

Programmation Par Contraintes

Cours 3 - Contraintes globales, Look-back

David Savourey

CNRS, École Polytechnique

Séance 3

inspiré des cours de Philippe Baptiste, Ruslan Sadykov et de la thèse d'Hadrien
Cambazard

Sommaire

Contraintes globales

Apprentissage : backtrack intelligent

Gestion de la mémoire dans un backtrack

Sommaire

Contraintes globales

Apprentissage : backtrack intelligent

Gestion de la mémoire dans un backtrack

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

All-different

- ▶ 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

Autres contraintes globales

- ▶ *sum*
- ▶ *knapsack*
- ▶ *cumulative*
- ▶ *element*
- ▶ souvent spécifique à un type de problèmes
- ▶ catalogue : <http://www.emn.fr/z-info/sdemasse/gccat/>

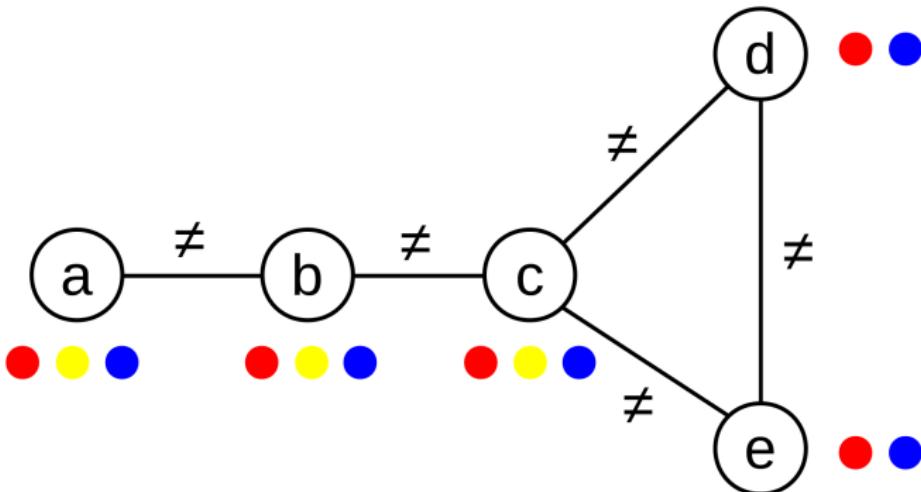
Sommaire

Contraintes globales

Apprentissage : backtrack intelligent

Gestion de la mémoire dans un backtrack

Deux exemples de thrashing



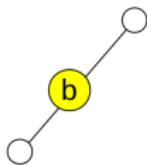
Un premier exemple de thrashing

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



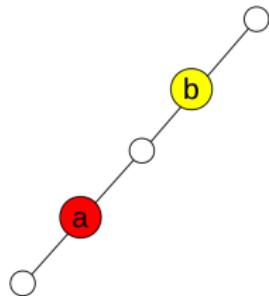
Un premier exemple de thrashing

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



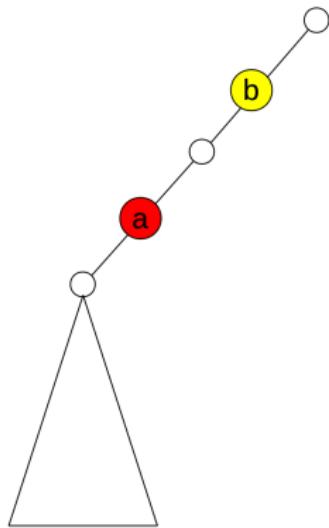
Un premier exemple de thrashing

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



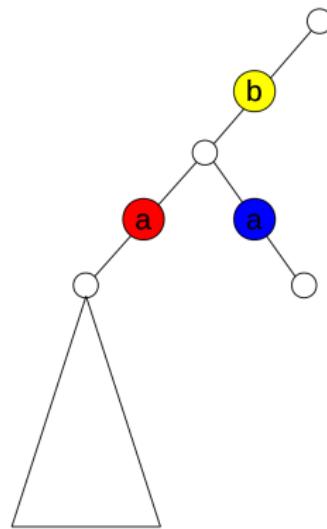
Un premier exemple de thrashing

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



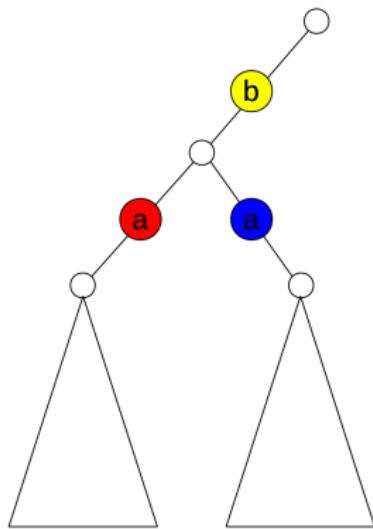
Un premier exemple de thrashing

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



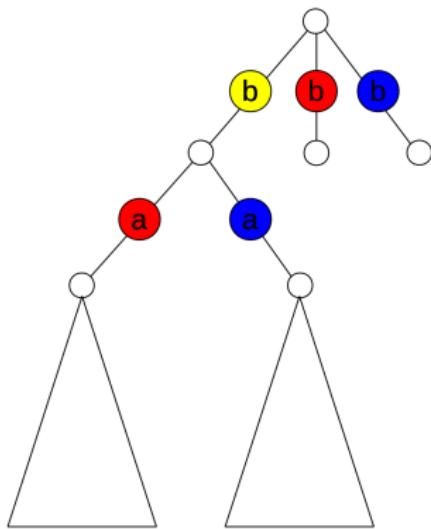
Un premier exemple de thrashing

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



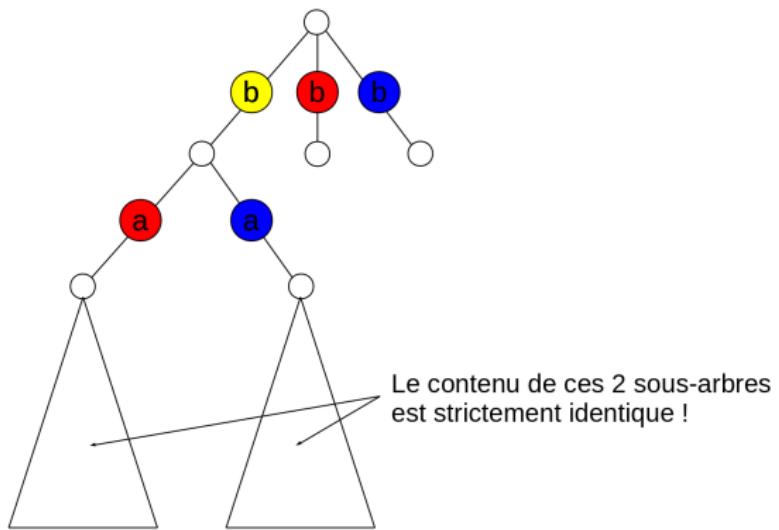
Un premier exemple de thrashing

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



Un premier exemple de thrashing

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



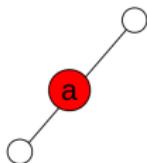
Un deuxième exemple de thrashing

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



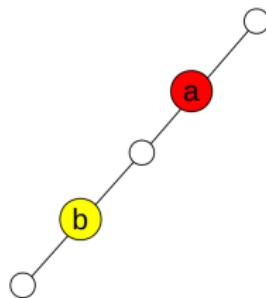
Un deuxième exemple de thrashing

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



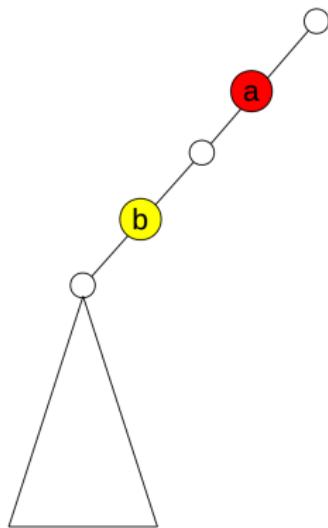
Un deuxième exemple de thrashing

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



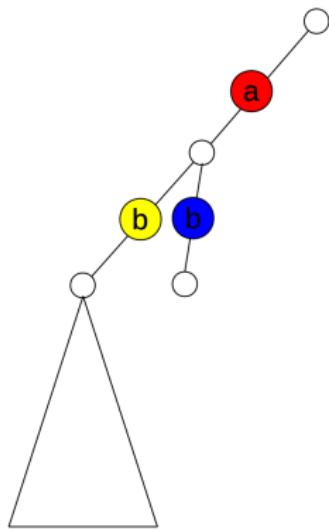
Un deuxième exemple de thrashing

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



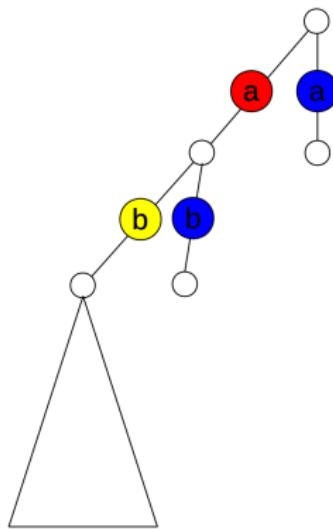
Un deuxième exemple de thrashing

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



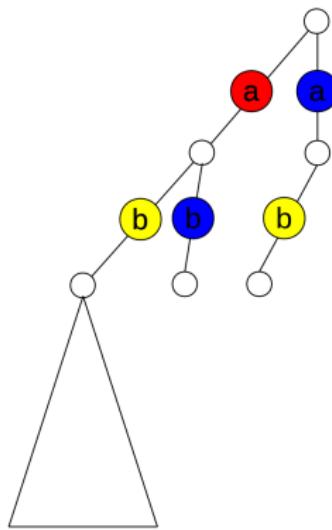
Un deuxième exemple de thrashing

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



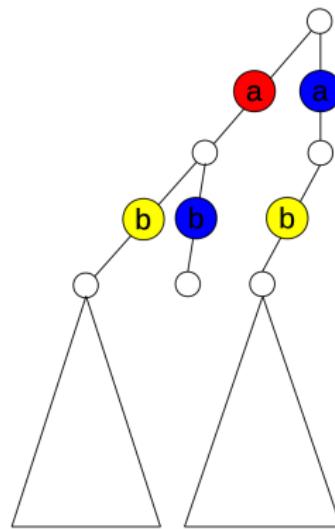
Un deuxième exemple de thrashing

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



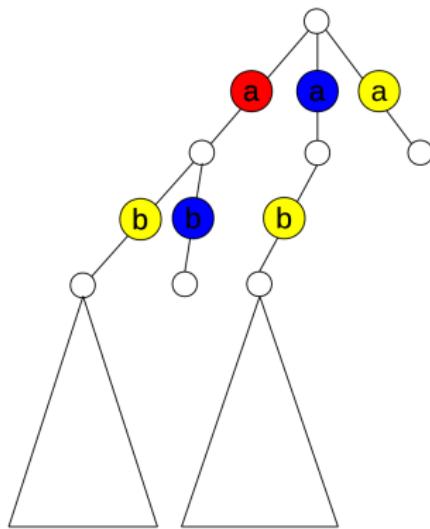
Un deuxième exemple de thrashing

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



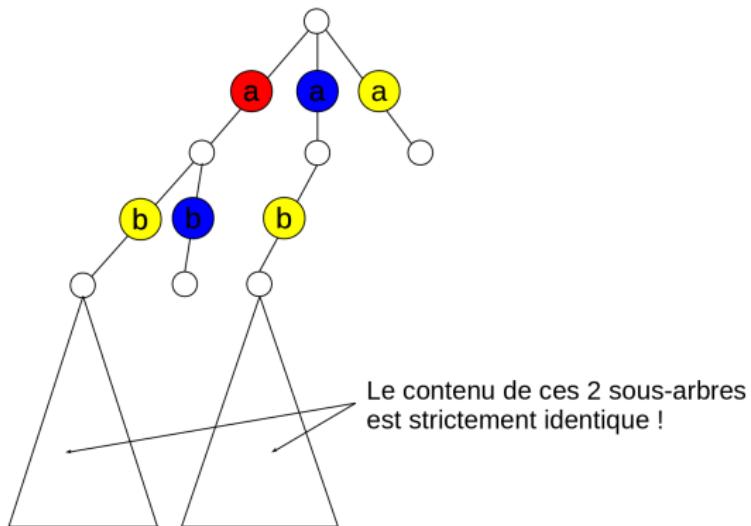
Un deuxième exemple de thrashing

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



Un deuxième exemple de thrashing

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



Comment limiter le thrashing?

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

Les explications

- ▶ 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 $\text{expl}(x \neq a) = C_e \cup DC_e$ une telle explication.
- ▶ C_e : conflict set, DC_e : no goods
- ▶ Une explication de contradiction est l'explication de l'absence de valeurs pour une variable x ($D_x = \emptyset$), de sorte que :

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

Calcul de l'explication

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

Algorithme 1 : Xplain

Données : C, m

$X \leftarrow \emptyset ;$

$X_{tmp} \leftarrow \emptyset ;$

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

$k \leftarrow 0 ;$

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

$k \leftarrow k + 1 ;$

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

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

;

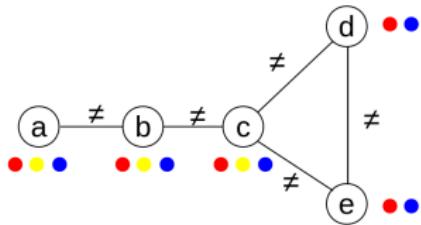
$X \leftarrow X \cup \{C_k\} ;$

$X_{tmp} \leftarrow X ;$

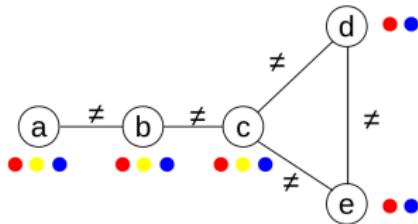
Retourner X ;

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

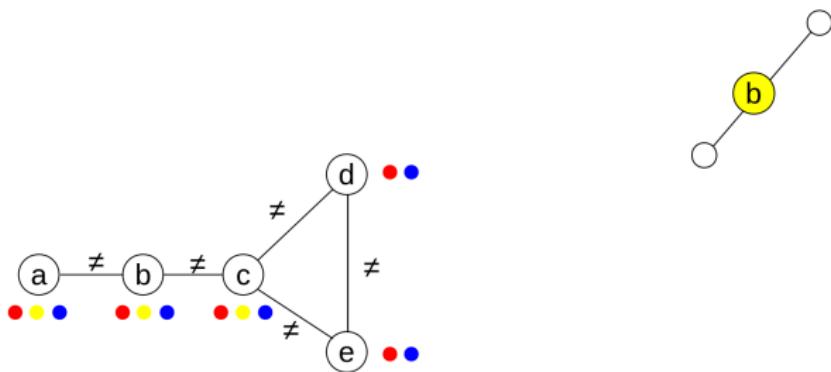
Premier exemple : backjumping



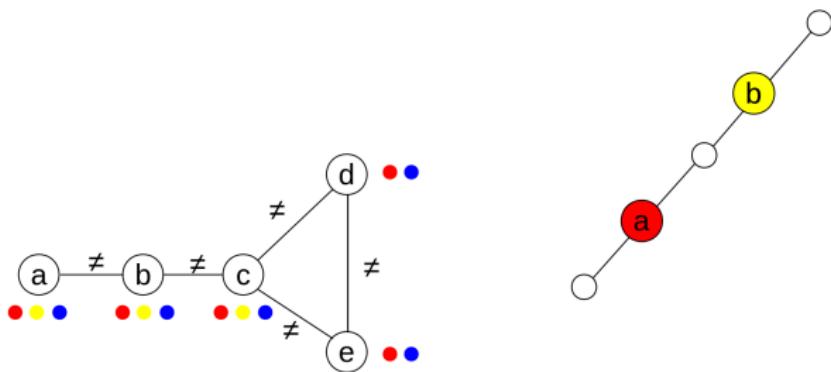
Premier exemple : backjumping



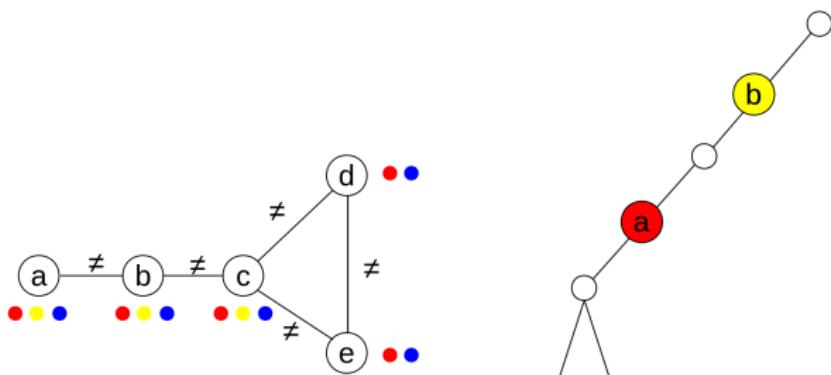
Premier exemple : backjumping



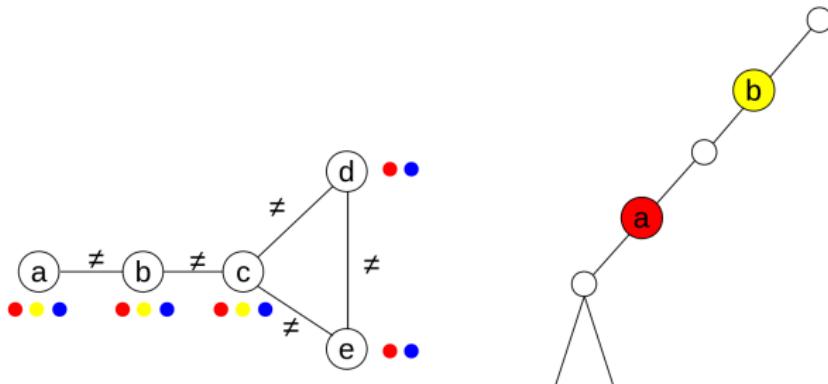
Premier exemple : backjumping



Premier exemple : backjumping

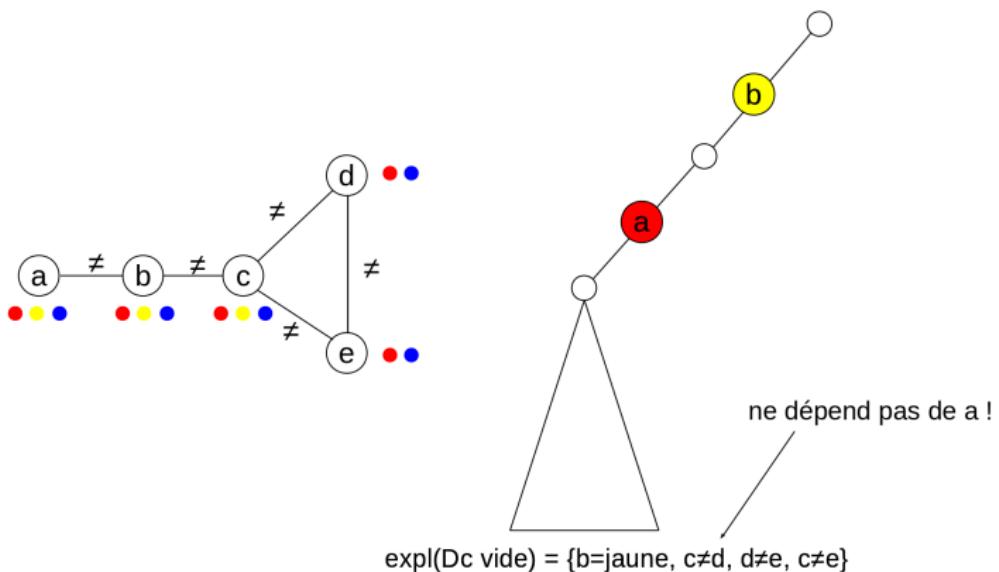


Premier exemple : backjumping

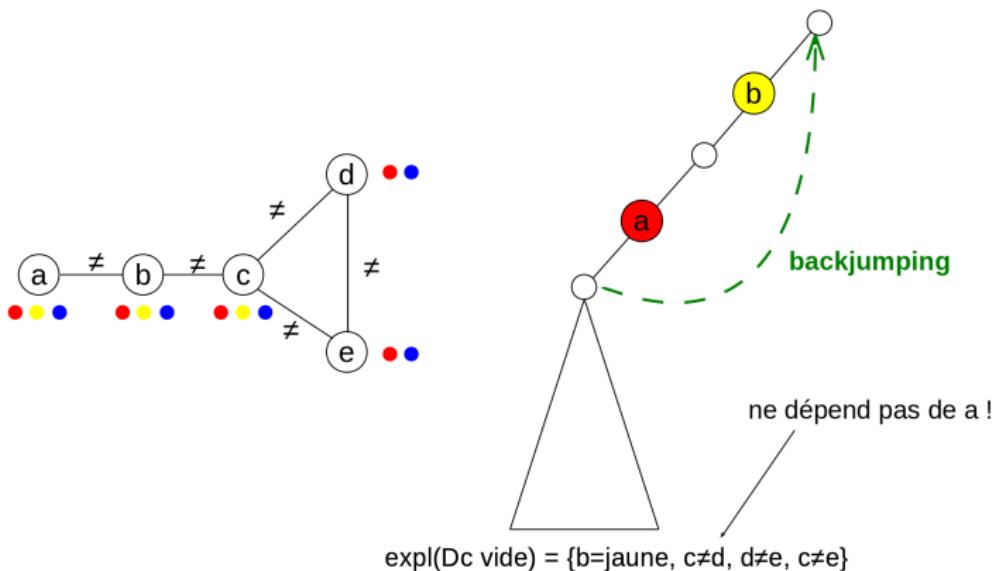


$\text{expl}(\text{Dc vide}) = \{b=\text{jaune}, c \neq d, d \neq e, c \neq e\}$

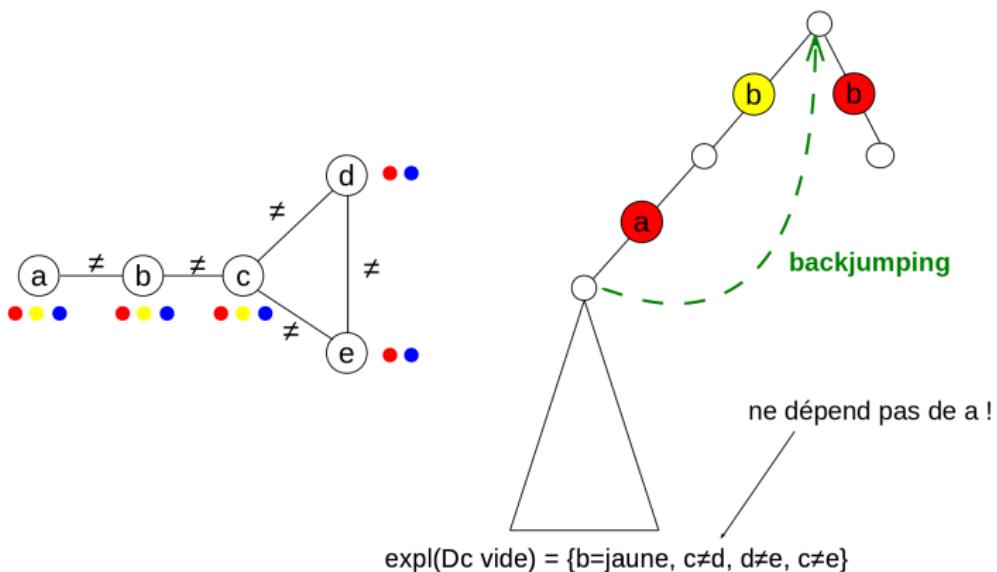
Premier exemple : backjumping



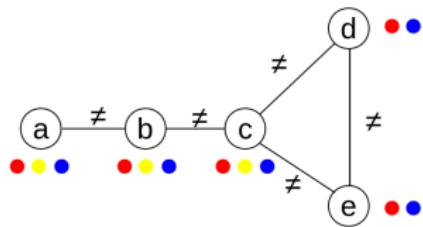
Premier exemple : backjumping



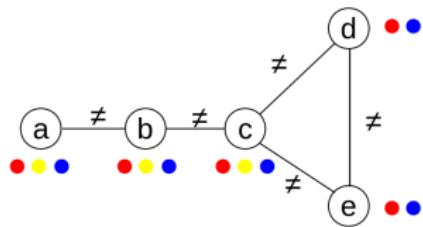
Premier exemple : backjumping



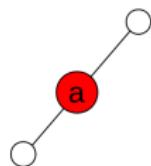
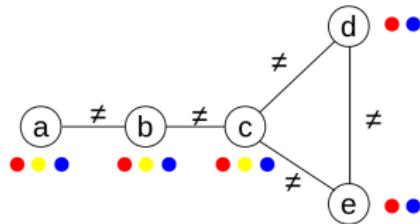
Deuxième exemple : learning



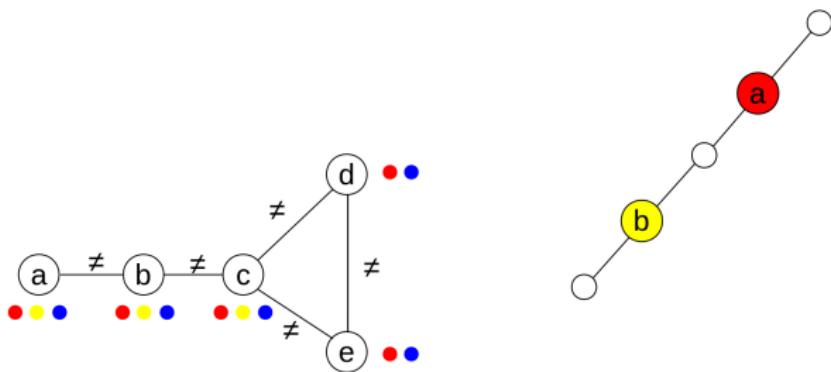
Deuxième exemple : learning



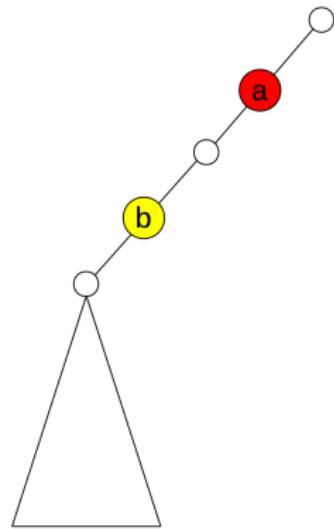
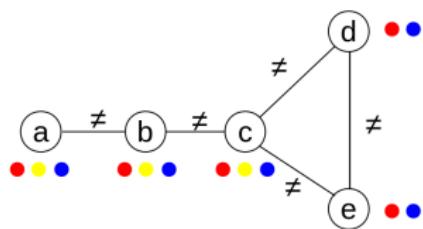
Deuxième exemple : learning



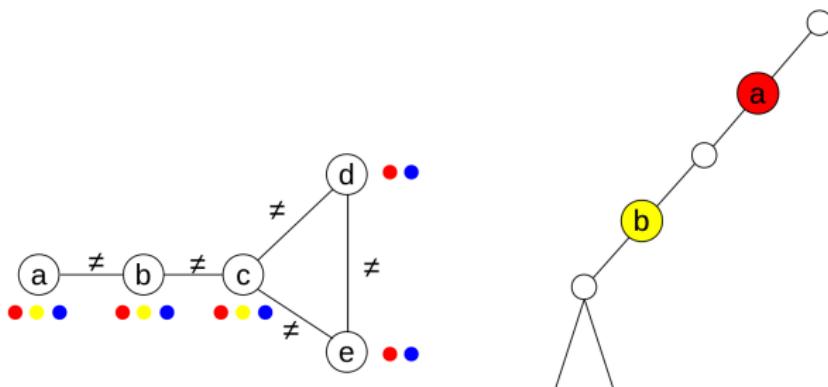
Deuxième exemple : learning



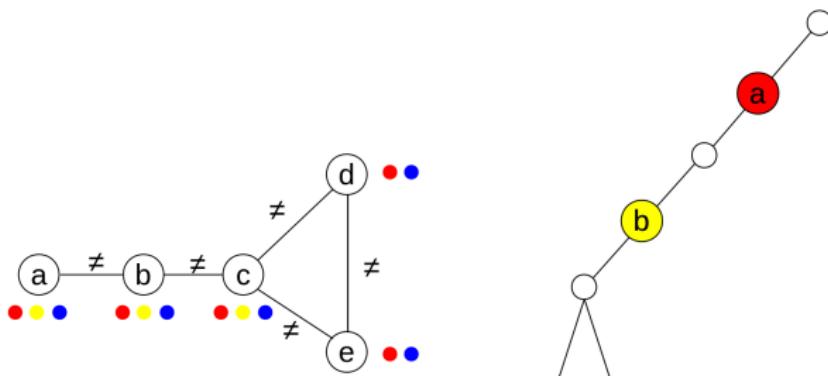
Deuxième exemple : learning



Deuxième exemple : learning



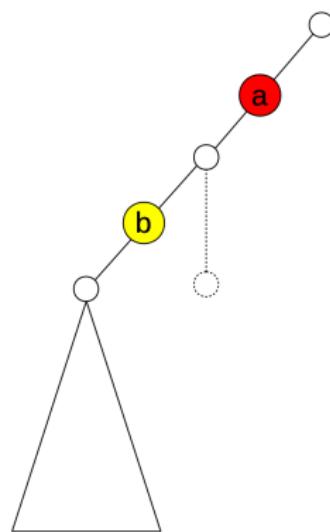
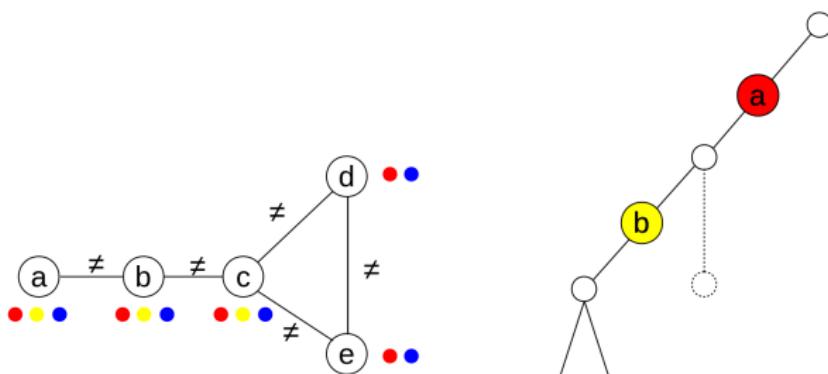
Deuxième exemple : learning



$\text{expl}(\text{Dc vide}) = \{b=\text{jaune}, c \neq d, d \neq e, c \neq e\}$

No goods
b=jaune

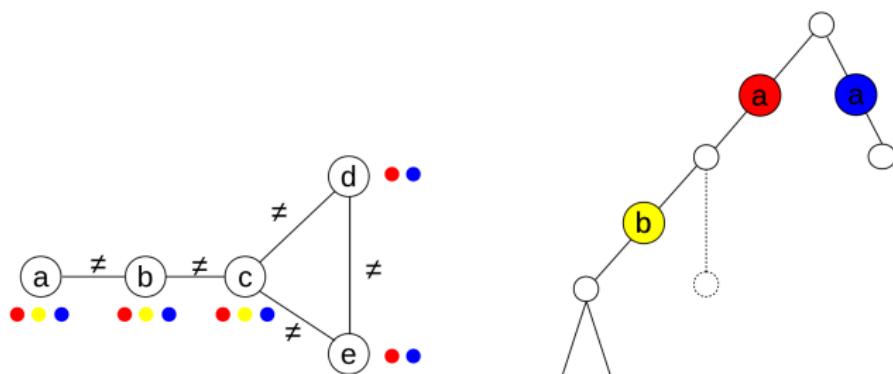
Deuxième exemple : learning



$\text{expl}(\text{Dc vide}) = \{b=\text{jaune}, c \neq d, d \neq e, c \neq e\}$

No goods
b=jaune

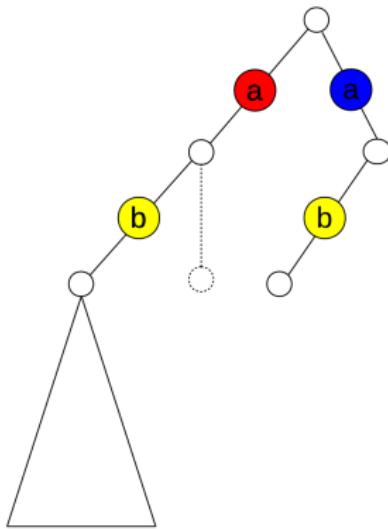
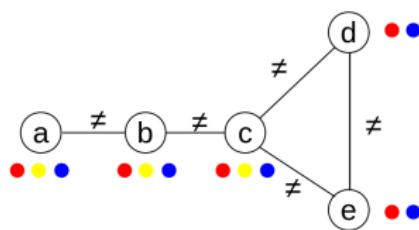
Deuxième exemple : learning



$\text{expl}(\text{Dc vide}) = \{b=\text{jaune}, c \neq d, d \neq e, c \neq e\}$

No goods
b=jaune

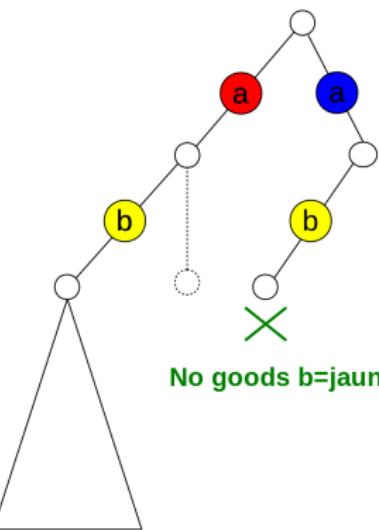
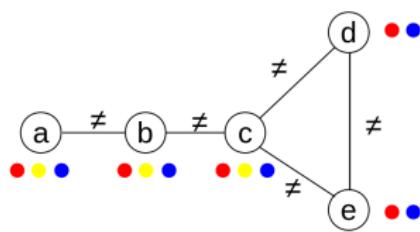
Deuxième exemple : learning



$\text{expl}(\text{Dc vide}) = \{b=\text{jaune}, c \neq d, d \neq e, c \neq e\}$

No goods
b=jaune

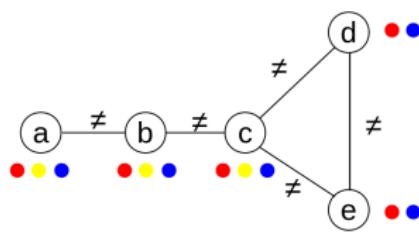
Deuxième exemple : learning



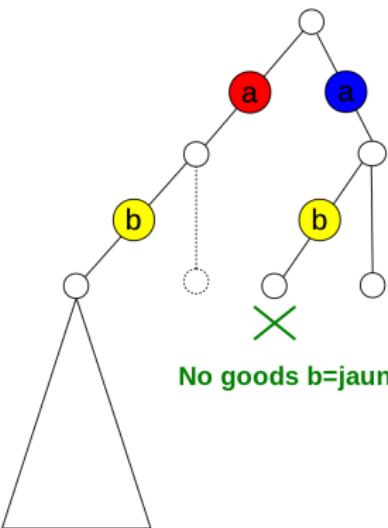
$\text{expl}(\text{Dc vide}) = \{b=\text{jaune}, c \neq d, d \neq e, c \neq e\}$

No goods
b=jaune

Deuxième exemple : learning



$\text{expl}(\text{Dc vide}) = \{b=\text{jaune}, c \neq d, d \neq e, c \neq e\}$



No goods b=jaune

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

Algo Gaschnig's backjump

```
procedure GASCHNIG'S-BACKJUMPING
```

Input: A constraint network $\mathcal{R} = (X, D, C)$.

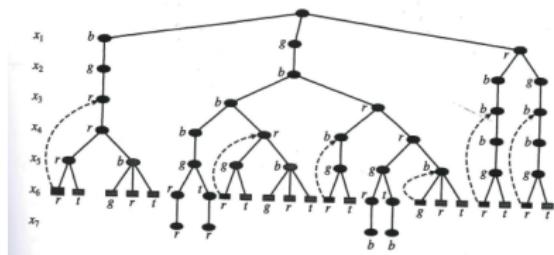
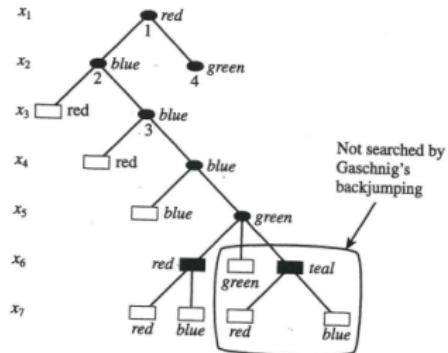
Output: Either a solution, or a decision that the network is inconsistent.

```
i ← 1          (initialize variable counter)
Di ← Di      (copy domain)
latesti ← 0    (initialize pointer to culprit)
while 1 ≤ i ≤ n
    instantiate xi ← SELECT-VALUE-GBJ
    if xi is null   (no value was returned)
        i ← latesti (backjump)
    else
        i ← i + 1
        Di ← Di
        latesti ← 0
    end while
    if i = 0
        return "inconsistent"
    else
        return instantiated values of (x1, ..., xn)
end procedure
```



```
procedure SELECT-VALUE-GBJ
    while Di is not empty
        select an arbitrary element a ∈ Di and remove a from Di
        consistent ← true
        k ← 1
        while k < i and consistent
            if k > latesti
                latesti ← k
            if not CONSISTENT(̄k, xi = a)
                consistent ← false
            else
                k ← k + 1
            end while
            if consistent
                return a
            end while
            return null           (no consistent value)
end procedure
```

Exemple



Algo Graph Based backjump

```
procedure GRAPH-BASED-BACKJUMPING
Input: A constraint network  $\mathcal{R} = (X, D, C)$ .
Output: Either a solution, or a decision that the network is inconsistent.

compute anc( $x_i$ ) for each  $x_i$  (see Definition 6.7 in text)
 $i \leftarrow 1$                                 (initialize variable counter)
 $D'_i \leftarrow D_i$                             (copy domain)
 $l_i \leftarrow \text{anc}(x_i)$                   (copy of anc() that can change)

while  $1 \leq i \leq n$ 
    instantiate  $x_i \leftarrow \text{SELECT-VALUE}$ 
    if  $x_i$  is null                         (no value was returned)
         $i_{\text{prev}} \leftarrow i$ 
         $i \leftarrow \text{latest index in } l_i$       (backjump)
         $l_i \leftarrow l_i \cup l_{\text{prev}} - \{x_i\}$ 
    else
         $i \leftarrow i + 1$ 
         $D'_i \leftarrow D_i$ 
         $l_i \leftarrow \text{anc}(x_i)$ 
    end while
    if  $i = 0$ 
        return "inconsistent"
    else
        return instantiated values of  $\{x_1, \dots, x_n\}$ 
end procedure

procedure SELECT-VALUE          (same as BACKTRACKING's)
    while  $D'_i$  is not empty
        select an arbitrary element  $a \in D'_i$ , and remove  $a$  from  $D'_i$ 
        if CONSISTENT( $\vec{\theta}_{i-1}, x_i = a$ )
            return  $a$ 
    end while
    return null                          (no consistent value)
end procedure
```

Algo Conflit Directed backjump

```
procedure CONFLICT-DIRECTED-BACKJUMPING
Input: A constraint network  $\mathcal{R} = (X, D, C)$ .
Output: Either a solution, or a decision that the network is inconsistent.

i  $\leftarrow 1$                                 (initialize variable counter)
 $D'_i \leftarrow D_i$                           (copy domain)
 $J_i \leftarrow \emptyset$                       (initialize conflict set)
while  $1 \leq i \leq n$ 
  instantiate  $x_i \leftarrow \text{SELECT-VALUE-CB}(J_i)$ 
  if  $x_i$  is null                         (no value was returned)
     $j_{\text{prev}} \leftarrow i$ 
     $i \leftarrow$  index of last variable in  $J_i$  (backjump)
     $J_i \leftarrow J_i \cup J_{j_{\text{prev}}} - \{x_i\}$  (merge conflict sets)
  else
     $j \leftarrow i + 1$                       (step forward)
     $D'_i \leftarrow D_i$                       (reset reusable domain)
     $J_i \leftarrow \emptyset$                   (reset conflict set)
  end while
  if  $i = 0$ 
    return "inconsistent"
  else
    return instantiated values of  $\{x_1, \dots, x_n\}$ 
  end procedure

subprocedure SELECT-VALUE-CB()
  while  $D'_i$  is not empty
    select an arbitrary element  $a \in D'_i$  and remove  $a$  from  $D'_i$ 
     $\text{consistent} \leftarrow \text{true}$ 
     $k \leftarrow i$ 
    while  $k < i$  and  $\text{consistent}$ 
      if  $\text{CONSISTENT}(S_k, x_i = a)$ 
         $k \leftarrow k + 1$ 
      else
         $R_S \leftarrow$  the earliest constraint causing the conflict,
        add the variables in  $R_S$ 's scope  $S$  excluding  $x_i$  to  $J_i$ 
         $\text{consistent} \leftarrow \text{false}$ 
      end while
    if  $\text{consistent}$ 
      return  $a$ 
    end while
  return null                               (no consistent value)
end procedure
```

Learning

- ▶ shallow learning : dead-end devient un nogood
- ▶ deep learning : utiliser l'algo xplain ou quickxplain

```
procedure GRAPH-BASED-BACKJUMP-LEARNING
    instantiate  $x_i \leftarrow \text{SELECT-VALUE}$ 
    if  $x_i$  is null           (no value was returned)
        record a constraint prohibiting  $\bar{a}_{i-1}[l_i]$ .
    iprev  $\leftarrow i$ 
    (algorithm continues as in Figure 6.5)
```

```
procedure CONFLICT-DIRECTED-BACKJUMP-LEARNING
    instantiate  $x_i \leftarrow \text{SELECT-VALUE-CBJ}$ 
    if  $x_i$  is null           (no value was returned)
        record a constraint prohibiting  $\bar{a}_{i-1}[J_i]$ 
    iprev  $\leftarrow i$ 
    (algorithm continues as in Figure 6.6)
```

Sommaire

Contraintes globales

Apprentissage : backtrack intelligent

Gestion de la mémoire dans un backtrack

Contexte

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.

Contexte

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.



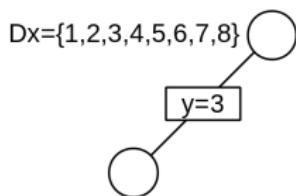
Contexte

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.

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

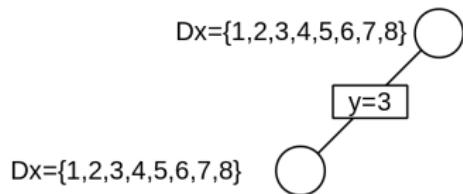
Contexte

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.



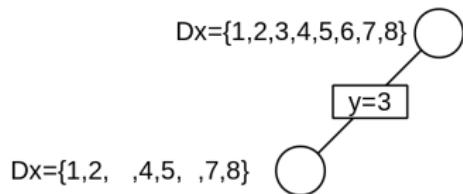
Contexte

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.



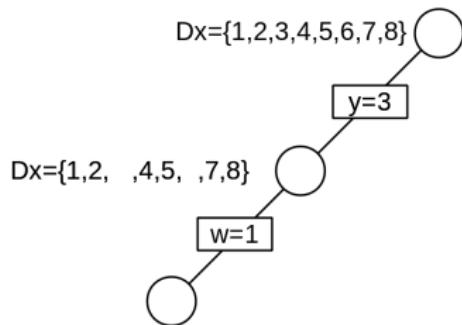
Contexte

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.



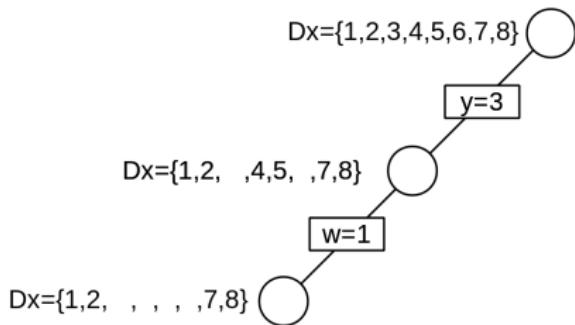
Contexte

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.



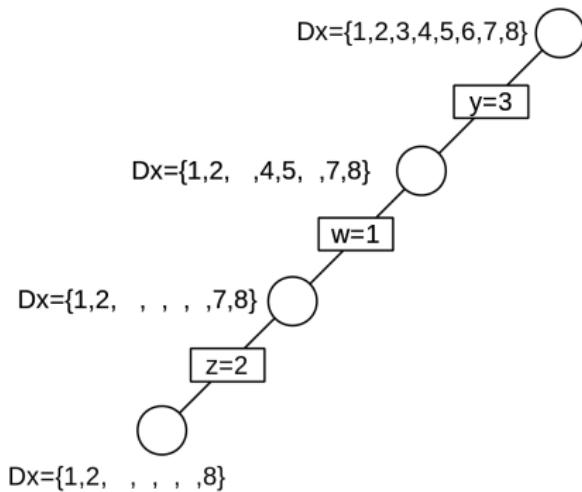
Contexte

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.



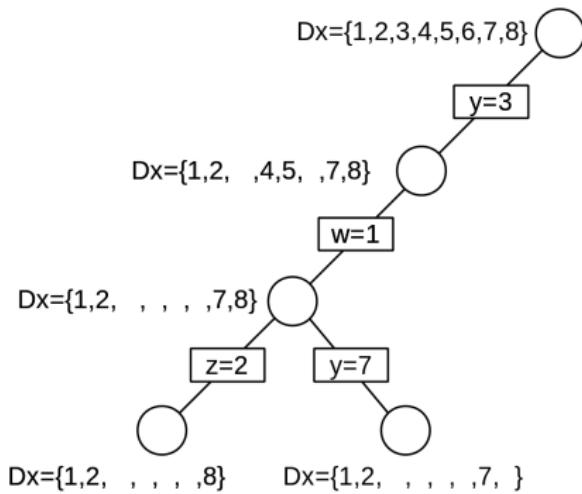
Contexte

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.



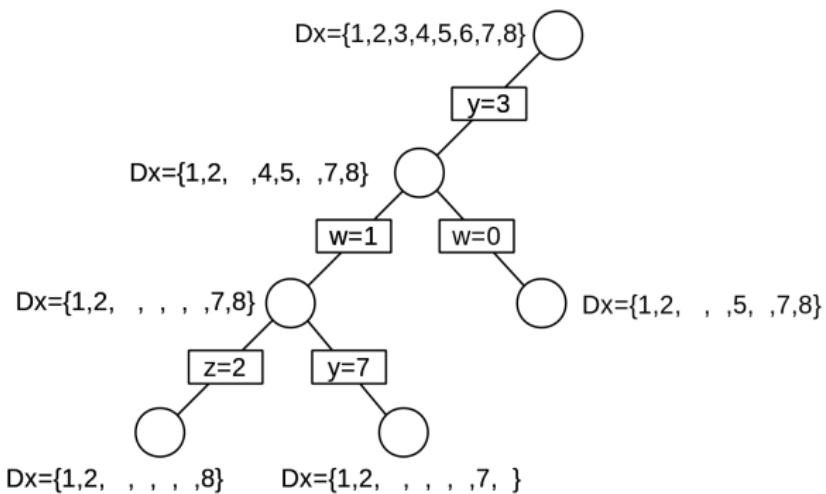
Contexte

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.



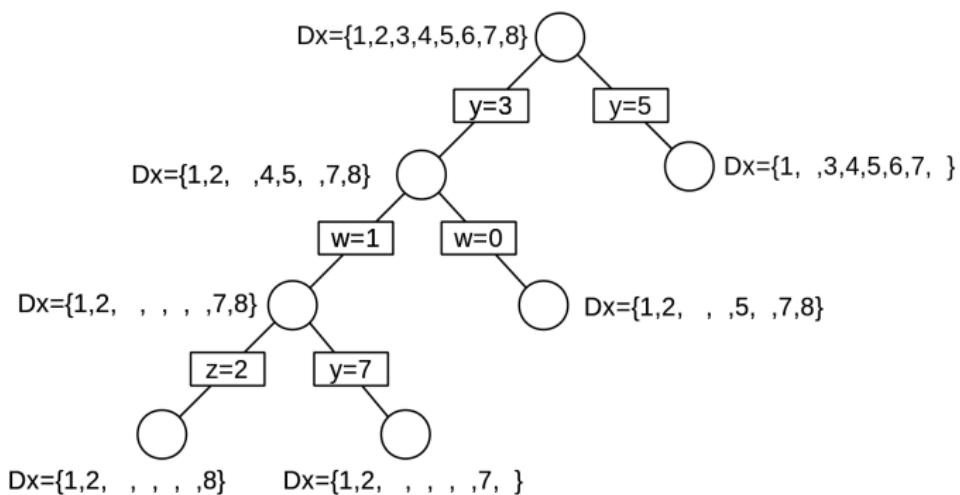
Contexte

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.



Contexte

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.



Une proposition

- ▶ 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

Une proposition en images



Une proposition en images

Dx={1,2,3,4,5,6,7,8} 

Une proposition en images

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

Dx={1,2,3,4,5,6,7,8}



Une proposition en images

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



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

$ix = 7$



Une proposition en images

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

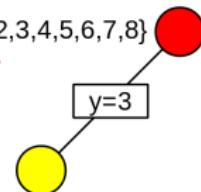


$Dx = \{1, 2, 3, 4, 5, 6, 7, 8\}$

$ix = 7$

$y = 3$

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



Une proposition en images

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

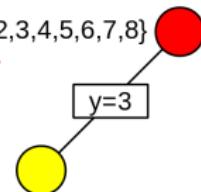


$Dx = \{1, 2, 3, 4, 5, 6, 7, 8\}$

$ix = 7$

$y = 3$

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



Une proposition en images

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

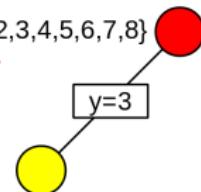


$Dx = \{1, 2, 3, 4, 5, 6, 7, 8\}$

$ix = 7$

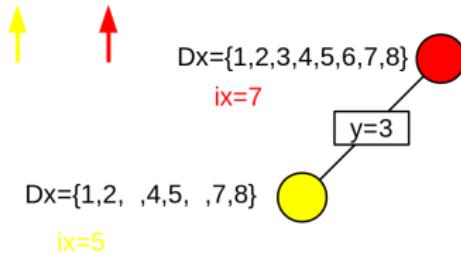
$y = 3$

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

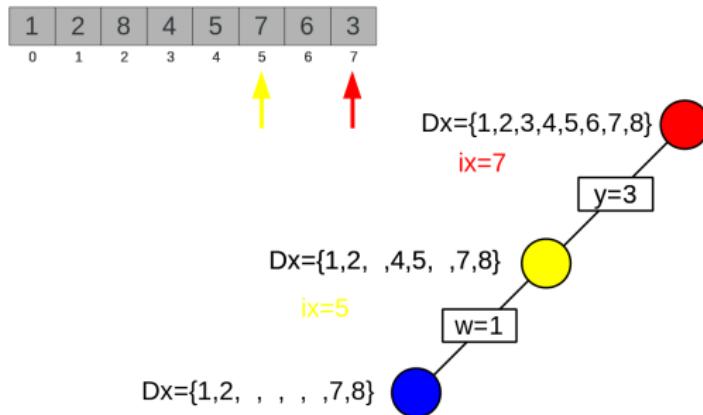


Une proposition en images

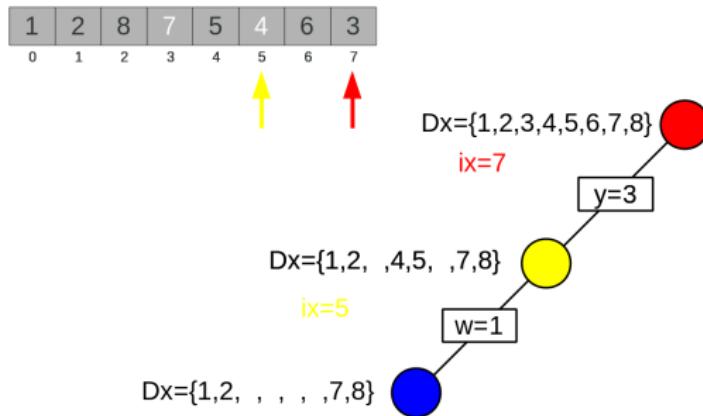
1	2	8	4	5	7	6	3
0	1	2	3	4	5	6	7



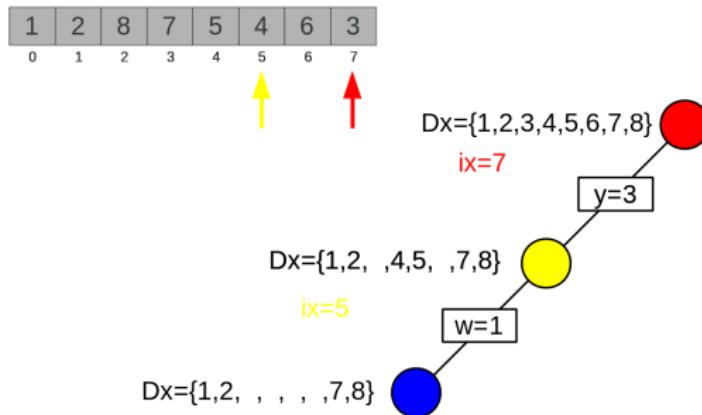
Une proposition en images



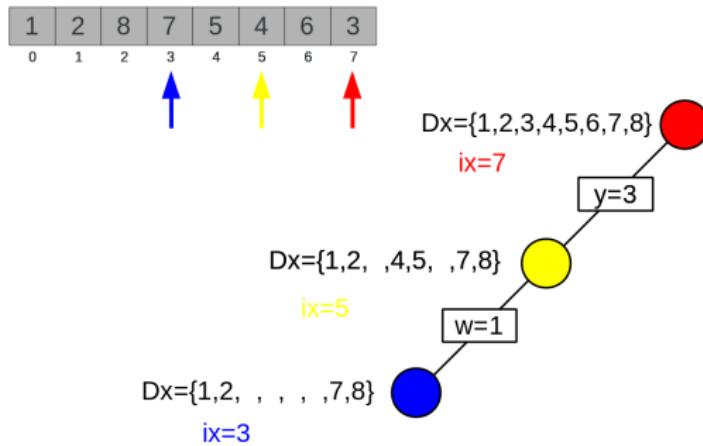
Une proposition en images



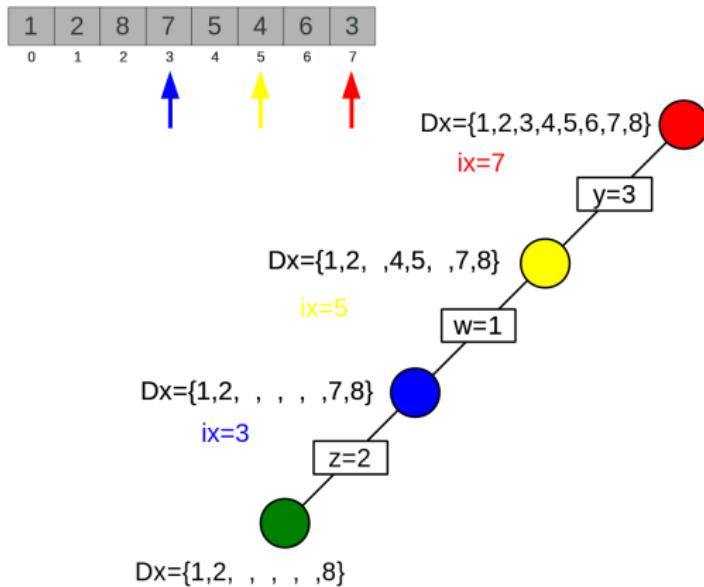
Une proposition en images



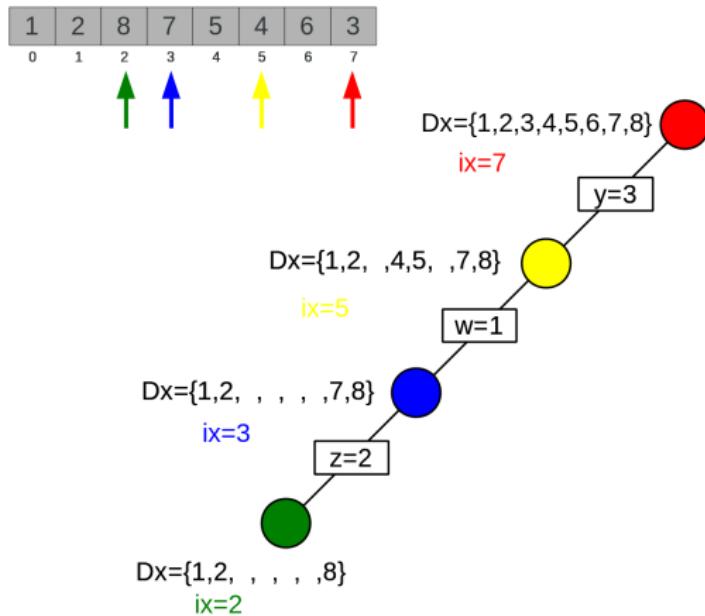
Une proposition en images



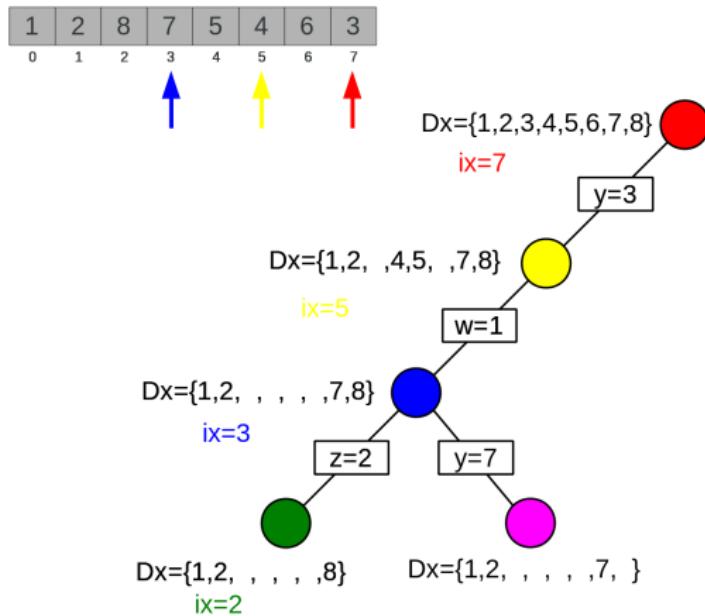
Une proposition en images



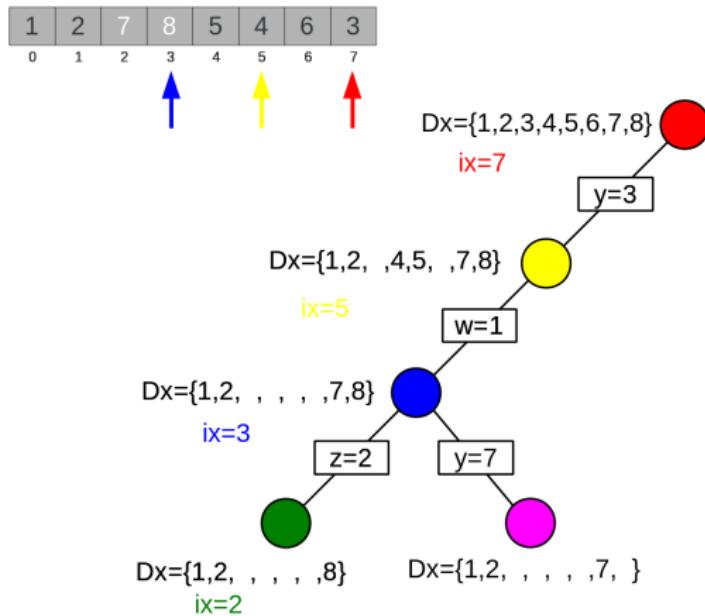
Une proposition en images



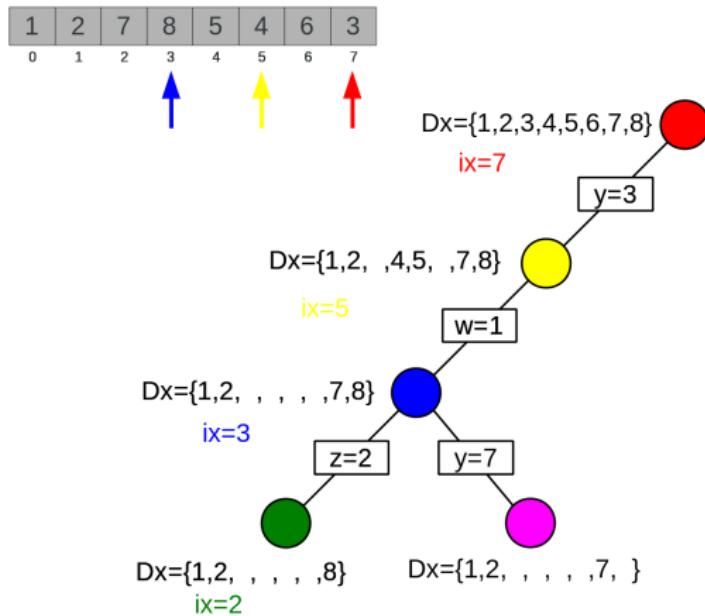
Une proposition en images



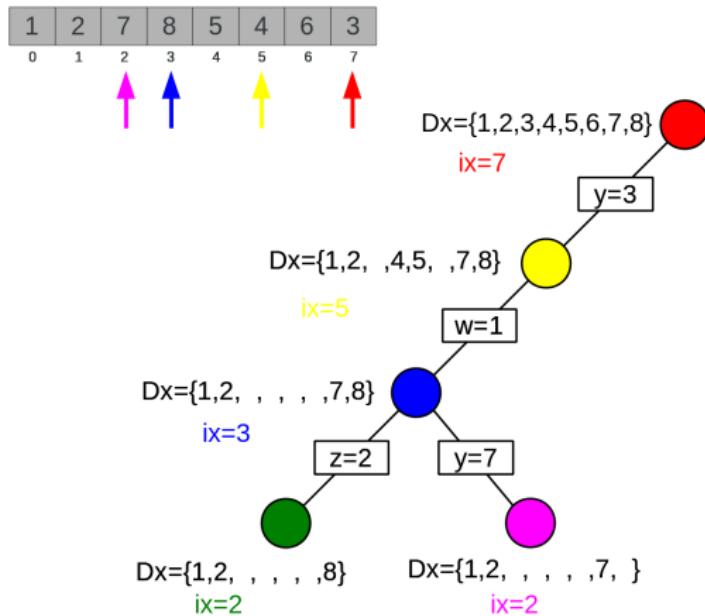
Une proposition en images



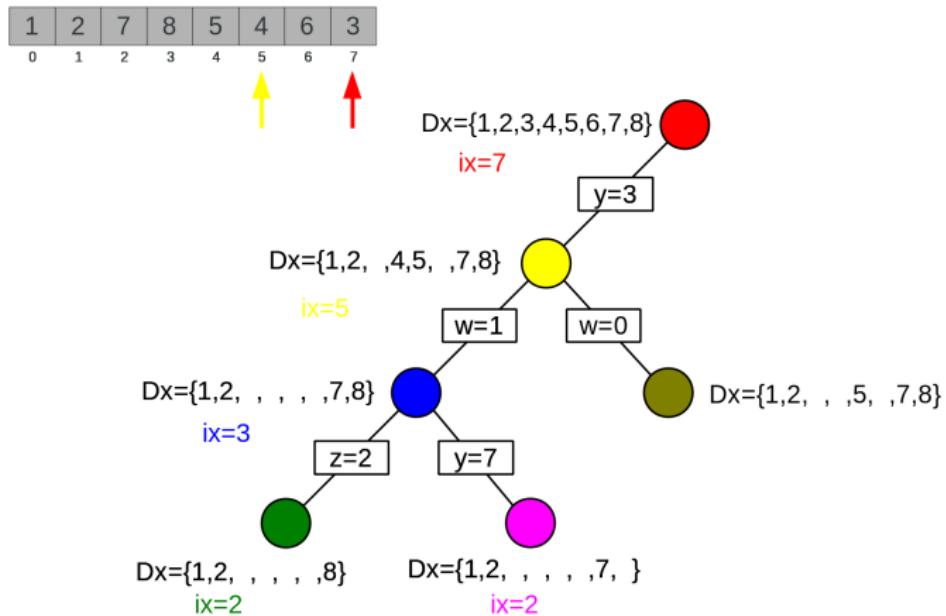
Une proposition en images



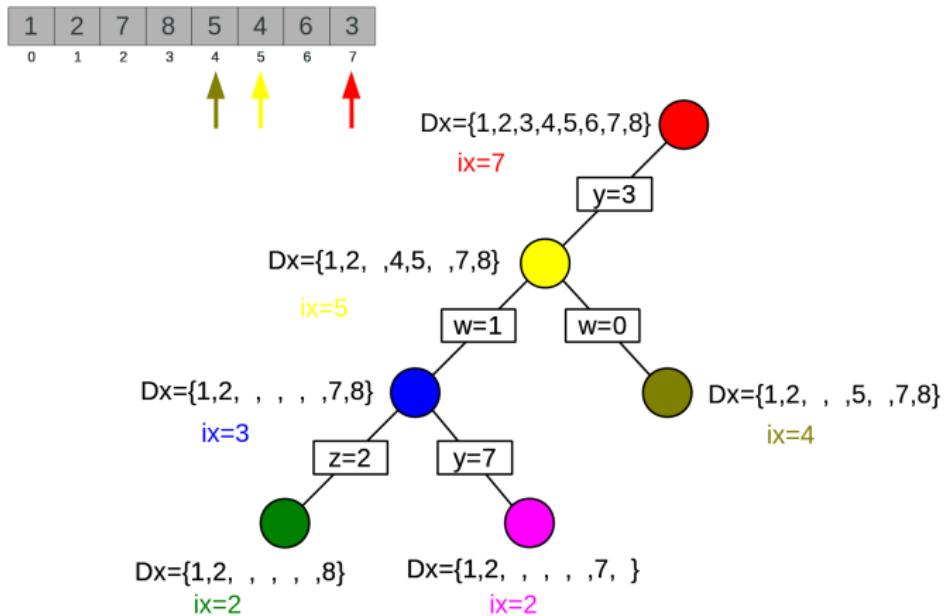
Une proposition en images



Une proposition en images

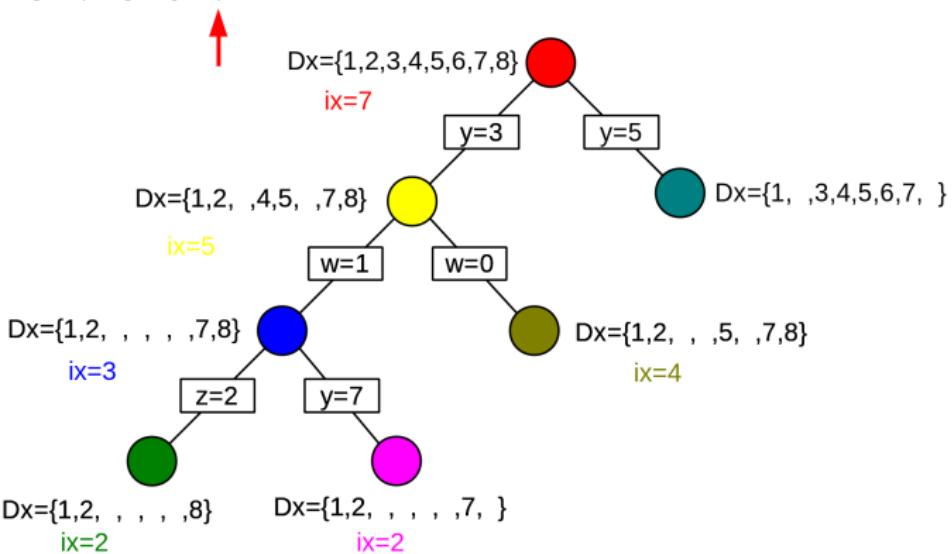


Une proposition en images



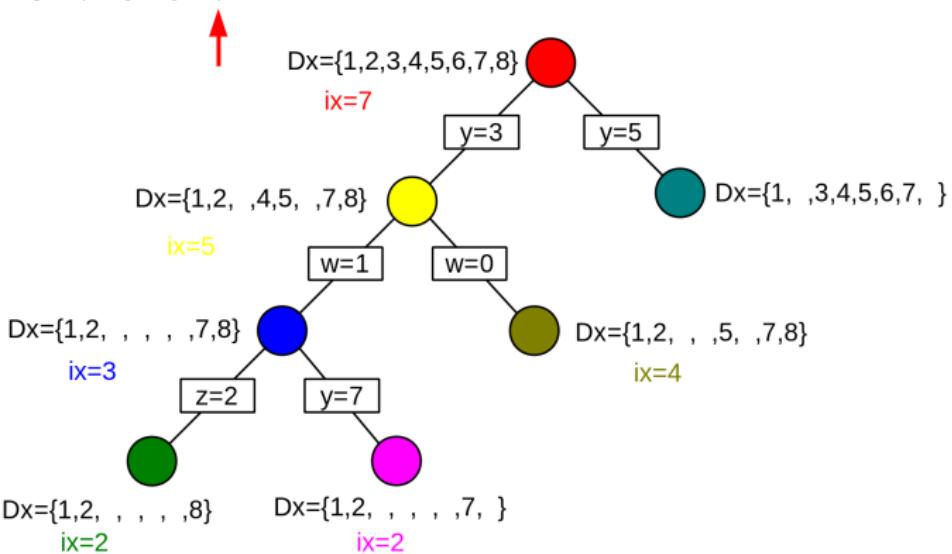
Une proposition en images

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



Une proposition en images

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



Une proposition en images

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

