Programmation Par Contraintes

Cours 3 - Contraintes globales, Look-back

David Savourey

CNRS, École Polytechnique

inspiré de la thèse d'Hadrien Cambazard et du livre Constraint Processing (R. Dechter)

Sommaire

Contraintes globales

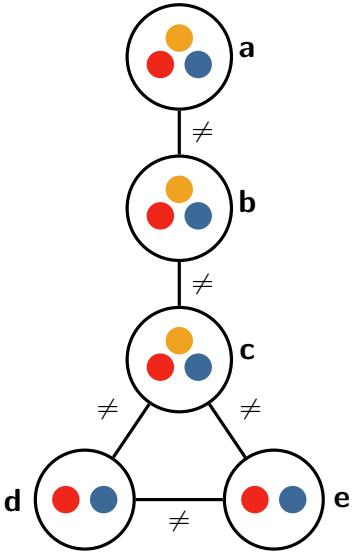
Qu'est-ce qu'une contrainte globale ?

- contrainte n-aire très structurée
- version binarisée très structurée aussi
- contrainte qui revient souvent
- intérêt : algos de propagation de consistance spécifiques et plus efficaces

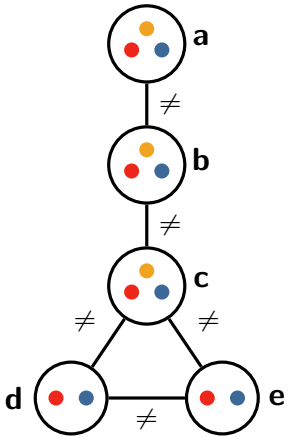
- un sous-ensemble de variables doivent prendre des valeurs 2 à 2 différentes
- s'écrit en binaire simpliment avec $n(n-1)/2$ contraintes de différence
- consistance $=$ recherche d'un couplage maximal

- *sum*
- *knapsack*
- *cumulative*
- *element*
- souvent spécifique à un type de problèmes
- catalogue : `https://sofdem.github.io/gccat/`

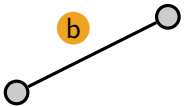
Apprentissage : backtrack intelligent

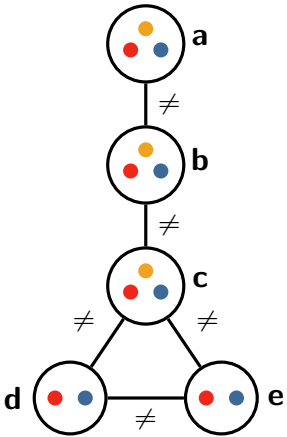

a
≠
b
≠
c
≠
≠
d
≠
e

Supposons qu'on branche dans l'ordre b, a, c, d, e.

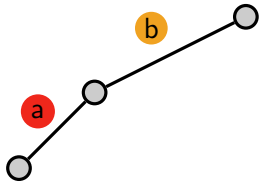


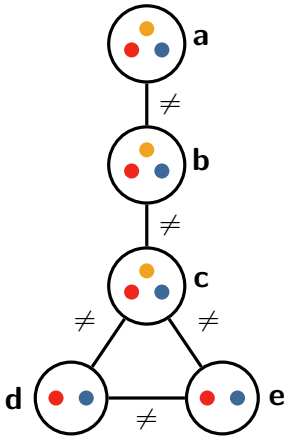
Supposons qu'on branche dans l'ordre b, a, c, d, e.



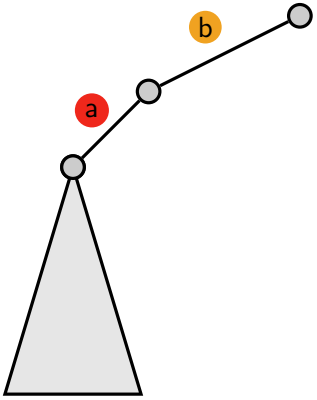


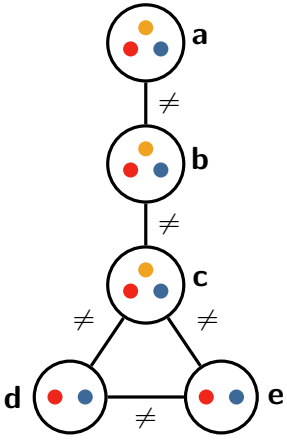
Supposons qu'on branche dans l'ordre b, a, c, d, e.



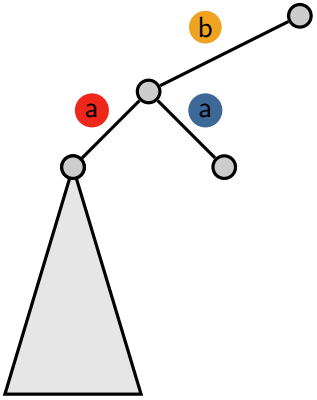


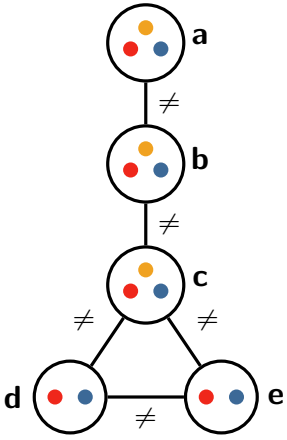
Supposons qu'on branche dans l'ordre b, a, c, d, e.



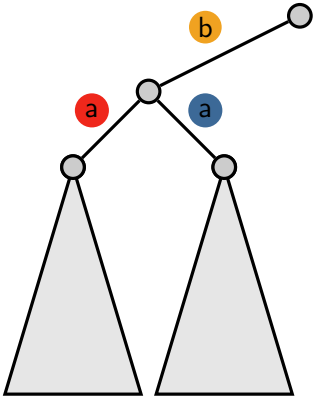


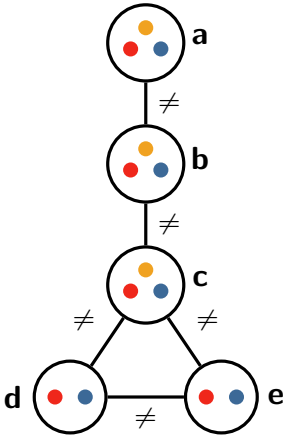
Supposons qu'on branche dans l'ordre b, a, c, d, e.



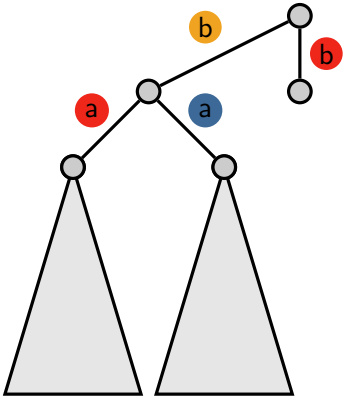


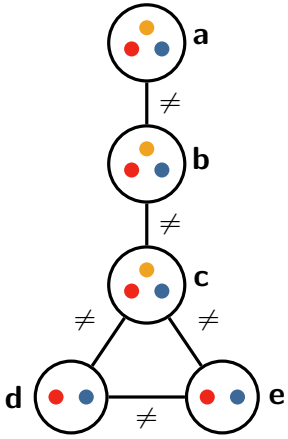
Supposons qu'on branche dans l'ordre b, a, c, d, e.



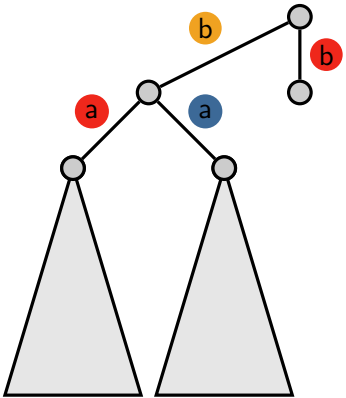


Supposons qu'on branche dans l'ordre b, a, c, d, e.

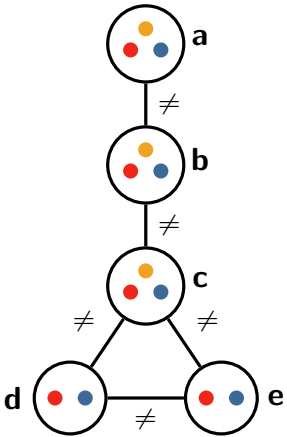




Supposons qu'on branche dans l'ordre b, a, c, d, e.

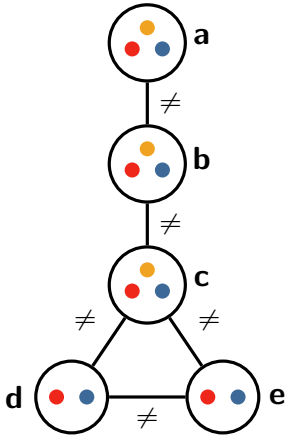


le contenu de ces 2 sous-arbres est identique



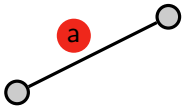
Un deuxième exemple de thrashing

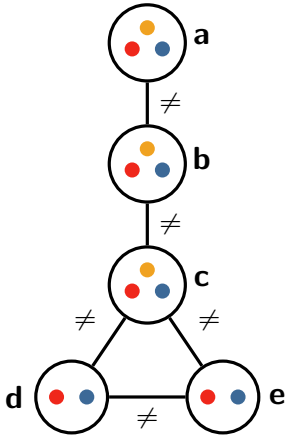
Supposons qu'on branche dans l'ordre a, b, c, d, e.



Un deuxième exemple de thrashing

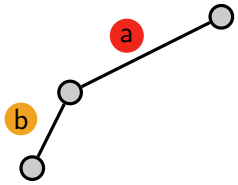
Supposons qu'on branche dans l'ordre a, b, c, d, e.

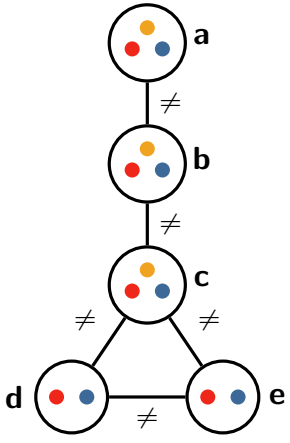




Un deuxième exemple de thrashing

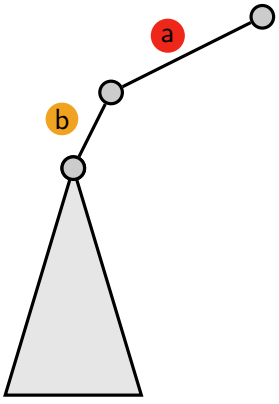
Supposons qu'on branche dans l'ordre a, b, c, d, e.

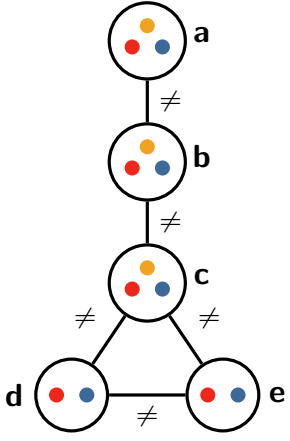




Un deuxième exemple de thrashing

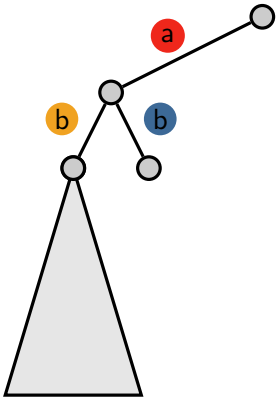
Supposons qu'on branche dans l'ordre a, b, c, d, e.

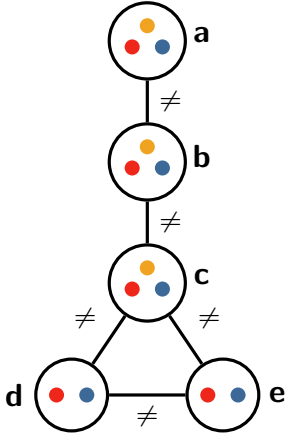




Un deuxième exemple de thrashing

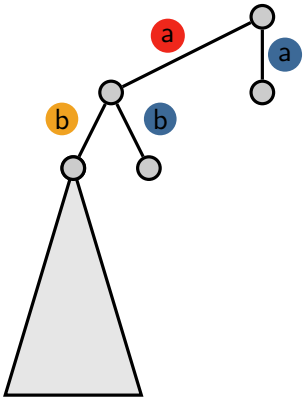
Supposons qu'on branche dans l'ordre a, b, c, d, e.

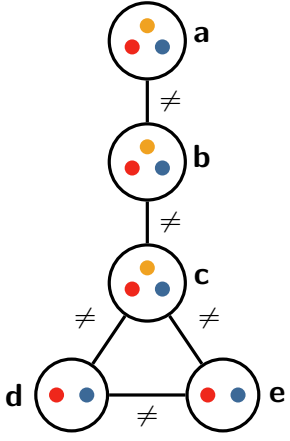




Un deuxième exemple de thrashing

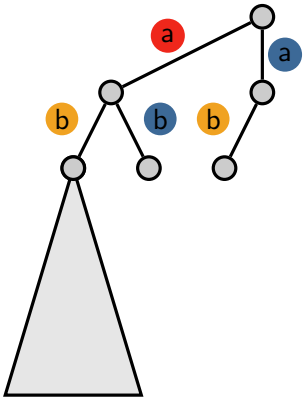
Supposons qu'on branche dans l'ordre a, b, c, d, e.

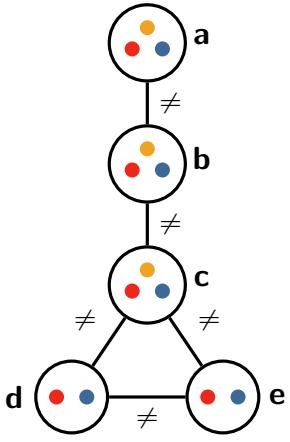




Un deuxième exemple de thrashing

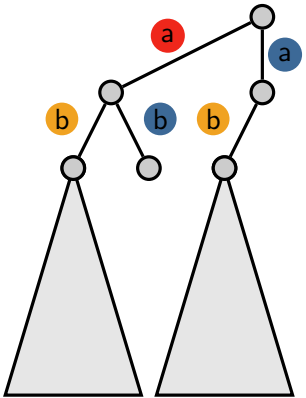
Supposons qu'on branche dans l'ordre a, b, c, d, e.

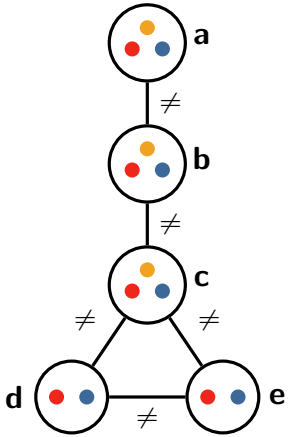




Un deuxième exemple de thrashing

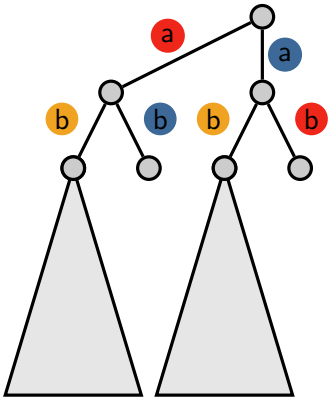
Supposons qu'on branche dans l'ordre a, b, c, d, e.

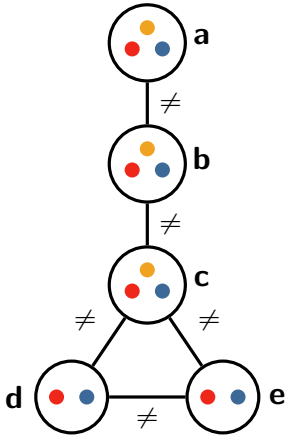




Un deuxième exemple de thrashing

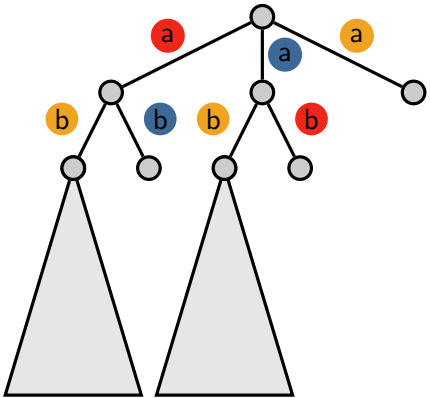
Supposons qu'on branche dans l'ordre a, b, c, d, e.

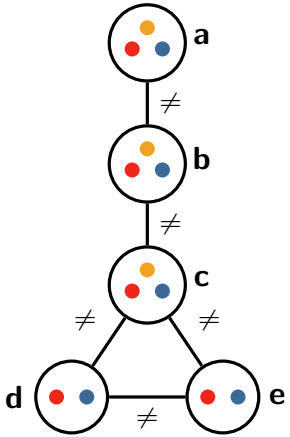




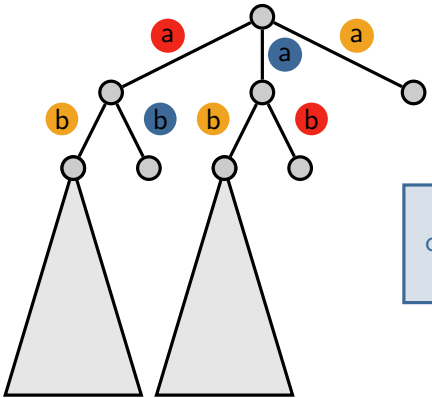
Un deuxième exemple de thrashing

Supposons qu'on branche dans l'ordre a, b, c, d, e.

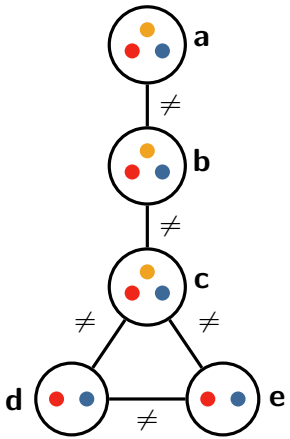




Supposons qu'on branche dans l'ordre a, b, c, d, e.



le contenu de
ces 2 sous-arbres
est identique



Comment limiter le thrashing ?

- un même outil : les explications
- deux méthodes :
  - premier exemple : backjumping
  - deuxième exemple : learning

- Soit C l'ensemble des contraintes originelles du problème
- Soit DC l'ensemble des contraintes de décisions
- L'explication e du retrait de la valeur a du domaine de la variable x est la donnée de 2 sous-ensembles $C_e \subseteq C$ et $DC_e \subseteq DC$ tels que $C_e \wedge DC_e \models x \neq a$.
- On notera $expl(x \neq a) = C_e \cup DC_e$ une telle explication.
- Une explication de contradiction est l'explication de l'absence de valeurs pour une variable x ($D_x = \varnothing$), de sorte que :

$$expl(D_x = \varnothing) = \bigcup_{v \in D_x} expl(x \neq v)$$

Algorithme : Xplain

Données : C, m

$X \leftarrow \emptyset$;

$X_{tmp} \leftarrow \emptyset$;

tant que $m(X) \neq \bot$ **faire**

  $k \leftarrow 0$;

  tant que $m(X_{tmp}) \neq \bot$ **et** $k < n$ **faire**

    $k \leftarrow k + 1$;

    $X_{tmp} \leftarrow X_{tmp} \cup \{C_k\}$;

  si $m(X_{tmp}) \neq \bot$ **alors** Retourner $\emptyset$;

  $X \leftarrow X \cup \{C_k\}$;

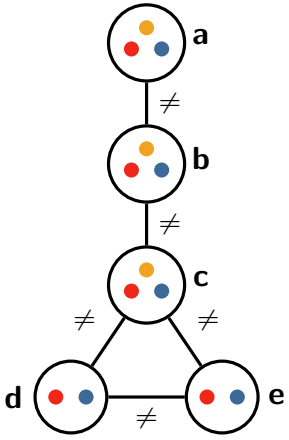
  $X_{tmp} \leftarrow X$;

Retourner X ;

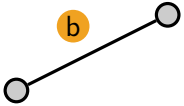
C : ensemble de contraintes C_1, C_2, C_n menant à une contradiction, m une technique de résolution (AC par exemple).

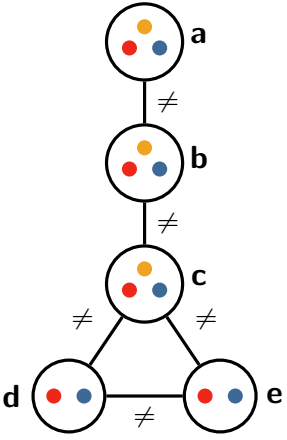
- L'algorithme Xplain calcule une explication minimale au sens de l'inclusion
- il est coûteux en temps !
- il existe des versions plus efficaces (QuickXplain)

Premier exemple : backjumping

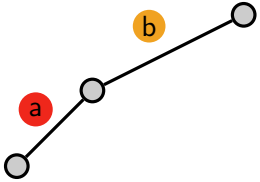


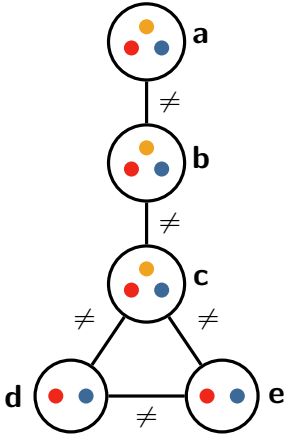
Premier exemple : backjumping



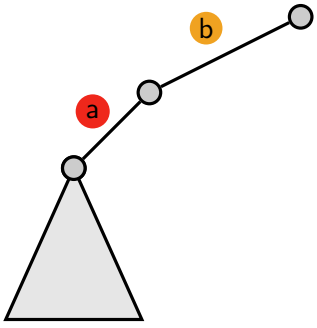


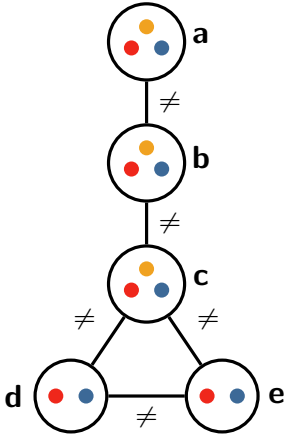
Premier exemple : backjumping

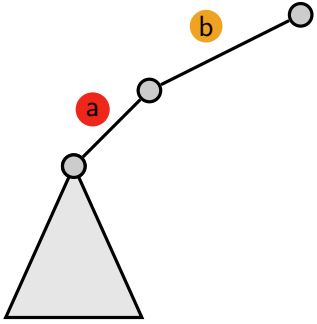




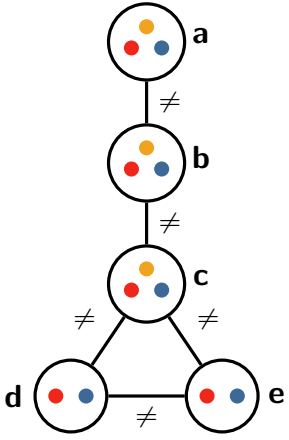
Premier exemple : backjumping

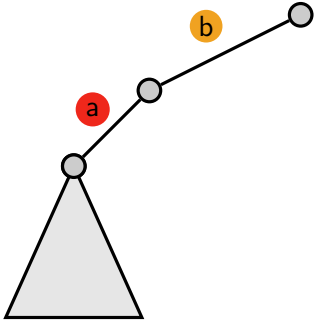






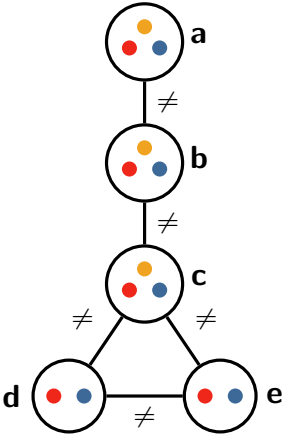
$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

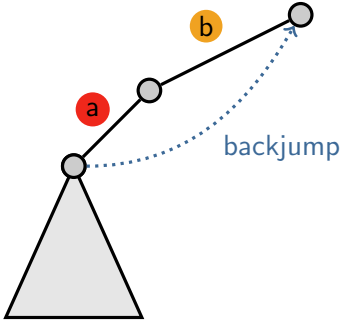




$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

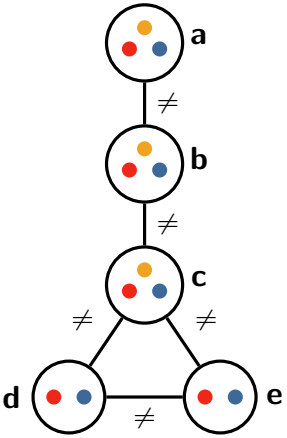
ne dépend pas de a

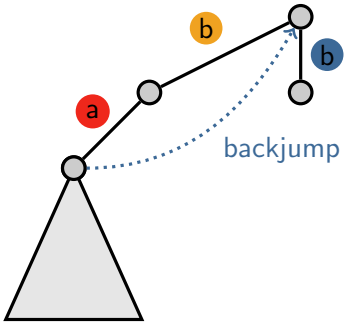




$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

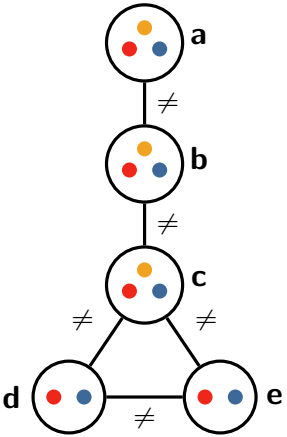
ne dépend pas de a

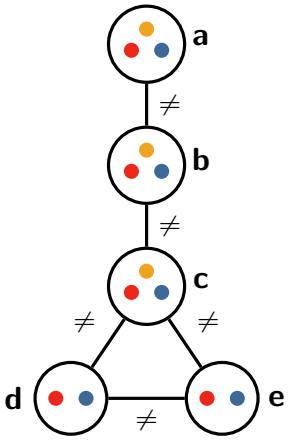


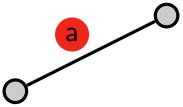


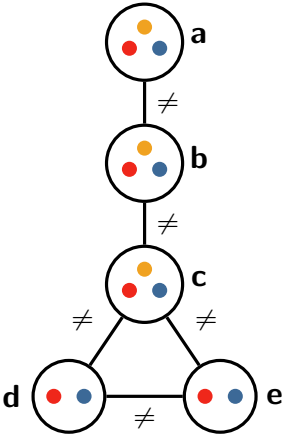
$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

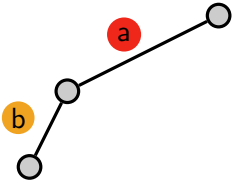
ne dépend pas de a

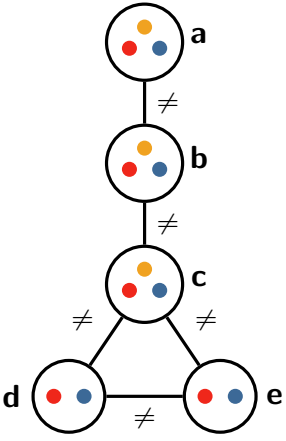


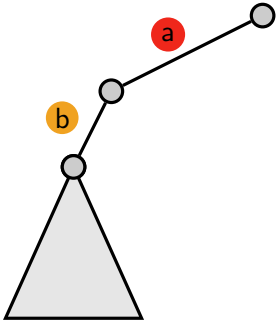

a
≠
b
≠
c
≠
≠
d
e
≠

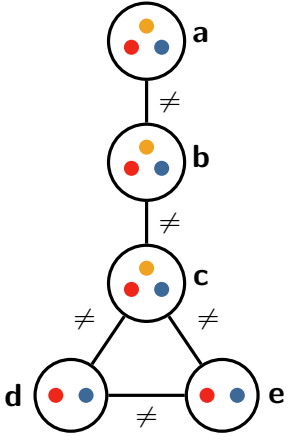

a

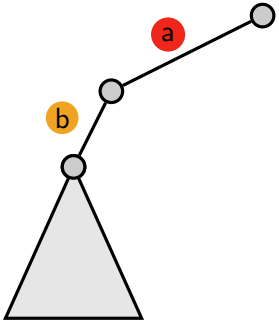

a
≠
b
≠
c
≠
≠
d
e
≠


a
b

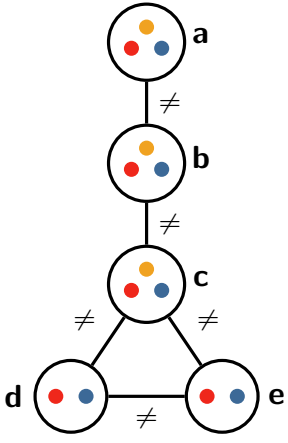

a
≠
b
≠
c
≠
≠
d
e
≠

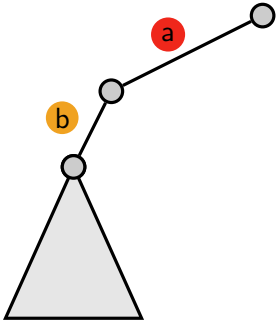

a
b


a
≠
b
≠
c
≠
≠
d
e
≠



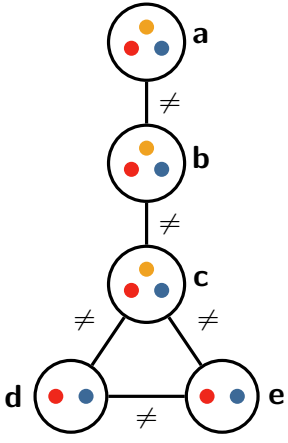
$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

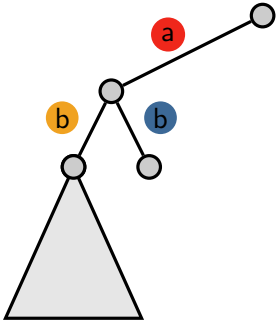




$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

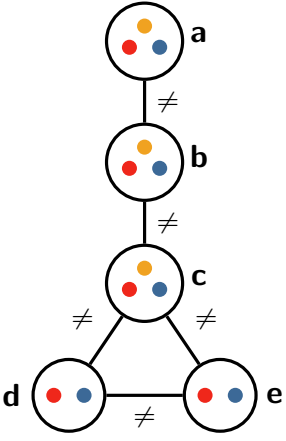
No goods : $\{b\ jaune\}$

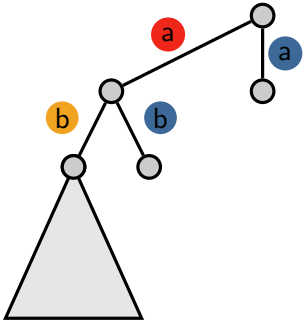




$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

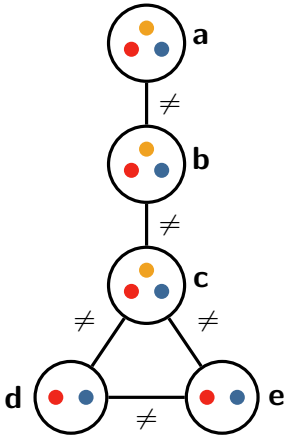
No goods : $\{b\ jaune\}$

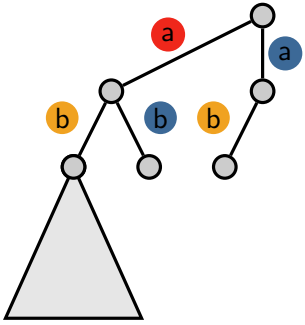




$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

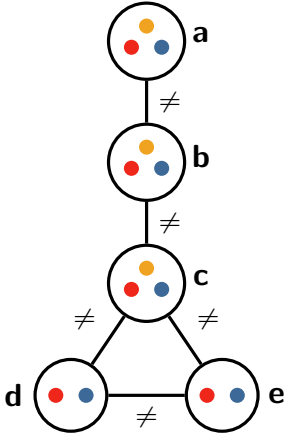
No goods : $\{b\ jaune\}$

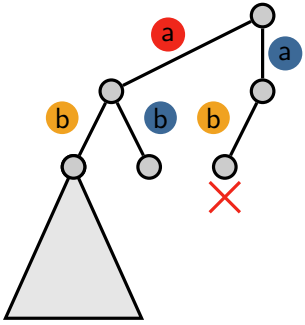




$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

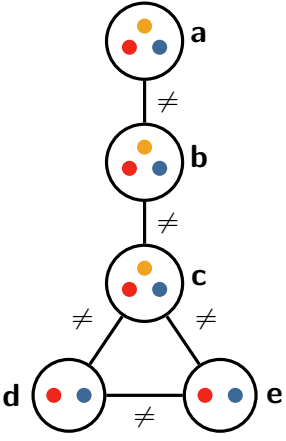
No goods : $\{b\ jaune\}$

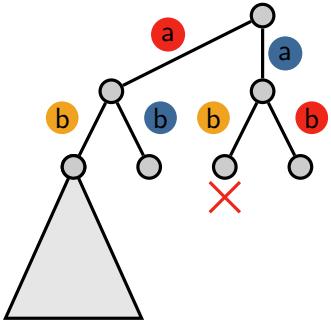




$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

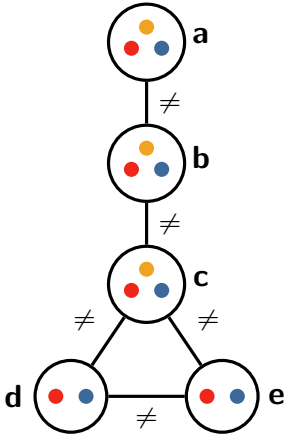
No goods : $\{b\ jaune\}$

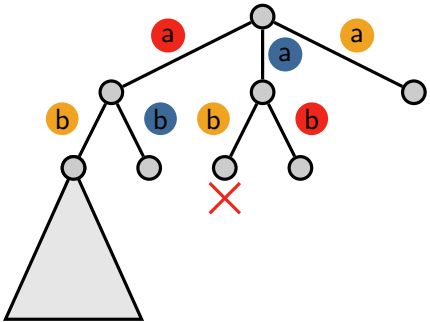




$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

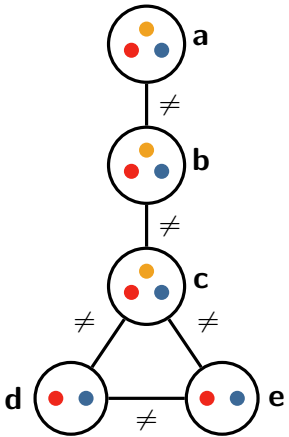
No goods : $\{b\ jaune\}$





$expl(D_c = \emptyset) = \{b\ jaune, b \neq c, c \neq d, d \neq e, c \neq e\}$

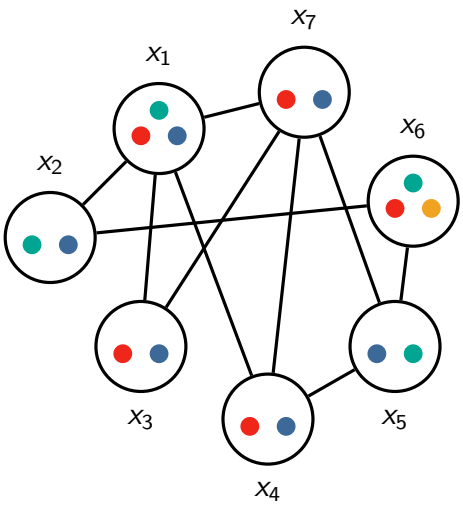
No goods : $\{b\ jaune\}$



Bilan sur les explications

- explications : cadre unificateur de l'apprentissage des erreurs
- existe aussi en version online
- versions présentées sont simplistes !
- différents types de backjumping
- idem pour l'apprentissage

Techniques de backjumping



toutes les contraintes = différence

Définition

Soit $\vec{a} = (a_{i_1}, \ldots, a_{i_k})$ une instanciation consistante d'un sous-ensemble quelconque de variables, et soit x une variable non instanciée. Si aucune valeur $b \in dom(x)$ ne vérifie que $(\vec{a}, x = b)$ est consistant, on dit que $\vec{a}$ est un **ensemble en conflit** avec x.

Définition

Si de plus, $\vec{a}$ ne contient aucun sous-ensemble en conflit avec x, on dit que $\vec{a}$ est un **ensemble en conflit minimal** avec x.

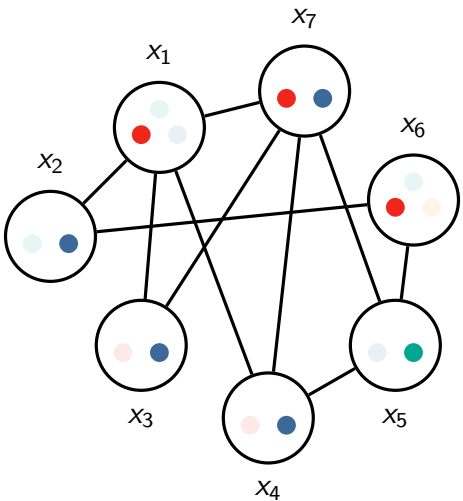
Définition

Soit un CSP (X, D, C). Toute instantiation partielle $\vec{a}$ qui n'est contenue dans aucune solution est appelée un **no-good**. Un no-good est dit minimal s'il ne contient aucun no-good.

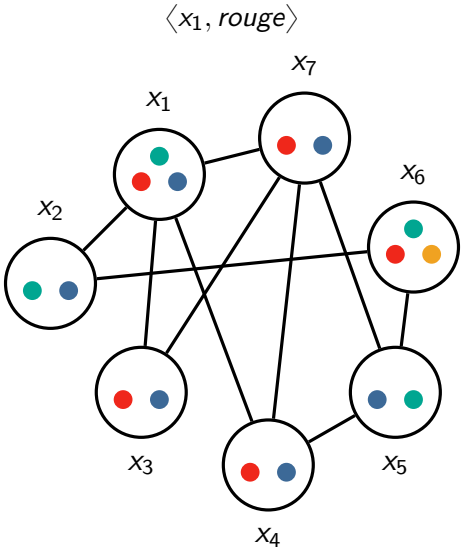
Remarque

Un ensemble en conflit est donc un no-good.

$\{\langle x_1, rouge\rangle, \langle x_2, bleu\rangle, \langle x_3, bleu\rangle, \langle x_4, bleu, \rangle, \langle x_5, vert\rangle, \langle x_6, rouge\rangle\}$



Exemple de no-good qui n'est pas un conflict-set



On se place désormais dans le cadre d'une recherche arborescente et on suppose, sans perte de généralités, que l'ordre sur les variables est $(x_1, x_2, \ldots, x_n)$.

Définition

Soit $\vec{a_i} = (a_1, \ldots, a_i)$ un tuple. Si $\vec{a_i}$ est en conflit avec x_{i+1}, on dit que $\vec{a_i}$ est une **feuille sans issue** et que x_{i+1} est une **variable sans issue**.

Définition

Soit $\vec{a_i} = (a_1, \ldots, a_i)$ une feuille sans issue. On dira que x_j, $j \leqslant i$, est un **saut valide** (ou sûr, *safe* en anglais) relativement à $\vec{a_i}$ si l'instanciation $\vec{a_j} = (a_1, \ldots, a_j)$ est un no-good.

Si l'on change la valeur a_j de la variable x_j dans $\vec{a_i}$, on n'explorera jamais un certain nombre d'instantiations qui commencent par $\vec{a_j}$. Mais dans la mesure où $\vec{a_j}$ est un no-good, on est sûr de ne rater aucune solution.

Définition

Soit $\vec{a_i} = (a_1, \ldots, a_i)$ une feuille sans issue. L'indice coupable (*culprit index*) relativement à $\vec{a_i}$ est $b = \min\{j \leqslant i, \vec{a_j}$ en conflit avec $x_{i+1}\}$. On dira alors que x_b est la variable coupable (*culprit variable*) par rapport à $\vec{a_i}$.

Proposition

Si $\vec{a_i}$ est une feuille sans issue et x_b sa variable coupable, alors $\vec{a_b}$ est le plus grand backjump valide possible depuis $\vec{a_i}$.

Algorithme : Backjumping de Gaschnig

Entrées : un CSP (X, D, C)
$i \leftarrow 1$, $D'_i \leftarrow D_i$, $latest_i \leftarrow 0$
tant que $1 \leqslant i \leqslant n$ **faire**
  $x_i \leftarrow$ Select-Value-GBJ()
  si $x_i = -1$ **alors**
    $i \leftarrow latest_i$
  sinon
    $i \leftarrow i + 1$, $D'_i \leftarrow D_i$, $latest_i \leftarrow 0$
si $i = 0$ **alors**
  Renvoyer Inconsistant
sinon
  Renvoyer $(x_1, \ldots, x_n)$

Algorithme : Select-Value-GBJ

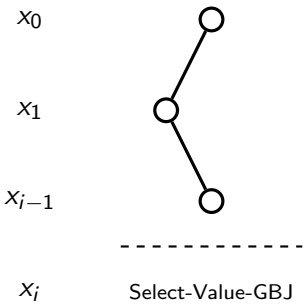
tant que D'_i **non vide faire**
  Choisir $a \in D'_i$, retirer a de D'_i
  $consist \leftarrow true$, $k \leftarrow 1$
  tant que $k < i$ **et** $consist$ **faire**
    si $k > latest_i$ **alors** $latest_i \leftarrow k$;
    si $(\vec{a_k}, x_i = a)$ **n'est pas consistant alors**
      $consist \leftarrow false$
    sinon
      $k \leftarrow k + 1$
  si $consist$ **alors** Renvoyer a;
Renvoyer -1

i	D_i :	a	b	c	d	e
1		✓	✓	✓	✓	✓
2		✓	✓	✓	✓	✓
3		X	✓	✓	X	X
...			✓	✓		
$i-1$			✓	✓		
		$latest_i = 2$	$latest_i = i-1$			
			return b	return c		return -1

Quand Select-Value-GBJ renvoie -1, on a toujours $latest_i = i - 1$, donc pas de backjumping !

i	D_i :	a	b	c	d	e
1		✓	✓	✓	✓	✓
2		✓	✓	X	✓	✓
3		X	✓		✓	X
...			X		X	
		$latest_i = 2$	$latest_i = 3$			
						return -1

Quand Select-Value-GBJ renvoie -1, on a toujours $latest_i = 3$, donc backjumping !



- Si Select-Value-GBJ trouve une valeur consistante dans D_i, alors il aura fixé $latest_i = i - 1$. Donc quand il aura épuisé toutes les valeurs possibles pour x_i, on aura toujours $latest_i = i - 1$ et on fera un backtrack simple.
- Si Select-Value-GBJ ne trouve aucun valeur consistante dans D_i, alors il aura fixé $latest_i$ à la variable k la plus tardive dans l'ordre d'instantiation qui est impliquée dans une inconsistance. Cela va déclencher un backjump en k.

Le backjump ne se produit que depuis des leaf dead end !

- se base sur les valeurs
- implémente les plus grands sauts possibles
- ne sait backjumper que depuis des feuille sans issue

Définition

Lorsqu'un algorithme de parcours effectue un backtrack (ou un backjump) vers une variable x_j depuis une feuille sans issue, et que cette variable x_j n'a pas valeur possible à instantier, on dit que x_j est une **variable interne sans issue** (*internal dead end variable*) et que $a_{j-1}^{\vec{}}$ est un **état interne sans issue**.

Définition

Le graphe des contraintes associé à un CSP est le graphe où les sommets représentent les variables et les arêtes l'existence d'une contrainte entre 2 variables.

Définition

Soit un graphe de contraintes et un ordre sur les sommets d, **l'ensemble des ancêtres** de la variables x, noté $anc(x)$, est le sous-ensemble des variables qui précède x dans d et qui sont reliées à x dans le graphe des contraintes. **Le parent** de x, noté $p(x)$, est le plus récent de ses ancêtres (selon d).

Définition

On dit que le backtrack **invisite** x_i s'il traite x_i en venant d'une variable précédant x_i dans l'ordre d.

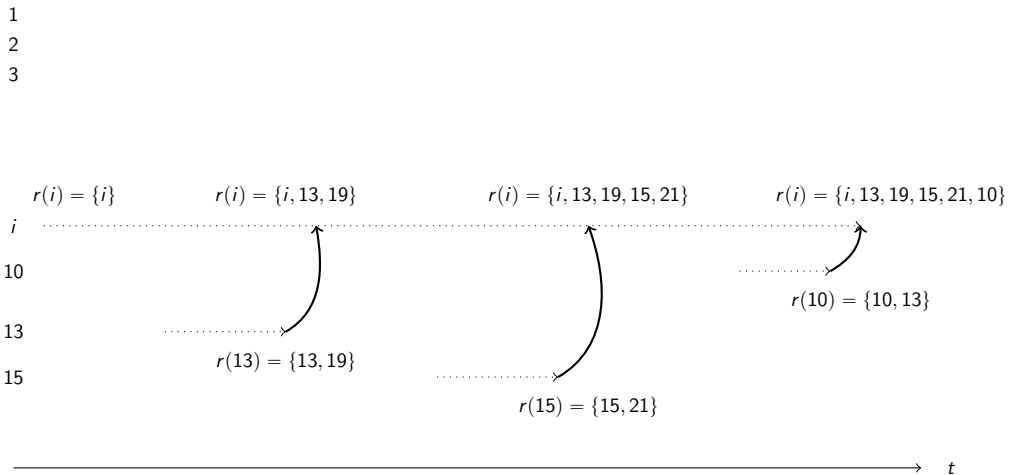
Définition

La **session** de x_i commence lors de son invisite Elle se termine lorsque l'on backtracke sur une variable qui précède x_i dans l'ordre d.

Définition

Les **variables sans issues pertinentes** (*relevant dead end*) de la session x_i, notées $r(x_i)$, se définissent récursivement :

- $r(x_i) = \{x_i\}$ quand on invisite x_i,
- $r(x_i) = r(x_i) \cup r(x_j)$ quand on backjumpe en x_i depuis x_j.


1
2
3
$r(i) = \{i\}$
$r(i) = \{i, 13, 19\}$
$r(i) = \{i, 13, 19, 15, 21\}$
$r(i) = \{i, 13, 19, 15, 21, 10\}$
i
10
13
15
$r(10) = \{10, 13\}$
$r(13) = \{13, 19\}$
$r(15) = \{15, 21\}$
t

Définition

Soit x_i une variable sans issue. Soit Y l'ensemble des variables sans issue pertinentes de la session de x_i. L'ensemble des **ancêtres induits** de x_i par rapport à Y, noté $I_i(Y)$ est l'union de tous les ancêtres des variables de Y, restreint aux variables précédant x_i. Formellement, $I_i(Y) = \bigcup_{y \in Y} anc(y) \cap \{x_1, x_2, \ldots, x_{i-1}\}$.

Définition

Le **parent induit** de x_i par rapport à Y, noté $P_i(Y)$ est la dernière variable de $I_i(Y)$. On appelle aussi $P_i(Y)$ le **coupable basé graphe** de x_i.

Proposition

Soit $a_i\vec{}_{-1}$ une variable sans issue (feuille ou interne), et soit Y l'ensemble des variables sans issue pertinentes de la session courante de x_i. Alors $x_j = P_i(Y)$ est la variable la plus précoce où il est valide de backjumper.

Algorithme : Graph based Backjumping

Entrées : un CSP (X, D, C)

Calculer $anc(x_i)$ pour tous les x_i

$i \leftarrow 1$, $D'_i \leftarrow D_i$, $I_i \leftarrow anc(x_i)$

tant que $1 \leqslant i \leqslant n$ **faire**

- $x_i \leftarrow$ Select-Value()
- **si** $x_i = -1$ **alors**
  - $ip \leftarrow i$, $i \leftarrow \max I_i$, $I_i \leftarrow I_i \cup I_{ip} \backslash \{x_i\}$
- **sinon**
  - $i \leftarrow i + 1$, $D'_i \leftarrow D_i$, $I_i \leftarrow anc(x_i)$

si $i = 0$ **alors**

- Renvoyer Inconsistant

sinon

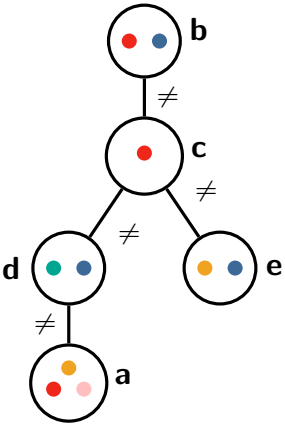
- Renvoyer $(x_1, \ldots, x_n)$

Algorithme : Select-Value

tant que D'_i **non vide faire**

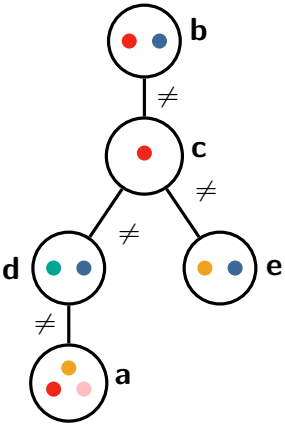
- Choisir $a \in D'_i$, retirer a de D'_i
- **si** $(\vec{a_{i-1}}, x_i = a)$ **consistant alors**
  - Renvoyer a

Renvoyer -1



Si on instancie dans l'ordre b,e,a,d,c :

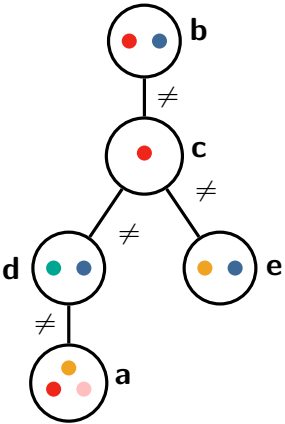
Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge

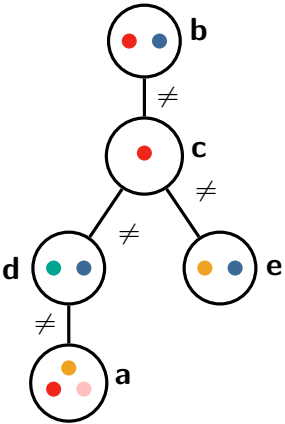
Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge
on fixe e=jaune

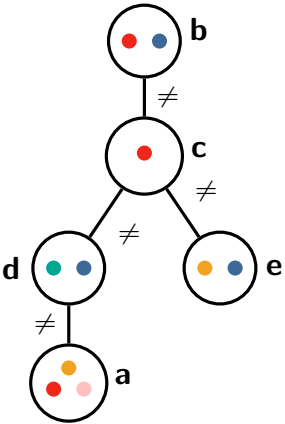
Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge
on fixe e=jaune
on fixe a=jaune

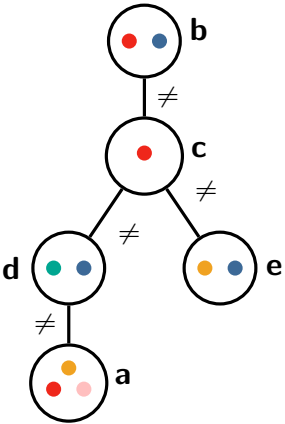
Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge
on fixe e=jaune
on fixe a=jaune
on fixe d=vert

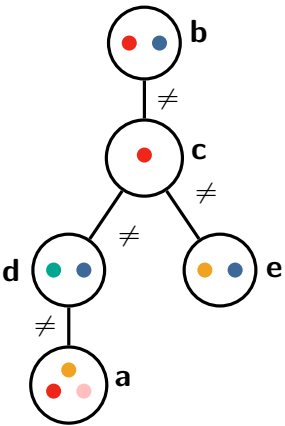
Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge
on fixe e=jaune
on fixe a=jaune
on fixe d=vert
c est une dead end : backjump en d

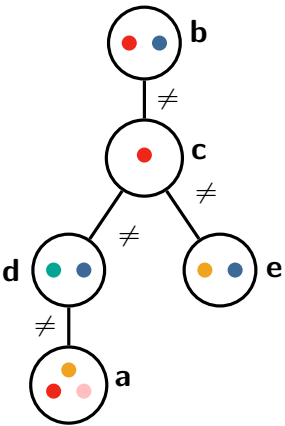
Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge
on fixe e=jaune
on fixe a=jaune
on fixe d=vert
c est une dead end : backjump en d
on fixe d=bleu

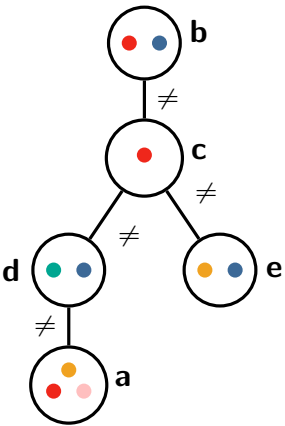
Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge
on fixe e=jaune
on fixe a=jaune
on fixe d=vert
c est une dead end : backjump en d
on fixe d=bleu
c est une dead end : backjump en d

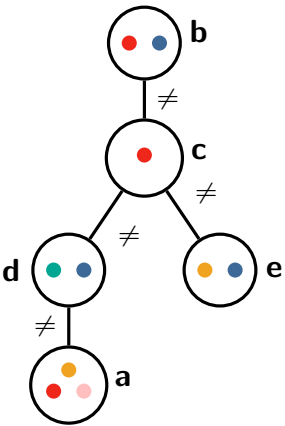
Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

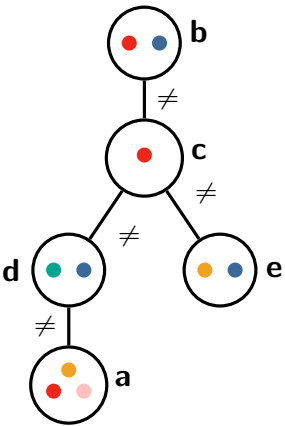
on fixe b=rouge
on fixe e=jaune
on fixe a=jaune
on fixe d=vert
c est une dead end : backjump en d
on fixe d=bleu
c est une dead end : backjump en d
d est une dead end : backjump en a

Limite du graph based backjumping



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge
on fixe e=jaune
on fixe a=jaune
on fixe d=vert
c est une dead end : backjump en d
on fixe d=bleu
c est une dead end : backjump en d
d est une dead end : backjump en a
on fixe a=rouge



Si on instancie dans l'ordre b,e,a,d,c :

on fixe b=rouge
on fixe e=jaune
on fixe a=jaune
on fixe d=vert
c est une dead end : backjump en d
on fixe d=bleu
c est une dead end : backjump en d
d est une dead end : backjump en a
on fixe a=rouge
...

Définition

Soit un ordre sur les variables. La contrainte R précède la contrainte Q si et seulement si la dernière variable de $scope(R) \backslash scope(Q)$ précède la dernière variable de $scope(Q) \backslash scope(R)$.

Définition

Soient d un ordre sur les variables, $\vec{a_i}$ un tuple et x_{i+1} la potentielle variable sans issue qui va avec. Pour chaque valeur b de D_{i+1}, on définit :

$$V_b = \begin{cases} \varnothing \text{ si } (\vec{a_i}, x_{i+1} = b) \text{ est consistant,} \\ scope(C_j) \text{ avec } C_j \text{ plus petite contrainte non satisfaite sinon.} \end{cases}$$

On note alors

$$\text{var-emc}(\vec{a_i}) = \bigcup_{b \in D_{i+1}} (V_b \backslash \{x_{i+1}\}).$$

Finalement, **l'ensemble en conflit minimal le plus précoce** (*earliest minimal conflict set*) de $\vec{a_i}$, noté $\text{emc}(\vec{a_i})$, est la restriction de $\vec{a_i}$ sur les variables de $\text{var-emc}(\vec{a_i})$:

$$\text{emc}(\vec{a_i}) = \vec{a_i}[\text{var-emc}(\vec{a_i})].$$

Définition

Le **jumpback set** d'une feuille sans issue $\vec{a_i}$ est son var-emc($\vec{a_i}$), soit

$$J_i = \text{var-emc}(\vec{a_i}).$$

Définition

Le **jumpback set** d'un nœud interne sans issue $\vec{a_i}$ est l'union de tous les var-emc($\vec{a_j}$) de toutes variables sans issues pertinentes $\vec{a_j}, j \geqslant i$ de la session de x_i, soit

$$J_i = \bigcup_{\vec{a_j} \in r(x_i)} \text{var-emc}(\vec{a_j}).$$

Proposition

Soient $\vec{a_i}$ une variable sans issue (feuille ou interne) et J_i son jumpback set, alors la dernière variable de J_i est la variable la plus précoce où il est valide de backjumper.

Algorithme : Graph based Backjumping

Entrées : un CSP (X, D, C)
$i \leftarrow 1$, $D'_i \leftarrow D_i$, $J_i \leftarrow \varnothing$
tant que $1 \leqslant i \leqslant n$ **faire**
  $x_i \leftarrow$ Select-Value-CBJ()
  si $x_i = -1$ **alors**
    $ip \leftarrow i$, $i \leftarrow \max J_i$,
    $J_i \leftarrow J_i \cup J_{ip} \backslash \{x_i\}$
  sinon
    $i \leftarrow i + 1$, $D'_i \leftarrow D_i$, $J_i \leftarrow \varnothing$
si $i = 0$ **alors**
  Renvoyer Inconsistant
sinon
  Renvoyer $(x_1, \ldots, x_n)$

Algorithme : Select-Value-CBJ

tant que D'_i **non vide faire**
  Choisir $a \in D'_i$, retirer a de D'_i
  $consist \leftarrow true$, $k \leftarrow 1$
  tant que $k < i$ **et** $consist$ **faire**
    si $(\vec{a_k}, x_i = a)$ **est consistant alors**
      $k \leftarrow k + 1$
    sinon
      $R_S \leftarrow$ plus petite contrainte à l'origine du conflit
      Ajouter les variables de $scope(R_S)$ sauf x_i à J_i
      $consist \leftarrow false$
  si $consist$ **alors** Renvoyer a;
Renvoyer -1

Learning

- shallow learning : dead-end devient un nogood
  - **dans Graph Based Backjumping** : en cas de backjump, on apprend la contrainte qui interdit $\vec{a_i}[li(Y)]$
  - **dans Conflit Directed Backjumping** : en cas de backjump, on apprend la contrainte qui interdit $\vec{a_i}[Ji]$
- deep learning : utiliser l'algo xplain ou quickxplain

Gestion de la mémoire dans un backtrack

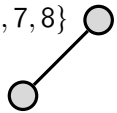
Lors d'un backtrack, il faut mémoriser l'état d'un sommet avant de brancher afin de pouvoir le rétablir si besoin. Toutes ces copies sont coûteuses.

$$D_x = \{1, 2, 3, 4, 5, 6, 7, 8\}$$ 

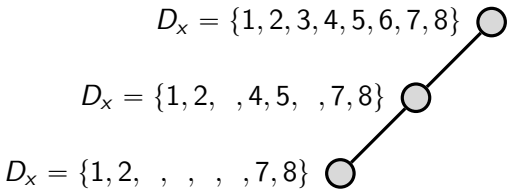
Lors d'un backtrack, il faut mémoriser l'état d'un sommet avant de brancher afin de pouvoir le rétablir si besoin. Toutes ces copies sont coûteuses.

$D_x = \{1, 2, 3, 4, 5, 6, 7, 8\}$

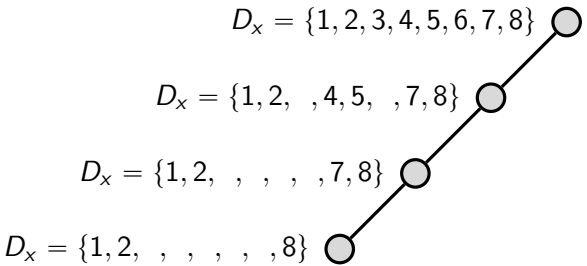
$D_x = \{1, 2, \ \ , 4, 5, \ \ , 7, 8\}$



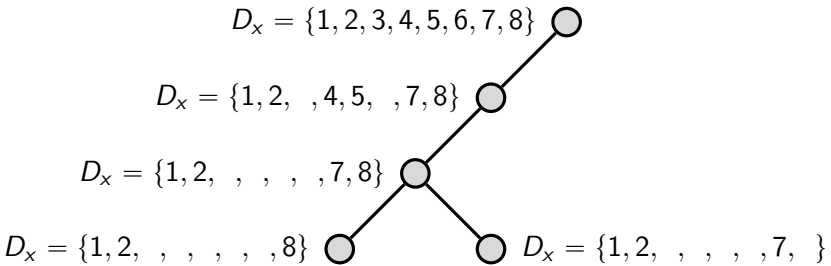
Lors d'un backtrack, il faut mémoriser l'état d'un sommet avant de brancher afin de pouvoir le rétablir si besoin. Toutes ces copies sont coûteuses.



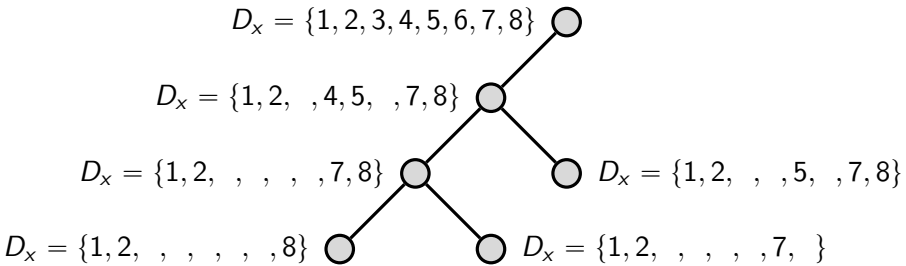
Lors d'un backtrack, il faut mémoriser l'état d'un sommet avant de brancher afin de pouvoir le rétablir si besoin. Toutes ces copies sont coûteuses.



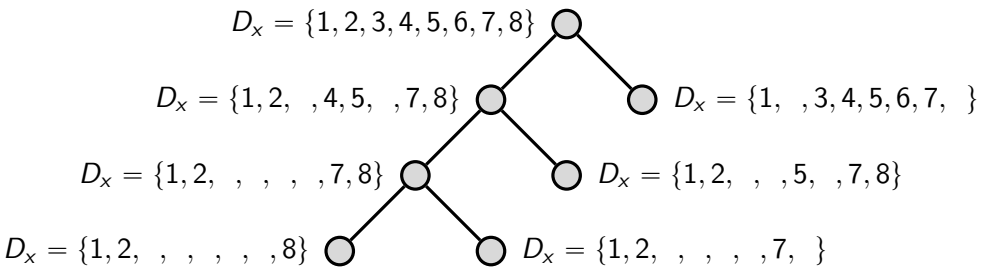
Lors d'un backtrack, il faut mémoriser l'état d'un sommet avant de brancher afin de pouvoir le rétablir si besoin. Toutes ces copies sont coûteuses.



Lors d'un backtrack, il faut mémoriser l'état d'un sommet avant de brancher afin de pouvoir le rétablir si besoin. Toutes ces copies sont coûteuses.

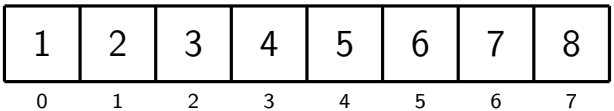


Lors d'un backtrack, il faut mémoriser l'état d'un sommet avant de brancher afin de pouvoir le rétablir si besoin. Toutes ces copies sont coûteuses.

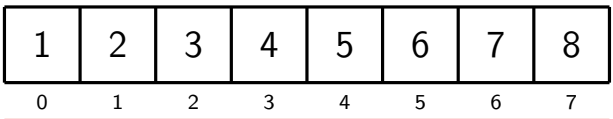


- dans un backtrack, les nœuds ouverts forment une lignée
- soient 2 nœuds ouverts N_1 et N_2 tels que N_2 est le fils de N_1
- alors $D_x(N_2) \subseteq D_x(N_1)$.
- idée : utiliser ces 2 constats pour améliorer la gestion des domaines

$D_x = \{1, 2, 3, 4, 5, 6, 7, 8\}$ 

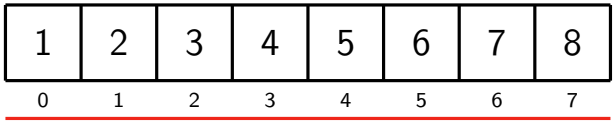


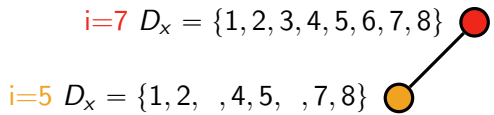
i=7 $D_x = \{1, 2, 3, 4, 5, 6, 7, 8\}$

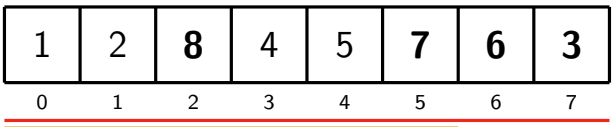


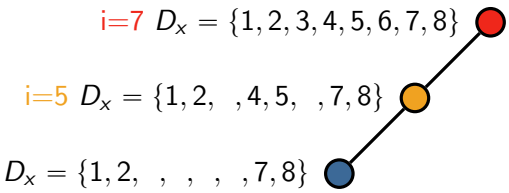
i=7 $D_x = \{1, 2, 3, 4, 5, 6, 7, 8\}$

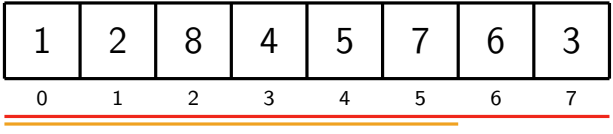
$D_x = \{1, 2, \ , 4, 5, \ , 7, 8\}$

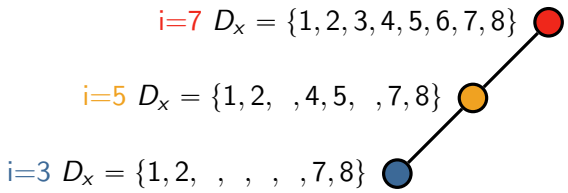


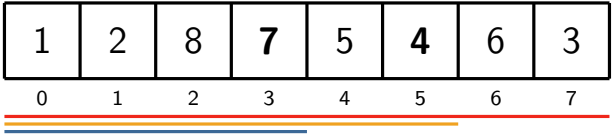

i=7 $D_x = \{1, 2, 3, 4, 5, 6, 7, 8\}$
i=5 $D_x = \{1, 2, \ , 4, 5, \ , 7, 8\}$

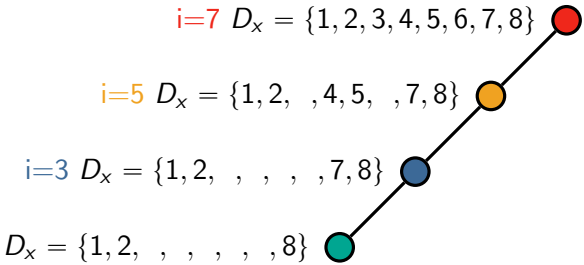

1 2 8 4 5 7 6 3
0 1 2 3 4 5 6 7

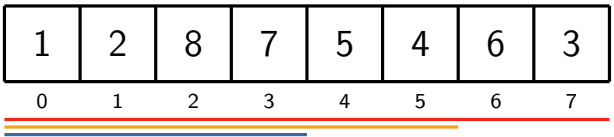

i=7 $D_x = \{1,2,3,4,5,6,7,8\}$
i=5 $D_x = \{1,2,\ ,4,5,\ ,7,8\}$
$D_x = \{1,2,\ ,\ ,\ ,\ ,7,8\}$

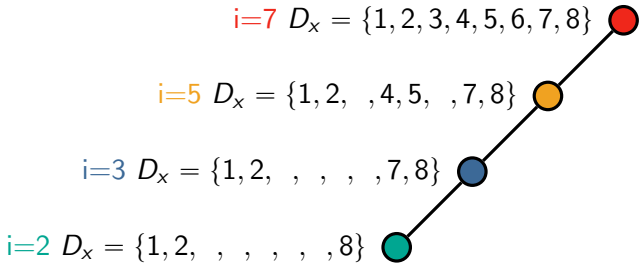

1 2 8 4 5 7 6 3
0 1 2 3 4 5 6 7

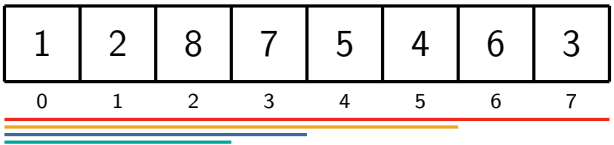

i=7 $D_x = \{1,2,3,4,5,6,7,8\}$
i=5 $D_x = \{1,2,\ ,4,5,\ ,7,8\}$
i=3 $D_x = \{1,2,\ ,\ ,\ ,\ ,7,8\}$

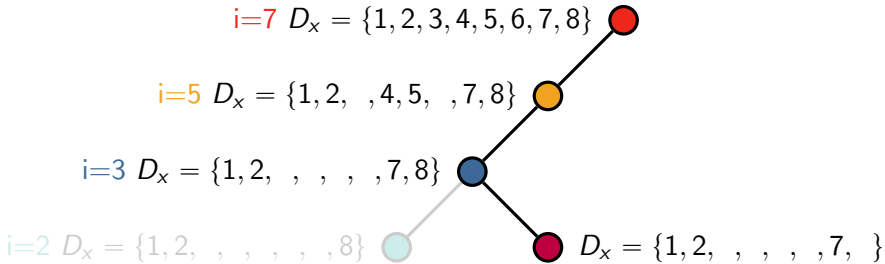

1 2 8 7 5 4 6 3
0 1 2 3 4 5 6 7

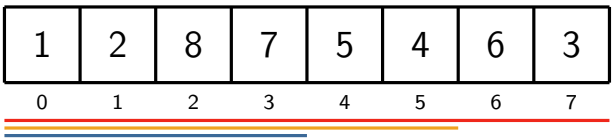

i=7 $D_x = \{1,2,3,4,5,6,7,8\}$
i=5 $D_x = \{1,2,\ ,4,5,\ ,7,8\}$
i=3 $D_x = \{1,2,\ ,\ ,\ ,\ ,7,8\}$
$D_x = \{1,2,\ ,\ ,\ ,\ ,\ ,8\}$

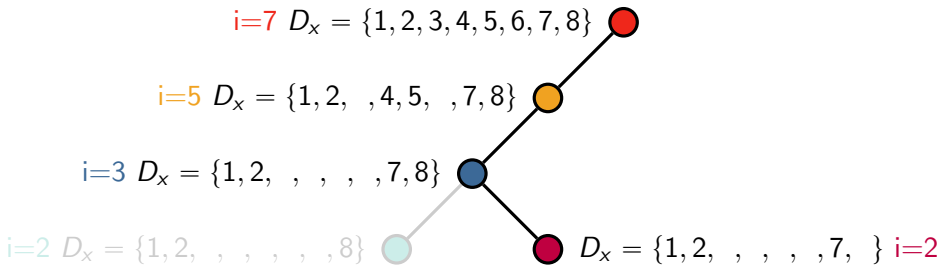

1 2 8 7 5 4 6 3
0 1 2 3 4 5 6 7

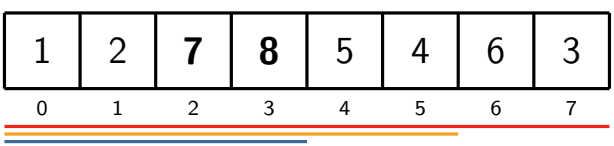

i=7 $D_x = \{1,2,3,4,5,6,7,8\}$
i=5 $D_x = \{1,2,\ ,4,5,\ ,7,8\}$
i=3 $D_x = \{1,2,\ ,\ ,\ ,\ ,7,8\}$
i=2 $D_x = \{1,2,\ ,\ ,\ ,\ ,\ ,8\}$

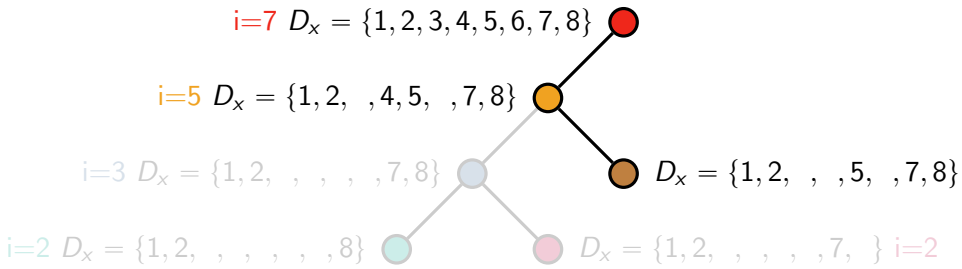

1 2 8 7 5 4 6 3
0 1 2 3 4 5 6 7


i=7 $D_x = \{1,2,3,4,5,6,7,8\}$
i=5 $D_x = \{1,2,\ ,4,5,\ ,7,8\}$
i=3 $D_x = \{1,2,\ ,\ ,\ ,\ ,7,8\}$
i=2 $D_x = \{1,2,\ ,\ ,\ ,\ ,\ ,8\}$
$D_x = \{1,2,\ ,\ ,\ ,\ ,7,\ \}$

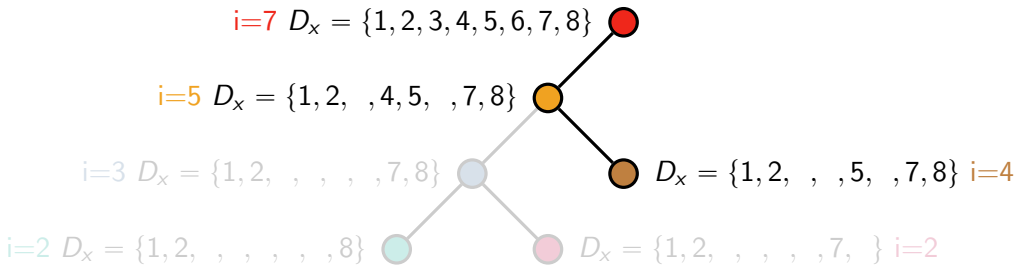

1 2 8 7 5 4 6 3
0 1 2 3 4 5 6 7


i=7 $D_x = \{1,2,3,4,5,6,7,8\}$
i=5 $D_x = \{1,2,\ ,4,5,\ ,7,8\}$
i=3 $D_x = \{1,2,\ ,\ ,\ ,\ ,7,8\}$
i=2 $D_x = \{1,2,\ ,\ ,\ ,\ ,\ ,8\}$
$D_x = \{1,2,\ ,\ ,\ ,\ ,7,\ \}$ i=2

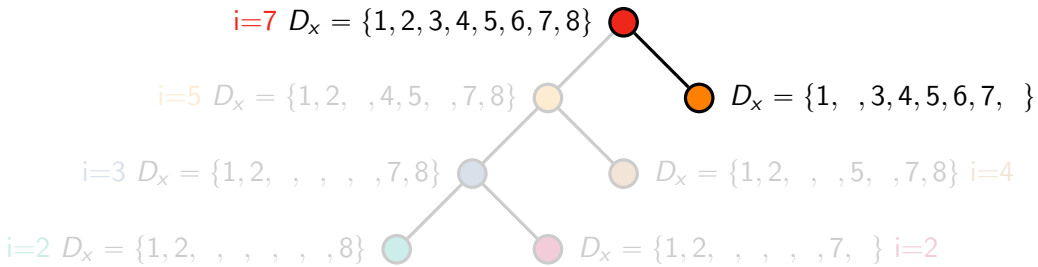

1
2
7
8
5
4
6
3
0
1
2
3
4
5
6
7



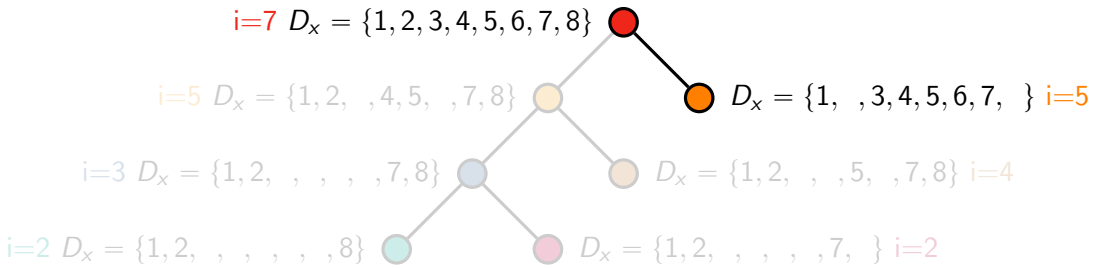
1	2	7	8	5	4	6	3
0	1	2	3	4	5	6	7

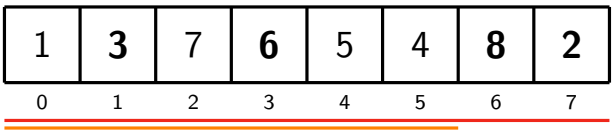


1	2	7	8	5	4	6	3
0	1	2	3	4	5	6	7



1	2	7	8	5	4	6	3
0	1	2	3	4	5	6	7


i=7 $D_x = \{1,2,3,4,5,6,7,8\}$
i=5 $D_x = \{1,2,\ ,4,5,\ ,7,8\}$
$D_x = \{1,\ ,3,4,5,6,7,\ \}$ i=5
i=3 $D_x = \{1,2,\ ,\ ,\ ,\ ,7,8\}$
$D_x = \{1,2,\ ,\ ,5,\ ,7,8\}$ i=4
i=2 $D_x = \{1,2,\ ,\ ,\ ,\ ,\ ,8\}$
$D_x = \{1,2,\ ,\ ,\ ,\ ,7,\ \}$ i=2


1
3
7
6
5
4
8
2
0
1
2
3
4
5
6
7