d$, such synchronous computations must be performed $d$ times to be sure that the maximum value has been propagated on the complete region. This method is applied on synchronous array-based multi-computers.

A third method, which do not seem totally explored in our opinion, consists to apply the local-to-the-node computation, described above, but without any synchronization. This idea is one of the main characteristics of the associative nets, and it is implemented by means of the direct-association computing primitive. The use of the asynchronism for such a computation has many advantages and we would like to point out some of them in the following sections.

## 3.2 Control

For an asynchronous computation, the data-dependent constraints are relaxed, and the control of the computation is low. Only the termination detection must be managed, instead of synchronization barriers between the synchronous computations (implemented by means of hardware or software techniques). Depending on the implementation, a significant improvement can be obtained, due to the diminution of control.

## 3.3 Communication

An asynchronous communication requires the average communication delay, while a synchronous communication is based on a period, that is computed on the base of the worst communication delay. Hence significant improvements can be obtained either for a pure communication or for a mixed computation-communication like the direct-association.

## 3.4 Electronic implementation

Some electronic implementations can take benefit of the asynchronism [10, 18, 17]. Asynchronism leads to faster communications, but also lower power consumptions (clock picks are avoided). The *Associative Mesh* is an electronic implementation of the associative nets, that has been studied at the IEF [11, 10]. It is an SIMD multiprocessor that implements asynchronous communication. It is based on the mesh topology.

## 3.5 Related works on asynchronism

Several works concern the asynchronous computations for matrix-like calculations. In [2], the author define the asynchronous iterations and shows that they give the same result as the synchronous one for the so-called contractant operators of $\mathbb{R}^n$. These results have been extended in [21] for operators of $S^n$, where $S$ is either a finite or infinite set.

Several experimentations or simulations have shown the interest of decreasing the control for convergence speed, communication delay and synchronization overhead [22].

In the image analysis field, one can find similar studies in [19, 18], especially for the purpose of VLSI implementation. The authors show that some low level operators can be implemented either asynchronously or synchronously. All these studies lead to the conclusion that algorithms, and especially iterative loops, needed to successively improve a temporary solution (see Chapter 6), are speeder when based on asynchronism, either provided by an electronic support or by a parallel environment.

# Chapter 4

# $s$-operators

Main drawback of asynchronous computations is that not all the calculations can be performed this way. Indeed, some synchronous computations can not be replaced by some asynchronous one, depending on the operators. In particular, a necessary condition is the idempotency of the operator [7]. Indeed, a plus-direct-association can not terminate on a ring, except if all the nodes of the ring have been initialized to 0, the identity element of plus.

We define an $s$-operator $\oplus$ as an associative, commutative and idempotent operator on a set $A$ of data containing a neutral element $e_\oplus$ for $\oplus$. These operators — including max, min, or, and, **gcd**, **lcm**, $\cup$, $\cap$, ... — allow to replace a synchronous computation by an asynchronous one.

**Theorem 1** *A $\oplus$-direct-association on a digraph $G(V, E)$, with $\oplus$ an $s$-operator, gives on each node $v$ the result*[1]

$$\oplus\{b_0^0(u), u \in \Gamma_G^-(v)\}$$

The $s$-operators have many applications. For instance, to broadcast a data from a node $v$ over the whole network (supposing it is strongly connected), it is sufficient to set to zero all data except in $v$ and then to perform a max-direct-association.

While they perform interesting computations on regions of the image, one can remark that the $s$-operators give only one result per strongly connected sets. For instance, a max-direct-association(`mg`, `I`) on the image `I` and on the mgraph `mg` (that defines the regions), gives, for each node $v$, the maximal grey level of the pixels of the region to which $v$ belongs.

Some applications however need to extract more information from the inter-pixel digraph than the strongly connected sets. It is the case for instance for distance computations [5], or Voronoï diagrams [1] (see Chapters 6.4 and 8).

In [7, 6], we developed new operators, called $r$-operators, that generalize the $s$-operators and still insure the termination of the asynchronous computations. Moreover these operators allow to better make use of the underlying inter-pixel digraph.

In this paper, we investigate the applications of these operators in image analysis. We first give the definition and some properties of these operators.

[1] $\Gamma_G^-(v)$ denotes the set of all the ancestors of $v$ in the digraph $G$. $b_0^0(u)$ denotes the initial data of the node $v$, which was in the buffer $b_0(u)$ at date 0. $\oplus\{x_1, \ldots, x_n\}$ stands for $x_1 \oplus \cdots \oplus x_n$.

# Chapter 5

# $r$-operators

In this chapter we shortly present the $r$-operators. Additional informations and proofs can be found in [7, 6]. In the rest of this paper, we admit the left-bracketing $(x \diamond y \diamond z = (x \diamond y) \diamond z)$ hypothesis. Moreover, all the operators are defined on the set $A$ and we suppose that they have a right neutral element to set into the masked incoming buffer (see Chapter 2). Finally, we suppose that they admit the (left) faithful property defined by

$$\forall y, z \in A, \quad (\forall x \in A, \quad x \diamond y = x \diamond z) \quad \Leftrightarrow \quad (y = z)$$

The following definition of the $r$-operators should be compared to the one of the $s$-operators.

**Definition 2 (binary $r$-operator)** *The binary operator $\triangleleft$ is an $r$-operator on the set of data $A$ if there is a bijective mapping $r$ from $A$ to $A$, called an* $r$-mapping*, such that $\triangleleft$ verifies the following properties for all $x$, $y$ and $z$ in $A$:*

$$\begin{array}{lcll} (x \triangleleft y) \triangleleft r(z) & = & x \triangleleft (y \triangleleft z) & r\text{-}associativity \\ r(x) \triangleleft y & = & r(y) \triangleleft x & r\text{-}commutativity \\ r(x) \triangleleft x & = & r(x) & r\text{-}idempotency \end{array}$$

As we can see, the $r$-operators generalize the $s$-operators since an $s$-operator is an $r$-operator with the $r$-mapping identity: $\mathsf{Id} : x \mapsto x$. We now lists some properties of the $r$-operators that we will use in this paper.

For any $r$-operator, $\triangleleft$, there is an associated $s$-operator $\oplus$ defined by $x \oplus y = x \triangleleft r^{-1}(y)$. The right-neutral element of $\triangleleft$ is the neutral element of $\oplus$. The $r$-mapping is an isomorphism of $(A, \triangleleft)$ and $(A, \oplus)$. There are as many $r$-operators on $A$ as there are couples of $s$-operators $\oplus$ and isomorphisms $r$ of $(A, \oplus)$. When the $r$-operator $\triangleleft$ verifies $x \triangleleft x = x$, it is said idempotent. To obtain an idempotent $r$-operator $\triangleleft$ from an $s$-operator $\oplus$ and an isomorphism $r$ of $(A, \oplus)$, it is sufficient that $\oplus$ and $r$ verify, for all $x$ in $A$, $x \preceq_{\oplus} r(x)$, where $\preceq_{\oplus}$ is the partial order relation induced by the $s$-operator $\oplus$, defined by $x \preceq_{\oplus} y \equiv x \oplus y = x$ (in the term of the $s$-operator, the $s$ stands for *sort*).

We showed in [6] that a $\triangleleft$-direct-association based on any idempotent $r$-operator $\triangleleft$ terminates. This result is based on the fact that any infinite expression built with a finite number of variable and an idempotent $r$-operator can be reduced into a finite expression (in the term $r$-operator, the $r$ stands for *reduce*). Moreover an $r$-operator defines a partial order relation $\preceq_{\oplus}$, through its associated $s$-operator.

**Theorem 3** *A $\triangleleft$-direct-association on a digraph $G(V,E)$, with $\triangleleft$ an idempotent r-operator based on the s-operator $\oplus$ and the r-mapping r, gives on each node v the result*[1]

$$\oplus\{r^{\mathrm{d}_G(u,v)}\left(b_0^0(u)\right), u \in \Gamma_G^-(v)\}$$

As we can see, a direct-association based on an $r$-operator builds a result that includes the distance related information of the underlying inter-node digraph. This information is more complete than the strongly connected information exploited by the $s$-operators.

Following the construction of the binary $r$-operators from the $s$-operators, one can define the $n$-ary $r$-operators.

**Definition 4 ($n$-ary $r$-operator)** *A mapping $\triangleleft$ from $A^n$ into $A$ is called an* $n$-ary $r$-operator *if there is an s-operator $\oplus$ on $A$ and $n-1$ isomorphisms $r_i$ of $(A,\oplus)$ such that for all $x$ in $A$,*

$$\triangleleft(x_0,\ldots,x_{n-1}) = x_0 \oplus r_1(x_1) \oplus \ldots \oplus r_{n-1}(x_{n-1})$$

In other words, an $n$-ary $r$-operator is built with $n-1$ $r$-operators $\triangleleft_i$ based on the same $s$-operator and the $r$-mappings $r_i$ as follows: $x \triangleleft_i y = x \oplus r_i(y)$. If all these $n-1$ $r$-operators $\triangleleft_i$ are idempotent ($x \triangleleft_i x = x \oplus r_i(x) = x$), we say that the $n$-ary $r$-operator is idempotent.

When each node $v$ of the network uses a 8-ary $r$-operator, then there is one $r$-mapping per edge. One can then associate to each dipath[2] $P$ of the network an *r-dipath-mapping* $r_P$ defined by the combination of the $r$-mappings of each edge of the dipath.

**Theorem 5** *A $\triangleleft$-direct-association on a digraph $G(V,E)$, with $\triangleleft$ an idempotent n-ary r-operator based on the s-operator $\oplus$, gives on each node v the result $\oplus\{r_P\left(b_0^0(u)\right), u \in \Gamma_G^-(v), P$ elementary dipath from u to v$\}$.*

In [6], we defined the *generalized r-operators* that still insure the termination of the direct-associations. They have many applications for computing, especially in image analysis (*e.g.*, mathematical morphology). For short, we limit our presentation of the $r$-operators to the binary and $n$-ary one. We now give in the following chapters some applications of these operators in image analysis.

[1] $\mathrm{d}_G(u,v)$ denotes the distance from $u$ to $v$ in the digraph $G$.

[2] A *dipath* of a directed digraph is a list of consecutive edges with the same orientation.

# Chapter 6

# Single source shortest path (sssp) problem

The single source shortest path (sssp) problem, is the basis of a set of image analysis techniques that we detail in further chapters. We first give some graph related definitions. Next, we present an implementation of the sssp problem by means of iterations, before showing how the use of $r$-operators can largely improve this classical implementation.

## 6.1 Definitions

The *main sources* of a digraph $G$ are the vertices $s$ such that there exists a dipath from $s$ to all other vertices.

A *tree* of the digraph $G(V, E)$ is a digraph $T(V_T, E_T)$ where $V_T \subset V$ and $E_T \subset E$ and such that there is one and only one path[1] between any vertices $u$ and $v$ in $T$. A *rooted tree* $T$ of $G$ is a tree of $G$ admitting a main source $r$, called the *root* of the tree. A *rooted spanning tree* $T(V_T, E_T)$ of $G(V, E)$ is a rooted tree of $G$ where $V_T = V$.

A *shortest dipath rooted spanning tree* of $G$ is a rooted spanning tree of $G$ such that, for any vertex $v$, the distance from the root $s$ to $v$ in the tree is equal to the distance from the root to $v$ in $G$ ($\mathrm{d}_T(s, v) = \mathrm{d}_G(s, v)$). A *shortest dipath rooted spanning forest* $F$ of $G$ is a set of rooted trees $T^i$ of $G$, rooted on some roots $s^i$, such that each vertex $v$ of $G$ belongs to one and only one of these trees. Moreover, if $s_j$ is the root of the tree $T_j$ to which $v$ belongs, $\mathrm{d}_F(s_j, v) = \mathrm{d}_G(s_j, v) \leq \mathrm{d}_F(s_i, v)$ for all roots $s_i$.

A *weighted digraph* $G_w(V, E)$ is defined as a digraph $G(V, E)$ with an application $w$ from $E$ to $\mathbb{N}$, which assigns a weight to each edge. The *weight* of a dipath $P$ of $G_w$, denoted by $w(P)$ is the sum of the weights of all its edges. The *weighted distance* between two vertices $u$ and $v$ in $G_w$, denoted by $\mathrm{d}_{G_w}(u, v)$, is the minimum of the weights of all dipaths from $u$ to $v$. The *weighted shortest dipath rooted spanning tree* of a weighted digraph $G_w$ is defined similarly to the shortest dipath rooted spanning tree in a digraph $G$, except that the weighted distance is used instead of the distance. The problem of finding such a tree in a weighted digraph is commonly called the *single source shortest path problem*, or *sssp* problem. When there are several sources, we can define similarly the *weighted shortest dipath rooted spanning forest*, and define the *multiple source shortest*

[1] A path in a digraph is a list of consecutive edges, which can have different orientations.

*path problem*, or *mssp* problem.

## 6.2 Iterative implementation of the sssp problem

Since the sssp problem implies to know by any way the sum of an attribute (the weights) over some dipaths, it requires the use of an addition. However the addition is not an idempotent operator and it cannot be used for the direct-associations on any network [7] (see Chapter 4). A direct-association converges with any operator provided the considered network has no cycle. Thus we can perform a plus-direct-association on a tree and use the result to determine a shortest dipath rooted spanning tree. An outline of the algorithm is the following (see [15] for details).

First, a random root-to-leaves rooted spanning tree $T_w$ is built on the weighted digraph $G_w$, rooted at the given source $s$ of the problem. Then using a plus-direct-association in $T_w$, the weighted distance $\mathrm{d}_{T_w}(s,v)$ from the source $s$ to every vertex $v$ in $T_w$ is computed (by summing the weights of the edges over the dipaths). Afterwards we compare in every vertex $v$ its weighted distance in $T_w$ to its neighbors and select a neighbor $u$ as the new parent of $v$ in the tree $T_w$ if $\mathrm{d}_{T_w}(s,v) > \mathrm{d}_{T_w}(s,u) + w((u,v))$. We then iterate these computing-updating steps until the tree $T_w$ is stable. The tree is then the shortest dipath rooted spanning tree and the distance $\mathrm{d}_{T_w}$ in $T_w$ is equal to the distance $\mathrm{d}_{G_w}$ in $G_w$. This algorithm can safely be extended to several sources to solve in parallel the shortest dipath forest problem.

## 6.3 Implementing the sssp problem by means of $r$-operators

The use of $r$-operators can largely improve the previous algorithm and avoid the iterative improvement steps. First, we consider the case where the weights are all equal to 1. The problem is then to find a shortest dipath rooted spanning tree in a digraph. Next, we study the general case.

### 6.3.1 In an unweighted digraph

Initially, we set the initial data $b_0^0(v)$ (*i.e.*, the value which is stored into the buffer $b_0$ at date 0) of each vertex $v$ to $+\infty$, except the data of the main source $s$, that we set to 0. The synchronous computation could be performed by the iteration of the following local computation (here for node $v$):

$$b_{\mathrm{out}}(v) = \mathsf{min}(b_0(v), b_1(v)+1, \cdots, b_8(v)+1)$$

The synchronous computation could then be applied iteratively, the number of iterations been at least equal to the largest distance from the main source to a node.

However the binary operator minc (for minimum and increment) defined by $\mathsf{minc}(x,y) = \mathsf{min}(x, y+1)$ give the same local computation:

$$b_{\mathrm{out}}(v) = \mathsf{minc}\,(b_0(v), b_1(v), \cdots, b_8(v))$$

Since $\mathsf{min}$ is an $s$-operator on $\overline{\mathbb{Z}} = \mathbb{Z} \cup \{+\infty\}$, and since the mapping $r : x \mapsto x+1$ is an isomorphism of $(\overline{\mathbb{Z}}, \mathsf{min})$, the operator $\mathsf{minc}$ is an $r$-operator (see Chapter 5). Moreover, this operator is idempotent, as $\mathsf{minc}(x, x) = \mathsf{min}(x, x+1) = x$. Then, according to Theorem 3, the $\mathsf{minc}$-direct-association converges and gives, on each node $v$, the result

$$\mathsf{min}\left\{r^{\mathrm{d}_G(u,v)}\left(b_0^0(u)\right), u \in \Gamma_G^-(v)\right\}$$

. Since the source $s$ is the unique ancestor of $v$ such that its initial data $b_0^0(s)$ is not $+\infty$, the result in $v$ is then $\mathrm{d}_G(s, v)$. To terminate the algorithm, each node must mask all its incoming buffers but one, containing the smallest received value.

The algorithm remains correct to compute the shortest dipath rooted forest, when several sources are initialized to 0 at the beginning.

### 6.3.2 In a weighted digraph

The method is similar for the multiple source shortest dipath problem in a weighted digraph $G_w(V, E)$ where each edge $(u, v)$ has a weight $w((u, v)) \geq 0$. Indeed, in that case, each node $v$ performs the following local computation:

$$b_{\mathrm{out}}(v) = \mathsf{min}\left(b_0(v), b_1(v) + w((u_1, v)), \cdots, b_8(v) + w((u_8, v))\right)$$

where $u_i$ is the $i^{\mathrm{th}}$ direct ancestor of $v$ in $G_w$. Since each mapping $r_v^{u_i} : x \mapsto x + w((u_i, v))$ $(1 \leq i \leq 8,\ v \in V)$ is an isomorphism of $(\overline{\mathbb{Z}}, \mathsf{min})$, the local computation defines an $n$-ary $r$-operator on $\overline{\mathbb{Z}}$, that we call $\mathsf{minc}_w$ (see Definition 4).

Since $x \leq r_v^{u_i}(x)$ $(1 \leq i \leq 8,\ v \in V)$, the $\mathsf{minc}_w$ operator is idempotent. For any dipath $P$ in $G_w$, the $r$-dipath-mapping of $P$ is an application which adds the weight $w(P)$ of the dipath $P$ $(r_P : x \mapsto x + w(P))$. According to Theorem 5, the $\mathsf{minc}_w$-direct-association terminates and gives on each node $v$ the result

$$\mathsf{min}\left\{b_0^0(u) + w(P), u \in \Gamma_G^-(v), P\text{dipath from } u \text{ to } v\right\}$$

. Since all the nodes have been initialized to $+\infty$ except the sources, the result in each node $v$ is the weighted distance from the closest source to $v$.

Finally, each node masks all its incoming buffers but one, containing the smallest received value. This solve the multiple source shortest dipath problem.

### 6.3.3 Conclusion

Hence, the $r$-operators can efficiently replace the iterations needed for solving the sssp and related problems by a single asynchronous computation. We now present some applications for image analysis.

## 6.4 Distance transforms in image analysis

We first define the distance transform before study its implementation by means of $r$-operators.

## 6.5 Definition

Given a set of $k$ points $p_i$, $i$ in $\{0, \ldots, k\}$, in an image, the *distance transform* (DT) consists to label every pixel with its geometrical distance to the closest point $p_i$. This transform is the basis of several useful methods of image analysis: median axis transform, "skeletization", etc. We will use it in Chapter 8 for the Voronoï diagrams.

The considered geometrical distance can be the Euclidean distance, but its large computational cost is useless in most applications and people prefer to use simpler metrics. Popular ones are the *city bloc* metric $d_4$ and the *chess board* metric $d_8$ defined by

$$d_4(p,q) = |x_p - x_q| + |y_p - y_q|$$

and

$$d_8(p,q) = \mathsf{max}\,(|x_p - x_q|, |y_p - y_q|)$$

where $p$ and $q$ are points of an image, $(x_p, y_p)$ and $(x_q, y_q)$ their coordinates.

It is also possible to define weighted approximations that fits more closely the Euclidean distance. The distance is computed as a weighted sum of a purely diagonal path — defined by $\mathsf{min}\,(|x_p - x_q|, |y_p - y_q|)$ — and of a vertical or horizontal path — defined by $\mathsf{max}\,(|x_p - x_q|, |y_p - y_q|) - \mathsf{min}\,(|x_p - x_q|, |y_p - y_q|)$. The distance from pixel $p$ to pixel $q$ is then:

$$\begin{aligned} d(P,Q) = \; & a \times \mathsf{min}\,(|x_p - x_q|, |y_p - y_q|) + \\ & b \times (\mathsf{max}\,(|x_p - x_q|, |y_p - y_q|) - \mathsf{min}\,(|x_p - x_q|, |y_p - y_q|)) \end{aligned}$$

For instance, the values $(a,b) = (3,2)$ or $(a,b) = (7,5)$ for diagonal and non-diagonal links improve the preciseness of the distance approximation [5].

## 6.6 Implementation by means of $r$-operators

The distance computation in the mesh can be seen as a sssp problem. Indeed, the city block metric $d_4$ computation is similar to the sssp problem in the spanning subgraph of the mesh obtained in masking all the diagonal connections and with all its weights equal to one. For the chess board metric, the considered subgraph is the whole mesh[2], with all the weights equal to one. Hence these metrics can be computed by a single minc-direct-association either applied on the whole mesh or with diagonal connections masked.

The weighted approximations of the Euclidean distance can be seen as a sssp problem in the weighted mesh, where all the diagonal links have the weight $a$ and all other links (horizontal or vertical) have the weight $b$. Hence a single $\mathsf{minc}_w$-direct-association computes the weighted approximations of the Euclidean distance.

[2]For these applications, toroidal connections must be masked.

# Chapter 7

# Contour closing

Contour closing is a classical problem in image analysis. While low-level contour extractors can compute a measure of the intensity of a contour in any point, noise can blur the information, and it is generally required to find the proper extension that will close contours. This can be done by searching the best dipath that connects some contour extremities.

The classical way to do that is by means of a single source shortest path search. After contour extraction, points with a contour intensity above a threshold are marked as certain, and the other ones are labelled with a quantity inversely proportional to their contour intensity. The problem is to find the sequence of points whose global sum is minimal. This is the sssp problem on the inter-pixel digraph where the weight of an edge $(u, v)$ is equal to the vertex contour measure of $v$.

It is also possible to improve the method by considering contour orientation; the weight of the edge from vertex $u$ to vertex $v$ will be reduced if contour orientation in vertex $u$ is directed towards $v$ and conversely.

These methods for contours closing can simply be solved by the above sssp technique using the $r$-operators.

# Chapter 8

# Image segmentation

Image segmentation is a computing intensive task in image analysis. Its purpose is to gather in the same region similar pixels, with respect to some homogeneity criterion. For instance, one could try to form regions whose variance is below a certain threshold, or whose difference between maximum and minimum value is bounded.

To implement that task, a classical method is to use the so-called split-and-merge algorithm [14]. First, the image is recursively split into regions, while the homogeneity criterion is not satisfied everywhere. Then, these regions are merged until no merging can be done that satisfies the homogeneity criterion.

The associative mesh model is perfectly adapted to this scheme, as it allows to perform efficiently the global region manipulations, by considering a graph that interconnects pixels of the same region [8]. Associations on this graph can implement useful computations as, for instance, the ones required by the homogeneity. Moreover the graph manipulations are sufficient to modify the regions.

Several schemes can be used for splitting the image. A classical one is to define a recursive geometrical decomposition *à la quad-tree*. This method is quite straightforward to implement, as the graph decomposition of the image can only rely on the pixels coordinates. A smarter way [20] is to use a decomposition of the region based on the Voronoï polygons.

In that case, to split inhomogeneous regions of the image, one needs to choose within that region a point, either randomly or by using some heuristics, that will be the *seed* of a new Voronoï region. The *Voronoï region* associated each seed $p_i$ in the image is the set $S_i$ of pixels whose distance to $p_i$ is smaller than the distance to any other point $p_j$, $i \neq j$ [1].

Computing the Voronoï regions can be seen as the shortest dipath rooting spanning forest in an unweighted digraph, where the roots of the trees are the seeds $p_i$ (see Chapter 6.1). Thus, as shown in Chapter 6.3, a single $\mathsf{minc}_w$-direct-association, followed by a local choice of a single smallest ancestor, gives the Voronoï tessellation.

Hence, the $r$-operators allow to asynchronously compute the Voronoï segmentation of an image. They considerably improve the previous implementation of this algorithm in the associative nets [8], which was based on iterative improvements of the regions. Applications of the $r$-operators to image segmentation is not limited to the Voronoï tessellation. We are currently studying other asynchronous segmentations based on these operators (such as a watershed segmentation).

# Chapter 9

# Conclusion

The associative nets model is well suited for computing on irregular data, and especially in image analysis. Moreover, it allows the replacement of synchronous iteration by asynchronous ones. Such replacements allow significant improvement on the speed of the algorithms, depending on the execution support. Electronic implementation (Associative Mesh) as well as parallel multithreaded programming library (`ANET`) have been designed for this purpose.

However, not all the computations can be replaced by asynchronous one (*e.g.*, additions are forbidden), and some basic important algorithms were only performed on an acyclic underlying graph by means of iterative loops, as, for instance, for the Voronoï segmentation.

Thanks to new operators that enlarge the class of well known associative, commutative and idempotent operators ($s$-operators), many algorithms are improved. In this article, we investigated the applications of these new operators, called $r$-operators, for distance transformation, contour closing and image segmentation by means of Voronoï tessellation.

These applications illustrate the interest of the $r$-operators for image analysis. But the $r$-operators are not limited to the algorithms detailed here. The authors are currently studying other applications in asynchronous image segmentation and mathematical morphology.

# Bibliography


[1] Ahuja, N., An, B., and Shachte, B. Image representation using voronoi tesselation. *CVGIP 29* (1985), 286–295.

[2] Baudet, G. M. Asynchronous iterative methods for multiprocessors. *Journal of the ACM 25*, 2 (April 1978), 226–244.

[3] Biancardi, A., and Mérigot, A. Connected component support for image analysis programs. *Proc. 13th ICPR, IEEE Press D* (1996), 620–624.

[4] Blelloch, G. *Vector Models for Data-Parallel Computing.* Artificial Intelligences Series. The MIT Press, Cambridge, Ma, USA, 1990.

[5] Borgefors, G. Distance transformations in digital images. *Computer vision, graphics, and image processing 34* (1986), 344–371.

[6] Ducourthial, B. Generalization of the $r$-operators. Tech. Rep. 1181, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France, June 1998.

[7] Ducourthial, B. New operators for computing with *associative nets.* In *Proceedings of the Fifth International Colloquium on Structural Information and Communication Complexity (SIROCCO'98), Amalfi, Italia* (June 1998), Carleton Scientific, p. (to appear).

[8] Ducourthial, B., Constantinescu, G., and Mérigot, A. Implementing image analysis with a graph based parallel computing model. *Computing Supplement 12* (1998), 111–121.

[9] Ducourthial, B., and Mérigot, A. Graph embedding in the associative mesh model. Tech. Rep. 96-02, Institut d'Electronique Fondamentale, Bât. 220. Université Paris-Sud, 91405 Orsay, France, December 1996. *(submitted to IEEE Transaction on Computer).*

[10] Dulac, D. *Contribution au Parallélisme Massif en Analyse d'Images : Une Architecture SIMD Fondée sur la Reconfigurabilité et l'Asynchronisme.* PhD thesis, Institut d'Electronique Fondamentale, bât. 220, Université Paris-Sud, 91405 Orsay, France, January 1996.

[11] Dulac, D., Mohammadi, S., and Mérigot, A. Implementation and evaluation of a parallel architecture using asynchronous communications. In *Proceedings of*

*the 1995 Workshop on Computer Architecture for Machine Perception (CAMP'95)* (Pavia, Italia, September 1995), V. Cantoni, L. Lombardi, et al., Eds., IEEE Press, pp. 106–111.

[12] FOUNTAIN, T. J. *Processors Arrays: Architecture and Applications.* Academic Press, 1987.

[13] HEYDEMANN, M.-C., AND DUCOURTHIAL, B. Cayley graphs and interconnection networks. Tech. Rep. 1122, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France, Septembre 1997.

[14] HOROWITZ, S., AND PAVLIDIS, T. Picture segmentation by a tree traversal algorithm. *Journal of the ACM 23*, 2 (1976), 368–388.

[15] MÉRIGOT, A. Associative nets: A graph-based parallel computing net. *IEEE Transactions on Computer 46*, 5 (May 1997), 558–571.

[16] OWENS, R. M., IRWIN, M. J., NAGENDRA, C., AND BAJWA, R. S. Computer vision on MGAP. In *Proceedings of the 1993 Workshop on Computer Architecture for Machine Perception* (New Orleans, USA, December 1993), L. S. Davis, M. A. Bayoumi, and K. P. Valavanis, Eds., pp. 337–341.

[17] PLANET, P., PRIVAT, G., AND RENAUDIN, M. Asynchronous relaxation of locally-coupled automata networks, with application to parallel VLSI implementation of iterative image prrocessing algorithms. In *Proceedings of the International Conference on Application Specific Array Processors* (1993), IEEE, pp. 156–159. (poster session).

[18] PRIVAT, G., ROBIN, F., RENAUDIN, M., AND EL HASSAN, B. A fine-grain asynchronous VLSI cellular array processor architecture. In *Proceedings of the International Symposium on Circuits and Systems* (Seattle, USA, April 1995), IEEE, pp. 1041–1044.

[19] ROBIN, F. *Etude d'architectures VLSI numériques parallèles asynchrones pour la mise en œuvre de nouveaux algorithmes d'analyse et rendu d'images.* PhD thesis, Ecole Nationale Supérieure des Télécommunications, Grenoble, France, Octobre 1997.

[20] TUCERYAN, M., AND JAIN, A. Texture segmentation using Voronoi polygons. *IEEE Transactions on PAMI 13* (1990), 211–216.

[21] ÜRESIN, A., AND DUBOIS, M. Sufficient conditions for the convergence of asynchronous iterations. *Parallel Computing 10* (1989), 83–92.

[22] ÜRESIN, A., AND DUBOIS, M. Effects of asynchronism on the convergence rate of iterative algorithms. *Journal of Parallel and Distributed Computing (JPDC) 34* (1996), 66–81.