INTRODUCTION À LA PROGRAMMATION PAR CONTRAINTE ET APPLICATION À L'APPRENTISSAGE D'ARBRES DE DÉCISIONS OPTIMAUX

Séminaire ULaval

Hélène Verhaeghe

8 Avril 2022

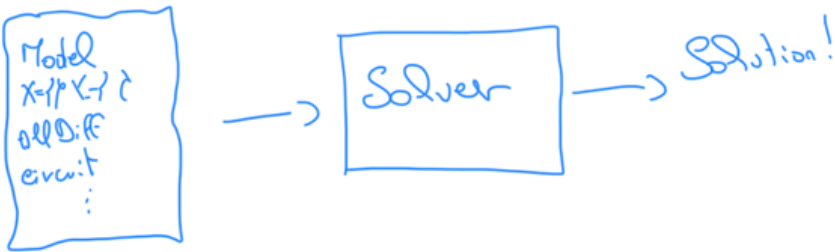
PROGRAMMATION PAR CONTRAINTES

Qu'est-ce que la programmation par contrainte?

"En informatique, de toutes les approches en programmation, la programmation par contraintes se rapproche le plus de l'idéal : l'utilisateur décrit le problème, l'ordinateur le résout." — E. Freuder

Qu'est-ce que la programmation par contrainte?

En programmation par contrainte, on modélise de manière déclarative la solution souhaitée, l'ordinateur/le solveur trouve la solution.



MODÉLISATION

$$CP = Modle + Recherche$$

- **Modèle**: Représentation déclarative de la solution (ce que l'on veut)
- **Recherche**: Description de l'arbre de recherche, de l'heuristique (comment le trouver)

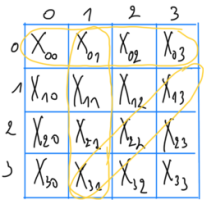
Le modèle contient :

- des **variables**, avec leurs valeurs possibles (domaines) : généralement des entiers, mais il y a aussi des solveurs avec des variables de type floats, graph, set,...

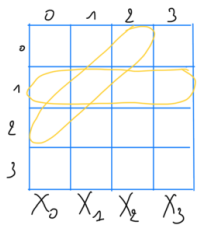
$$X \in \{1, 2, 5, 9\} \qquad Y \in \{red, blue, yellow, ...\}$$

- des **contraintes**, qui lient les valeurs des variables entre elles : contraintes arithmétiques ($X + Y = Z$), contraintes logiques ($X < Y$), contraintes combinatoires ($\texttt{circuit}(x_1, ..., x_n)$),...
- un **objectif** (quand il y en a un) : $min/max\ f(X)$

- Variables : X_{ij} représentant si l'emplacement (i, j) contient une reine, $X_{ij} \in \{0, 1\}$
- Contraintes :
  - $\texttt{Sum}(X \in ligne_i) = 1$, pour éviter d'avoir deux reines sur la même ligne
  - $\texttt{Sum}(X \in colone_i) = 1$, pour éviter d'avoir deux reines sur la même colonne
  - $\texttt{Sum}(X \in diagonale_k) \leq 1$, pour éviter d'avoir deux reines sur la même diagonale



- Variables : X_i représentant l'emplacement de la reine de la colone i, $X_i \in \{1, 2, ..., N\}$
- Contraintes :
  - `AllDifferent`$(X \in ligne_i)$, pour éviter d'avoir deux reines sur la même ligne
  - `AllDifferent`$(X \in diag_i)$, pour éviter d'avoir deux reines sur la même diagonale



```
object Queens extends CPModel with App {
  val nQueens = 10 // Number of queens
  val Queens = 0 until nQueens
  // Variables
  val queens = Array.fill(nQueens)(CPIntVar.sparse(0, nQueens - 1))
  // Constraints
  add(allDifferent(queens))
  add(allDifferent(Queens.map(i => queens(i) + i)))
  add(allDifferent(Queens.map(i => queens(i) - i)))
  // Search heuristic
  search(binaryFirstFail(queens))
  // Execution
  val stats = start()
  println(stats)
}
```

(OscaR model)

Ces deux modèles donnent les même solutions.

Cependant, utiliser moins de variable et moins de contrainte est généralement une bonne chose (réduire la taille de l'arbre de recherche).

On peut aussi tomber dans le cas où les deux modèles sont utiles en combinaison l'un avec l'autre. On aura alors des contraintes de **channeling** pour lier les valeur des variables d'un des modèles avec les variables de l'autre modèle.

$$X_i = j \leftrightarrow X_{ij} = 1$$

Une contrainte peut s'exprimer également de deux manières différente. Ajouter les deux formulation au modèle crée des contraintes dites **redondantes**.

Utiliser du channeling ou des contraintes redondantes est à double tranchant. Des fois cela peut aider, des fois cela va ralentir le solveur.

On parle de symétrie quand à partir d'une solution, on sait en déduire d'autres.

Dans le cas du N-reines, nous avons les symétries suivantes:

- rotation du plateau de 90°, 180° et 270°
- symétrie verticale et horizontale

Ajout de $X_0 < X_{N-1}$ par exemple pour casser la symétrie verticale et horizontale

CONSTRAINTES

Une contrainte est généralement décomposable en plusieurs éléments:

- Un `état` : comprends les structures pour éviter de répéter certain calculs,...
- Une `initialisation` : initiale vérification avant même l'exploration de l'arbre de recherche
- Un algorithme de `propagation` : algorithme pour retirer les valeurs imposibles du domaine

But d'une contrainte : Retirer les valeurs qui ne mènent pas à des solutions au fur et à mesure de la recherche

$$X > Y$$

Propagation:

$min(X) = \min\{min(X), min(Y) + 1\}$

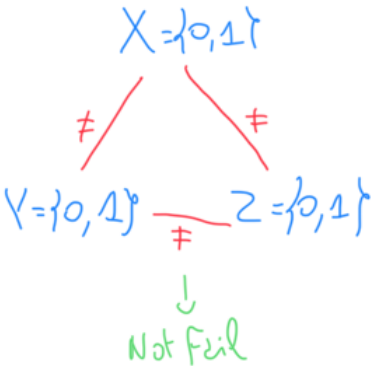
$max(Y) = \max\{max(Y), max(X) - 1\}$

$$\texttt{AllDifferent}(X, Y, Z)$$

ou

$$X \neq Y, X \neq Z \text{ et } Y \neq Z$$

Quelle est la différence entre ces deux formulations?



AllDifferent(X={0,1}, Y={0,1}, Z={0,1})

2 valeurs
vs
3 variables

Fail

RECHERCHE

L'arbre de recherche est composé de noeuds et de feuilles.

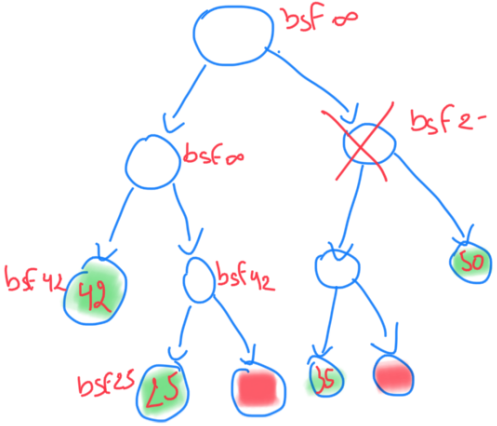
- **noeud**: exécution du point-fixe, sauvegarde des états si nécessaire pour revenir en arrière, prise de décision pour continuer la recherche
- **feuille**: atteindre une feuille signifie qu'on a assigné toutes les variables sans violer les contraintes, nous avons donc une solution

Une décision dans un noeud choisit une variable encore non-assignée et réduit son domaine

- soit en choisissant une valeur et en testant l'assignation et le retrait de cette valeur ($X = v$ et $X \neq v$)
- soit en testant toutes ses valeurs ($X = v_1$, $X = v_2$, ... et $X = v_n$)
- soit en découpant le domaine en deux ($X \leq v$ et $X > v$)

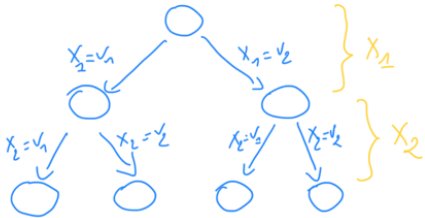
A chaque solution trouvée, une nouvelle contrainte est ajoutée

$$objectif < best - so - far$$



Caractéristiques :

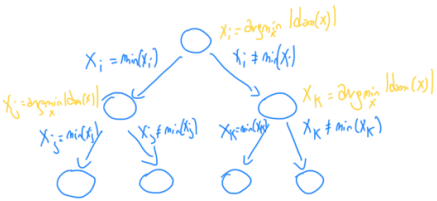
- Ordre de variable fixe
- Ordre de valeur fixe
- Arbre déterministe



Positif : Exécution déterministe
Négatif : Très peu efficace en pratique
Utilité : Comparer la force de propagation de deux algorithmes

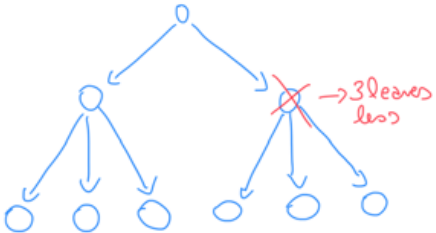
Caractéristiques :

- Ordre de variable dynamique
- Variable avec le domaine le plus petit
- But: trouver une solution rapidement, taille minimale théorique (si il n'y a pas de propagation) de l'arbre de recherche

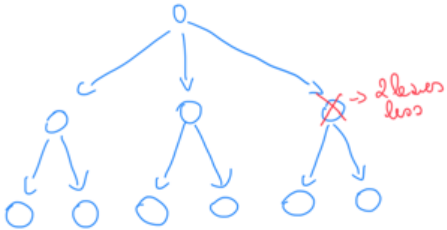


Positif : Arbre minimal théorique
Négatif : Ne tiens pas compte du nombre de contraintes qui utilise les variables
Utilité : Trouver vite une première solution

Min-Dom: (3 noeuds et 6 feuilles potentielles)



Max-Dom: (4 noeuds et 6 feuilles potentielles)



Min-Dom permet d'avoir un plus petit arbre au départ. De plus, lorsqu'il y a élagage d'une branche, plus de feuilles sont retirées d'un coup.

ARBRES DE DÉCISION AVEC LA PPC

POLYTECHNIQUE MONTRÉAL UCLouvain

”Learning Optimal Decision Trees Using Constraint Programming”

Hélène Verhaeghe[1,2], Siegfried Nijssen[1], Claude-Guy Quimper[3], Gilles Pesant[2], Pierre Schaus[1]

[1] ICTEAM, UCLouvain, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium, $\{firstname.lastname\}@uclouvain.be$

[2] Polytechnique Montréal, Montréal, Canada, $\{gilles.pesant, helene.verhaeghe\}@polymtl.ca$

[3] Université Laval, Québec, Canada, $claude-guy.quimper@ift.ulaval.ca$

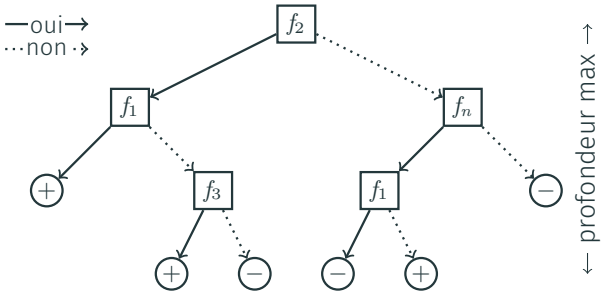
CP 2019 25 YEARS

Base de donnée

f_1	f_2	f_3	$\dots$	f_n	c
1	0	1	$\dots$	1	$+$
0	1	0	$\dots$	1	$-$
1	1	0	$\dots$	0	$+$
0	0	0	$\dots$	0	$+$
1	0	0	$\dots$	0	$+$
0	1	1	$\dots$	1	$-$
1	1	1	$\dots$	0	$-$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1	1	1	$\dots$	1	$+$

Base de donnée

f_1	f_2	f_3	$\dots$	f_n	c
1	0	1	$\dots$	1	+
0	1	0	$\dots$	1	−
1	1	0	$\dots$	0	+
0	0	0	$\dots$	0	+
1	0	0	$\dots$	0	+
0	1	1	$\dots$	1	−
1	1	1	$\dots$	0	−
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1	1	1	$\dots$	1	+



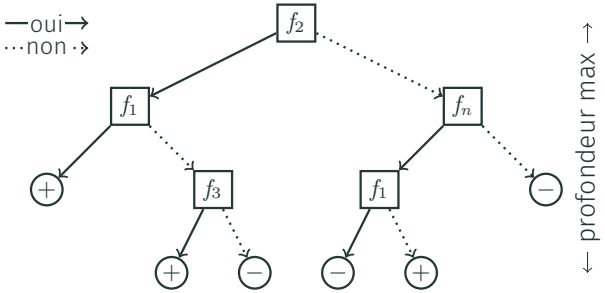
$$\min \sum (pred(i) - c(i))$$

Base de donnée

f_1	f_2	f_3	$\dots$	f_n	c
1	0	1	$\dots$	1	+
0	1	0	$\dots$	1	−
1	1	0	$\dots$	0	+
0	0	0	$\dots$	0	+
1	0	0	$\dots$	0	+
0	1	1	$\dots$	1	−
1	1	1	$\dots$	0	−
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1	1	1	$\dots$	1	+

Nouvelle entrée

0	0	1	$\dots$	0	?



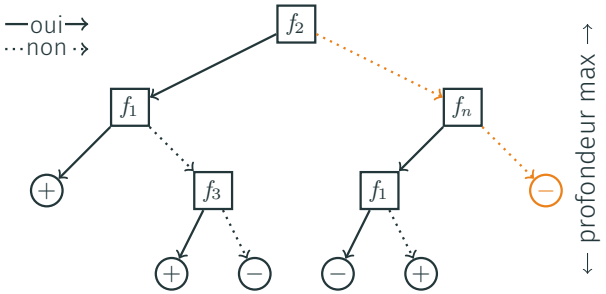
$$\min \sum (pred(i) - c(i))$$

Base de donnée

f_1	f_2	f_3	$\dots$	f_n	c
1	0	1	$\dots$	1	+
0	1	0	$\dots$	1	−
1	1	0	$\dots$	0	+
0	0	0	$\dots$	0	+
1	0	0	$\dots$	0	+
0	1	1	$\dots$	1	−
1	1	1	$\dots$	0	−
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1	1	1	$\dots$	1	+

Nouvelle entrée

0	0	1	$\dots$	0	−



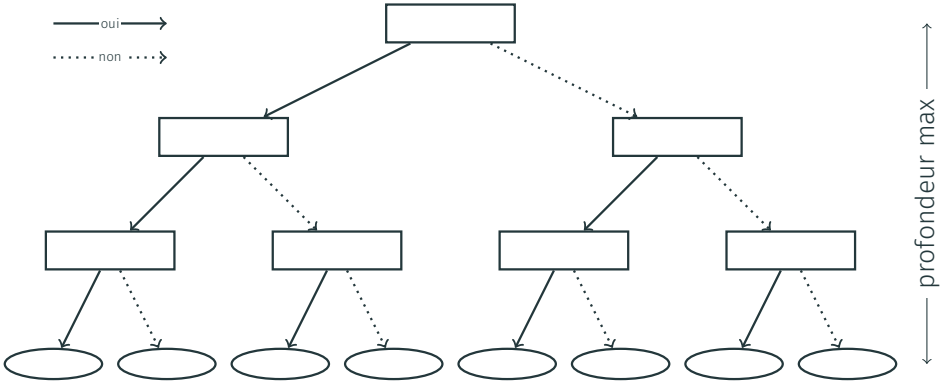
$$\min \sum (pred(i) - c(i))$$

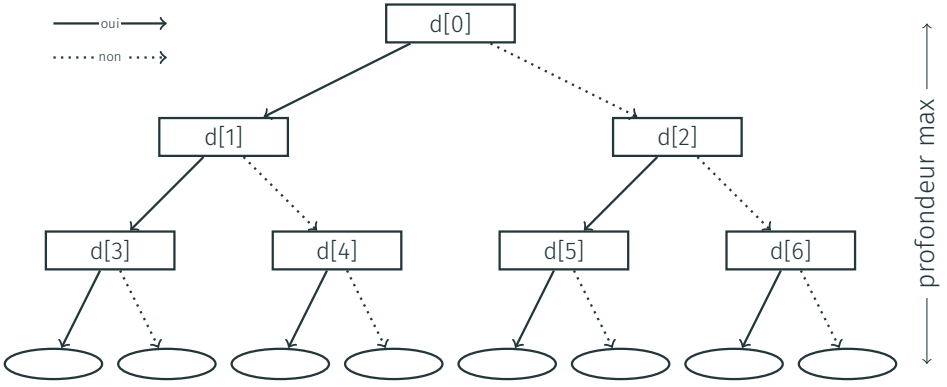
Méthodes gloutonnes :

- ✔ construction facile
- ✗ difficulté d'imposer des contraintes additionnelles
- ✗ arbre potentiellement inutilement complexe

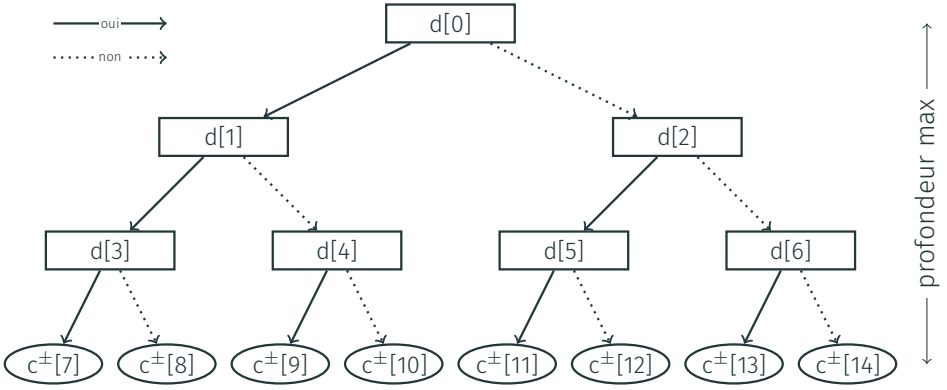
- Mining optimal decision trees from itemset lattices, Nijssen, S., Fromont, E., 2007
- Minimising decision tree size as combinatorial optimisation, Bessiere, C., Hebrard, E., O'Sullivan, B., 2009
- Optimal constraint-based decision tree induction from itemset lattices,Nijssen, S., Fromont, É., 2010
- **Optimal classification trees**, Bertsimas, D., Dunn, J., 2017
- Learning optimal decision trees with sat, Narodytska, N., Ignatiev, A., Pereira, F., Marques-Silva, J., RAS, I., 2018
- Learning optimal and fair decision trees for non-discriminative decision-making, Aghaei, S., Azizi, M.J., Vayanos, P., 2019
- Learning optimal classification trees using a binary linear program formulation, Verwer, S., Zhang, Y., 2019

MODÈLE PPC


oui
non
profondeur max

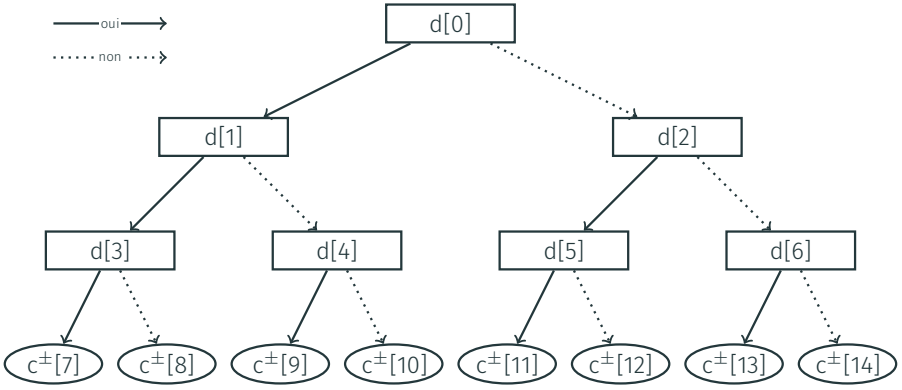


$$dom(d[i]) = \{1, ..., n\}$$



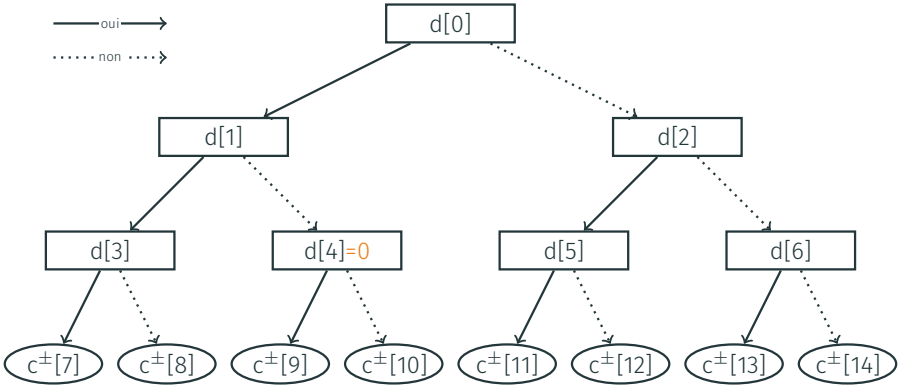
$dom(d[i]) = \{1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



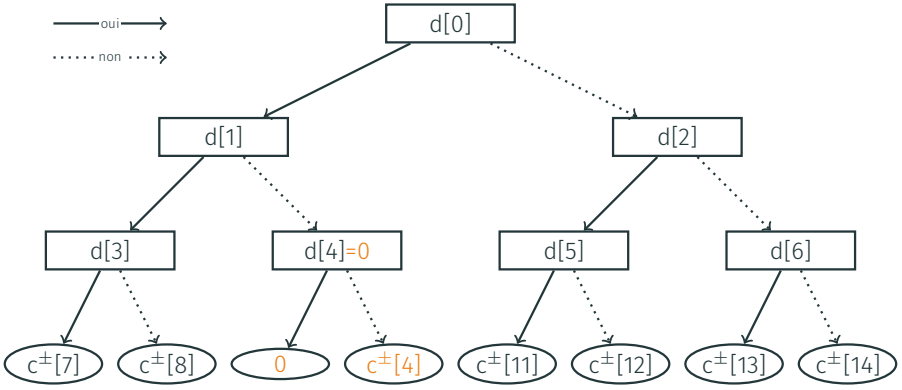
$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



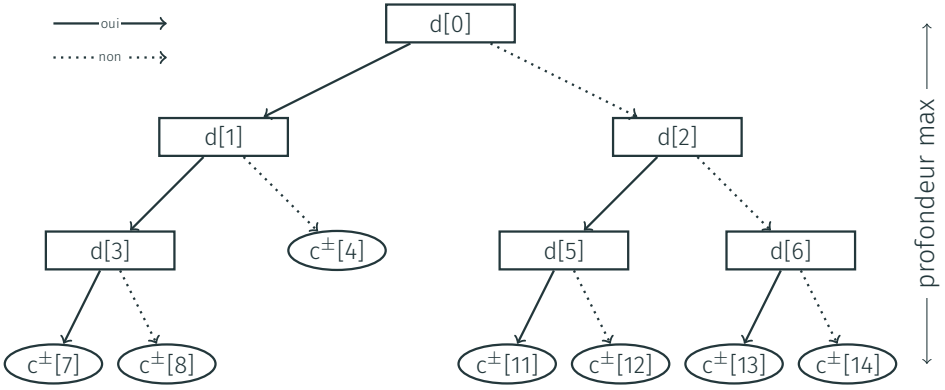
$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



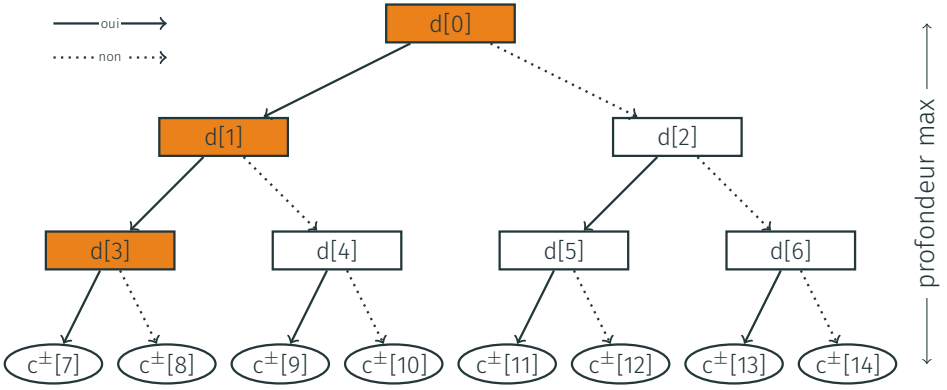
$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



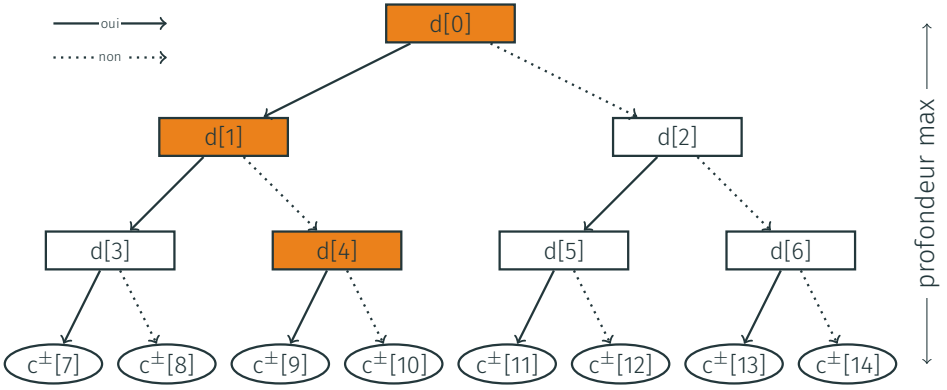
$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



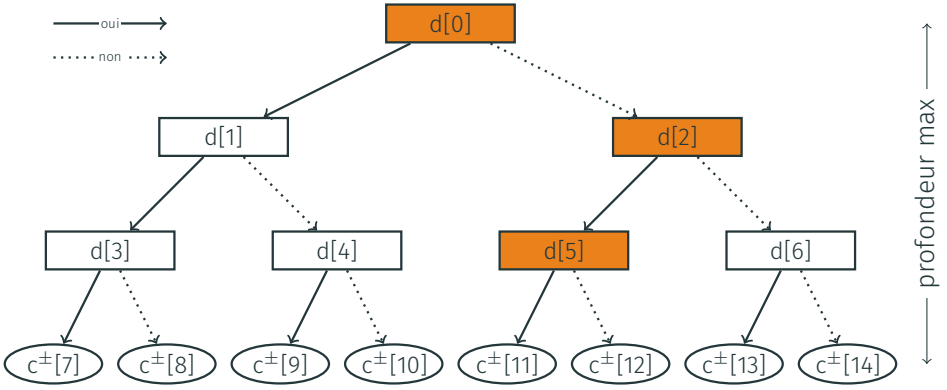
$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



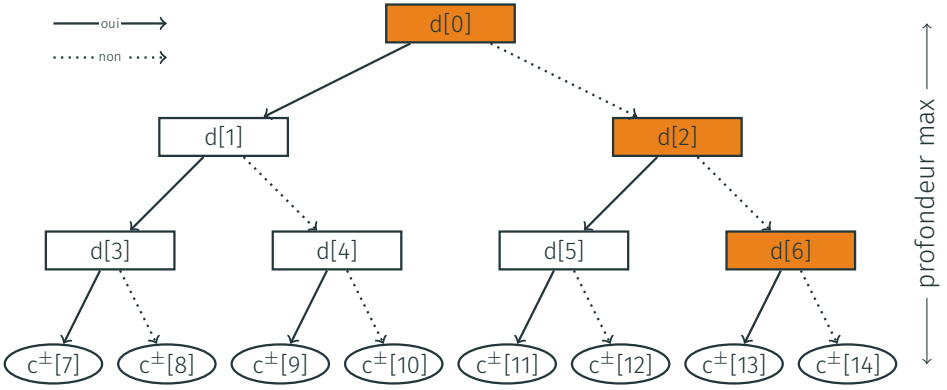
$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$

f_1	f_2	f_3	f_4
1	0	1	1
0	1	0	1
1	1	0	0
0	0	0	0
1	0	0	0
0	1	1	1
1	1	1	0
1	1	1	1

Caractéristiques (Dense)				Compteur
x_1	x_2	x_3	x_4	

f_1	f_2	f_3	f_4
1	0	1	1
0	1	0	1
1	1	0	0
0	0	0	0
1	0	0	0
0	1	1	1
1	1	1	0
1	1	1	1

Caractéristiques (Dense)				Compteur
x_1	x_2	x_3	x_4	
0	1	0	1	

f_1	f_2	f_3	f_4
1	0	1	1
0	1	0	1
1	1	0	0
0	0	0	0
1	0	0	0
0	1	1	1
1	1	1	0
1	1	1	1

Caractéristiques (Dense)				Compteur
x_1	x_2	x_3	x_4	
0	1	0	1	3

f_1	f_2	f_3	f_4
1	0	1	1
0	1	0	1
1	1	0	0
0	0	0	0
1	0	0	0
0	1	1	1
1	1	1	0
1	1	1	1

Caractéristiques (Dense)				Compteur
x_1	x_2	x_3	x_4	
0	1	0	1	3

- Représentation dense
- Pas de rejet de caractéristiques

f_1	f_2	f_3	f_4
1	0	1	1
0	1	0	1
1	1	0	0
0	0	0	0
1	0	0	0
0	1	1	1
1	1	1	0
1	1	1	1

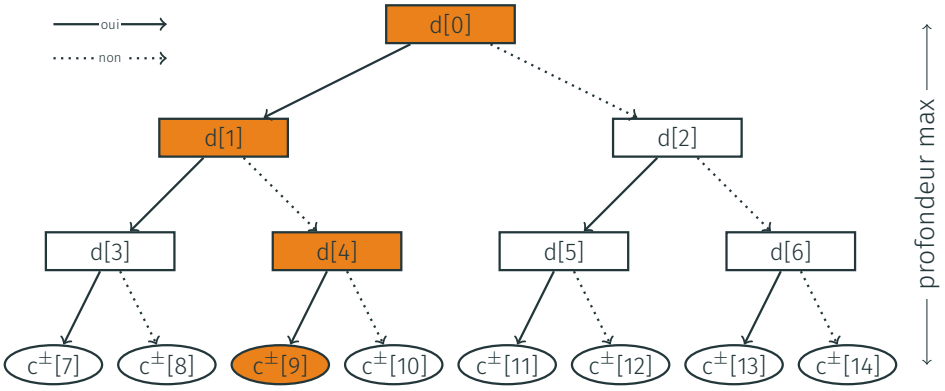
Caractéristiques (Clairsemé)		Compteur
y_1	y_2	
2	4	2

- ~~Représentation dense~~
- Pas de rejet de caractéristiques

f_1	f_2	f_3	f_4
1	0	1	1
0	1	0	1
1	1	0	0
0	0	0	0
1	0	0	0
0	1	1	1
1	1	1	0
1	1	1	1

✔Caractéristiques (Clairsemé)		✗Caractéristiques (Clairsemé)	Counter
y_1	y_2	z_1	
2	4	3	1

- ~~Représentation dense~~
- ~~Pas de rejet de caractéristiques~~



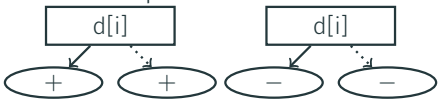
$Coversize(\{d[0], d[4]\}, \{d[1]\}, c^{+}[9])$

$Coversize(\{d[0], d[4]\}, \{d[1]\}, c^{-}[9])$

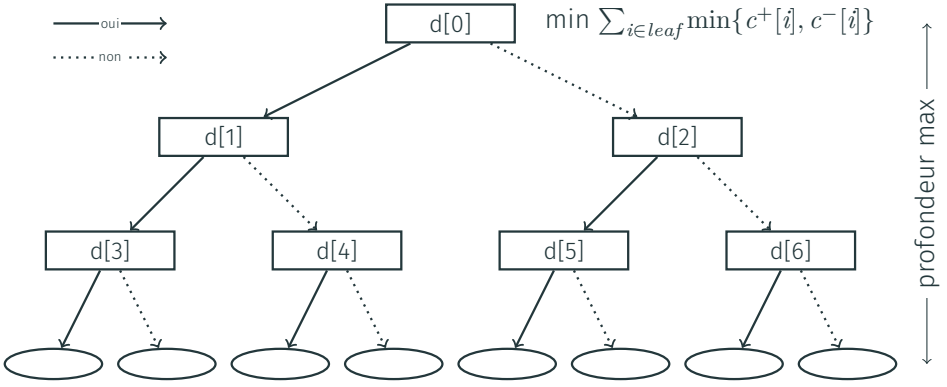
- contraintes pour imposer un minimum aux feuilles

$$c^+[i] + c^-[i] \geq N_{min}$$

- contraintes pour éviter des décisions inutiles

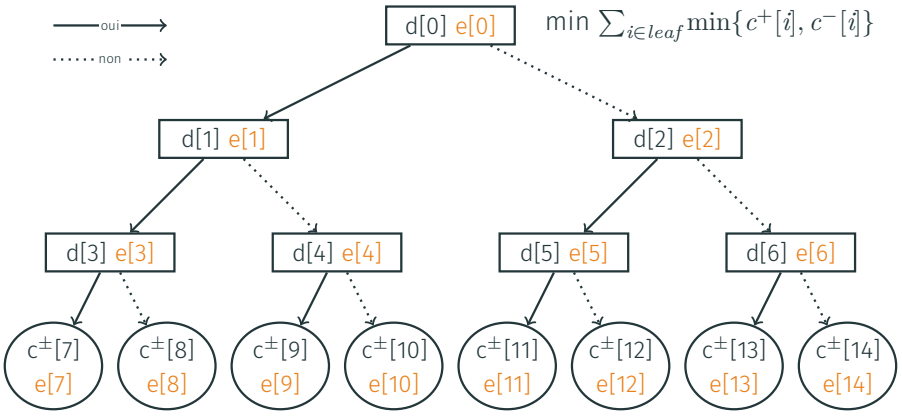


- contraintes redondantes pour améliorer la vitesse

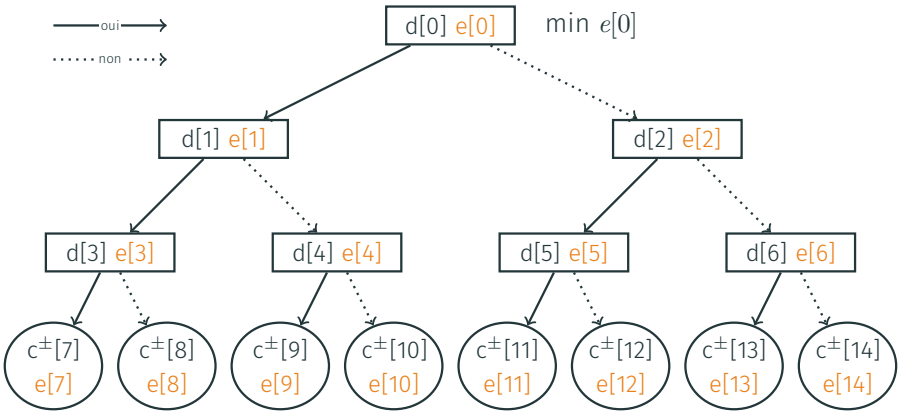


$dom(d[i]) = \{0, 1, ..., n\}$

$dom(c[i]) = \{0, ..., N\}$



$dom(d[i]) = \{0, 1, ..., n\}$ $dom(c[i]) = \{0, ..., N\}$ $dom(e[i]) = \{0, ..., N\}$



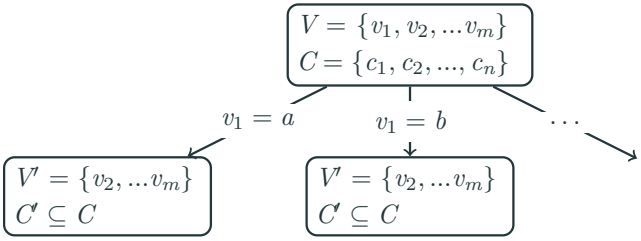
$dom(d[i]) = \{0, 1, ..., n\}$ $dom(c[i]) = \{0, ..., N\}$ $dom(e[i]) = \{0, ..., N\}$

RECHERCHE

$$V = \{v_1, v_2, ...v_m\}$$
$$C = \{c_1, c_2, ..., c_n\}$$

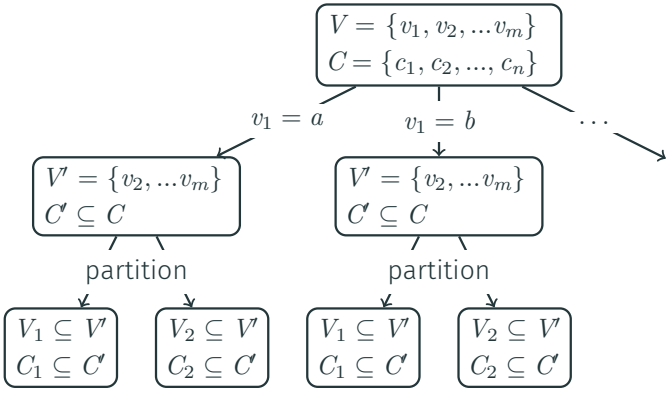
noeuds OU

SOL = SOL_1 ou SOL_2 ou ...



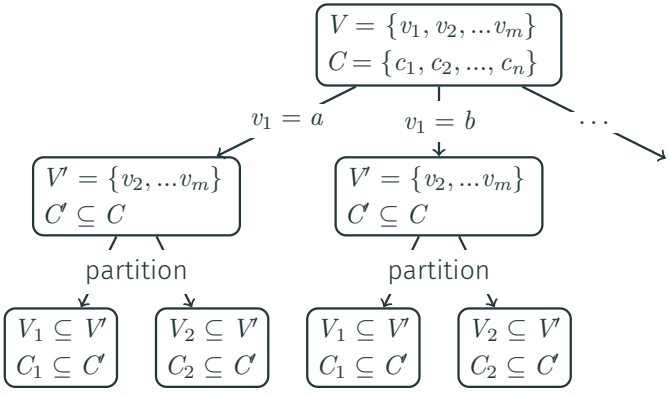
noeuds OU

SOL = SOL_1 ou SOL_2 ou ...



noeuds OU

SOL = SOL_1 ou SOL_2 ou ...

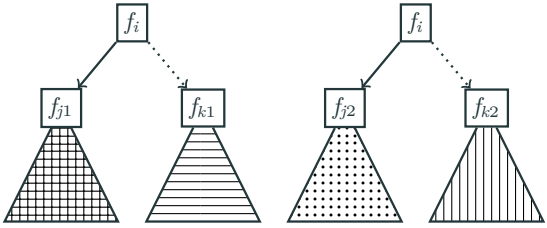


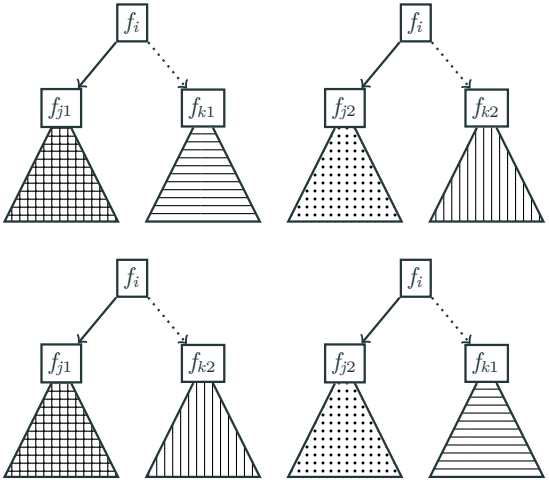
noeuds OU

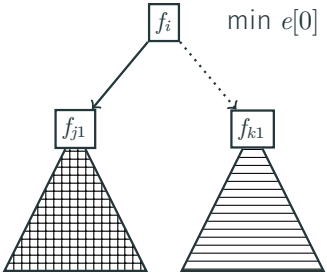
SOL = SOL_1 ou SOL_2 ou ...

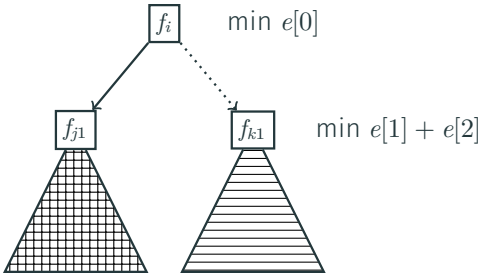
noeuds ET

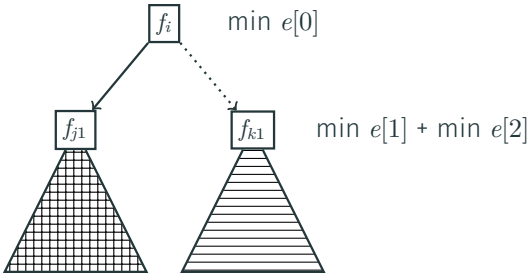
SOL = SOL_1 et SOL_2 et ...

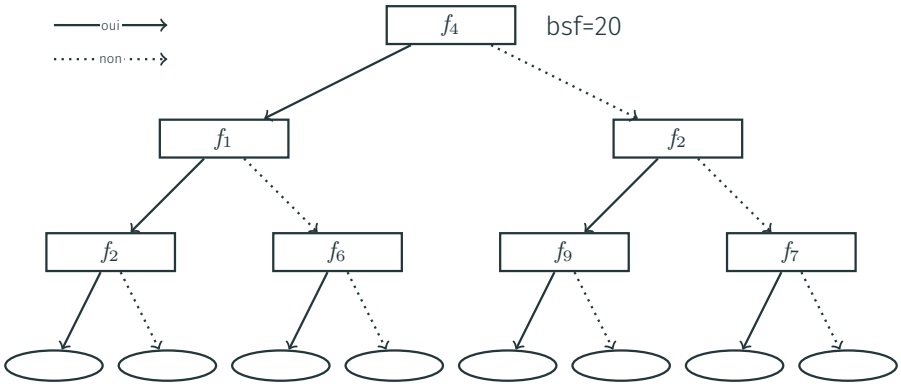

f_i
f_{j1}
f_{k1}
f_i
f_{j2}
f_{k2}

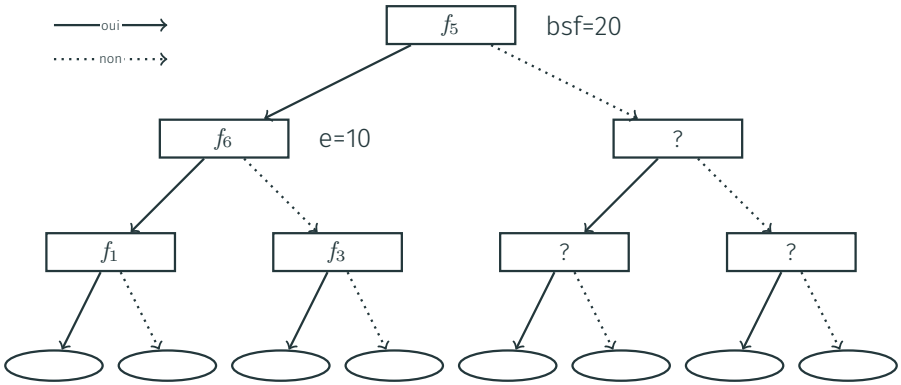

f_i
f_{j1}
f_{k1}
f_i
f_{j2}
f_{k2}
f_i
f_{j1}
f_{k2}
f_i
f_{j2}
f_{k1}

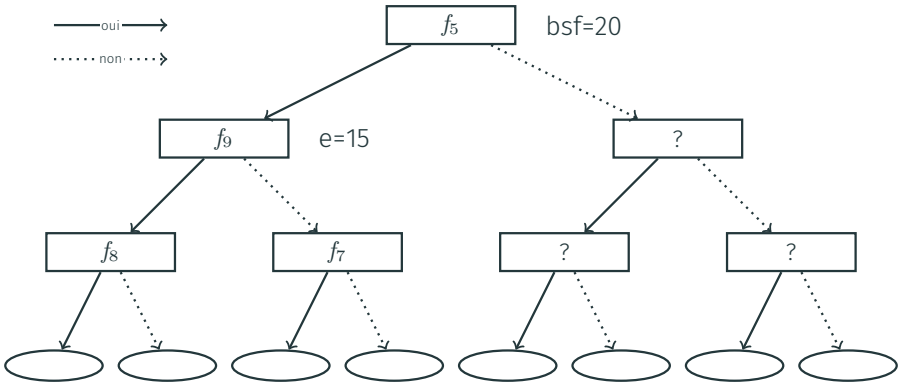

f_i
min $e[0]$
f_{j1}
f_{k1}

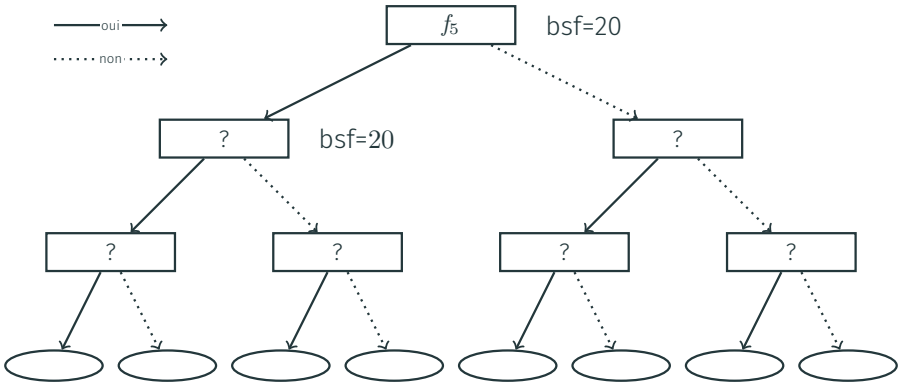

f_i
min $e[0]$
f_{j1}
f_{k1}
min $e[1] + e[2]$

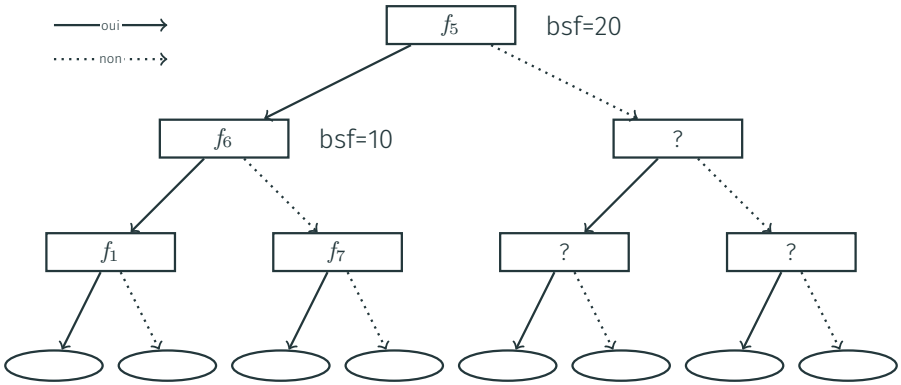

f_i
min $e[0]$
f_{j1}
f_{k1}
min $e[1]$ + min $e[2]$

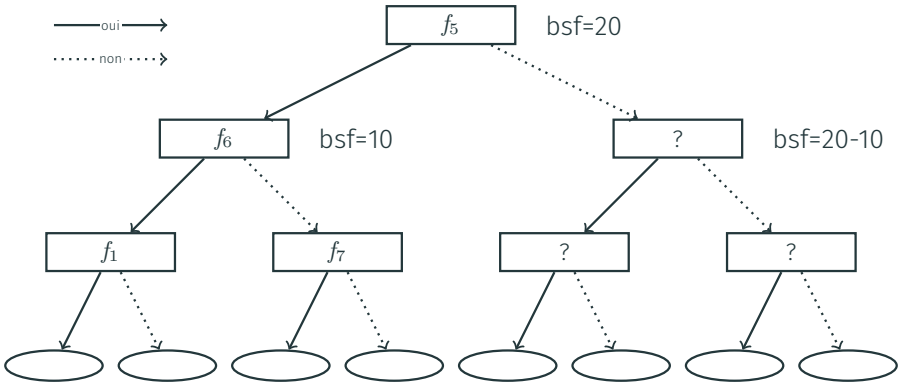

oui
non
f_4
bsf=20
f_1
f_2
f_2
f_6
f_9
f_7

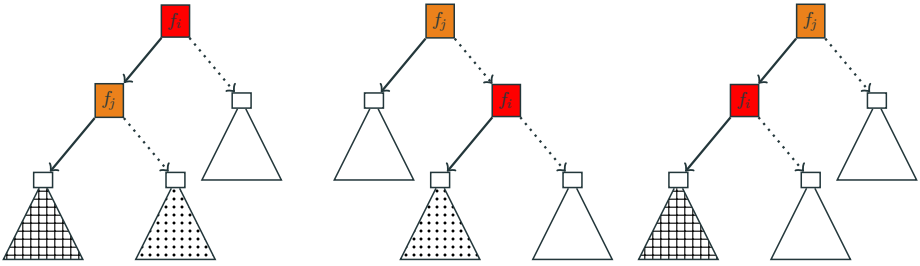

oui
non
f_5
bsf=20
f_6
e=10
?
f_1
f_3
?
?

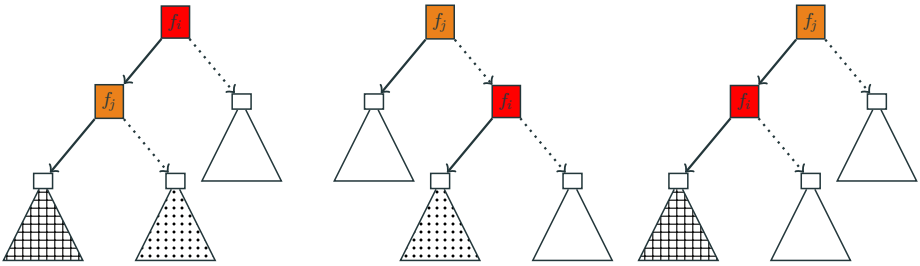

oui
non
f_5
bsf=20
f_9
e=15
?
f_8
f_7
?
?


oui
non
f_5
bsf=20
?
bsf=20
?
?
?
?
?


oui
non
f_5
bsf=20
f_6
bsf=10
?
f_1
f_7
?
?


oui
non
f_5
bsf=20
f_6
bsf=10
?
bsf=20-10
f_1
f_7
?
?


f_j
f_j
f_j
f_j
f_j
f_j



	oui	non	hash
	f_i f_j		f_i, f_j-
	f_i	f_j	$f_i - f_j$

RÉSULTATS

	$N_{\min} = 1$			$N_{\min} = 5$			
	DL8	BinOCT	CP	DL8	CP	CP-c	CP-m
Optimalité prouvée	49(64%)	13(17%)	**57**(75%)	54(71%)	56(74%)	56(74%)	**58**(76%)
Meilleur solution trouvée	49(64%)	21(28%)	**76**(100%)	54(71%)	**74**(97%)	**74**(97%)	70(92%)
Le plus rapide	23(30%)	11(14%)	**49**(64%)	28(37%)	**40**(53%)	33(43%)	22(29%)
Dépasse le temps alloué	27(36%)	63(83%)	**19**(25%)	22(29%)	21(28%)	21(28%)	**19**(25%)

23 instances, profondeurs de 2 à 5, 10 min limite de temps

DL8: Approche en programmation dynamique basée sur le minage d'ensemble d'objets fréquents

BinOCT: Approche basée sur un modèle MIP qui tourne avec CPLEX

Pour résumer

- méthode éfficace
- basée sur la programmation par contraintes
- exploite la structure du problème
- meilleur solution jusque là

Pour aller plus loin

- arbre de décisions multi-classes
- caractéristiques continues (via binarisation)
- autre fonctions de coût basées sur une somme
- ...

Merci pour votre attention!

Des questions?