Algorithmique et Complexité

5. Stratégie III : Backtracking

Nicole Bidoit

Université Paris XI, Orsay

Année Universitaire 2008–2009

Backtracking : quels problèmes ?

Problème de satisfaction de contrainte (CSP)

problèmes exprimant des conditions contraignant le temps, l'espace, les ressources, ...

Exemples :

planification / ordonnancement	production manufacturée	trafic ferroviaire	
affectation de ressources	emploi du temps	système exploitation	appariements
problèmes d'optimisation	placements financiers	routages réseaux	découpes

Forme général d'un CSP

n variables $x_1 ... x_n$, et pour chaque variable x_i un domaine D_i (ens. de valeurs possibles pour x_i)

m contraintes C_i exprimées sous forme de relations entre les variables

Une **solution** du CSP $\vec{X}, \vec{D}, \vec{C}$ est une affectation ”variable $\longleftarrow$ valeur” valide pour $\vec{C}$ (qui satisfait $\vec{C}$).

Autres problèmes cibles :

Jeux (beaucoup de jeux se rapprochent ou sont des CSP)

Recherche d'un chemin de ”coût minimal” dans un graphe

Implémentation de langages de programmation: Icon, Planner et Prolog, text parsing.

Principe du Backtracking

alternative à la recherche exhaustive de solutions

méthode / technique systématique de parcours de l'espace de recherche

$\hookrightarrow$ **éliminer l'examen de certaines configurations** i.e.

$\hookrightarrow$ **élagage de l'espace de recherche**

À chaque étape de l'algorithme de backtracking, on a une solution partielle $s = (v_1 ... v_k)$;

on essaie d'étendre cette solution :

- une affectation v_{k+1} pour x_{k+1} est choisie
- si cette affectation produit une solution partielle, ... étape suivante

  sinon on teste si il y a une autre extension potentielle de s

    - si il y a une autre extension, ... étape suivante

      sinon défaire l'affectation v_k pour x_k.

Résolution d’une grille de Sudoku

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

6	7	3	8	9	4	5	1	2
9	1	2	7	3	5	4	8	6
8	4	5	6	1	2	9	7	3
7	9	8	2	6	1	3	5	4
5	2	6	4	7	3	8	9	1
1	3	4	5	8	9	2	6	7
4	6	9	1	2	8	7	3	5
2	8	7	3	5	6	1	4	9
3	5	1	9	4	7	6	2	8

$\hookrightarrow$ Recherche exhaustive

déjà vu (planche TD 1)

$\hookrightarrow$ **Backtracking**

Exploiter les règles comme des contraintes pour explorer moins de configurations

Résolution naïve – exhaustive d'un problème de type CSP

```
Algo GenTest
```

Entrée : $\vec{X}, \vec{D}, \vec{C}$: un CSP de taille n;

A une affectation partielle de taille $k \leq n$

Sortie : B booleen — % vrai si A peut être étendue en une solution, faux sinon

---

$B \leftarrow false$;

if $k = n$ **then** — % l'affectation est totale

  if tester(A,k,$\vec{C}$) **then** $B \leftarrow true$

else — % affectation partielle

  $k \leftarrow k{+}1$ — % extension de l'affection A

  while $\neg B$ and "nouveau" v dans D_k **do**

    choisir un "nouveau" v dans D_k;

    $A \leftarrow Ajout(A, v, k)$; — % par une affectation pour x_{k+1}

    $B \leftarrow$ GenTest($\vec{X}, \vec{D}, \vec{C}$,$A$, k)

  endwhile

endelse;

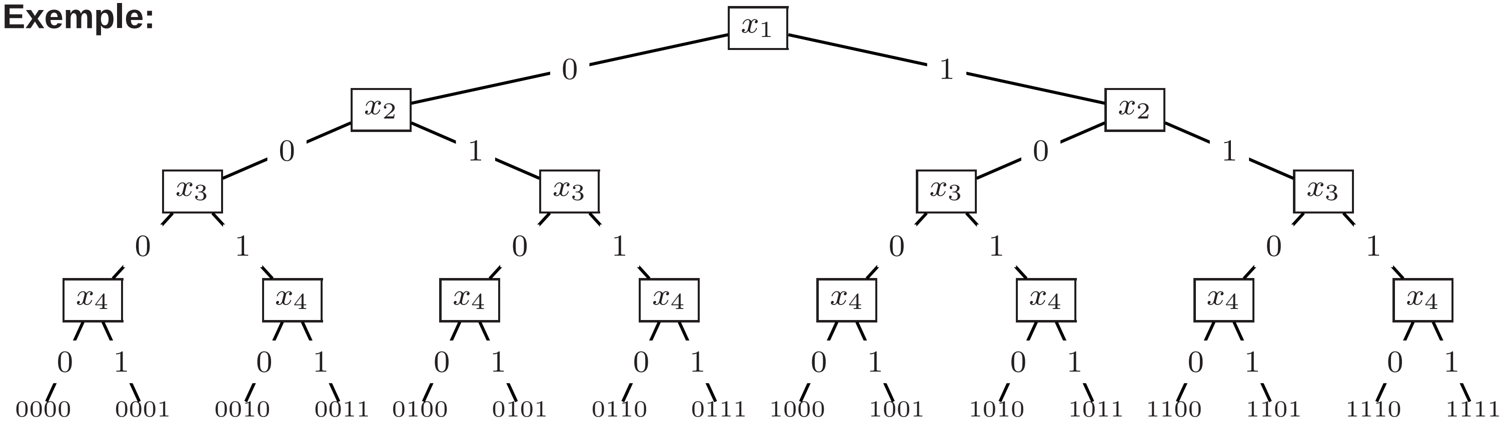
On génére les affectations totales une par une jusqu'à obtention d'une solution.

Exemple (fait en cours): Simuler l'algorithme pour le problème suivant:

$$n = 4 \qquad D_i = \{0, 1\} \text{ pour } i = 1..4, \qquad C = \{x_1 \neq x_2,\ x_3 \neq x_4,\ x_1{+}x_3 < x_2\}$$

Espace de recherche

Exemple:



L'**espace de recherche** $\mathcal{E}$ du problème $(\vec{X}, \vec{D}, \vec{C})$ est

$$\{(v_1, ..., v_n) \mid v_i \in D_i \ pour\ i = 1..n\}$$

et donc la taille de cet espace de recherce est (en supposant que $\mid D_i \mid= d$ pour $i = 1..n$)

$$\mid \mathcal{E} \mid= d^n$$

$\hookrightarrow$ **croissance exponentielle en fonction du nombre de variables**

$n = 10$ $\mid \mathcal{E} \mid\approx 10^3$ (10^{-6} sec.) **...**

$n = 70$ $\mid \mathcal{E} \mid\approx 10^{21}$ (317 siècles)

Exemple : L'espace de recherche $\mathcal{E}$ est donné par les feuilles de l'arbre de décision.

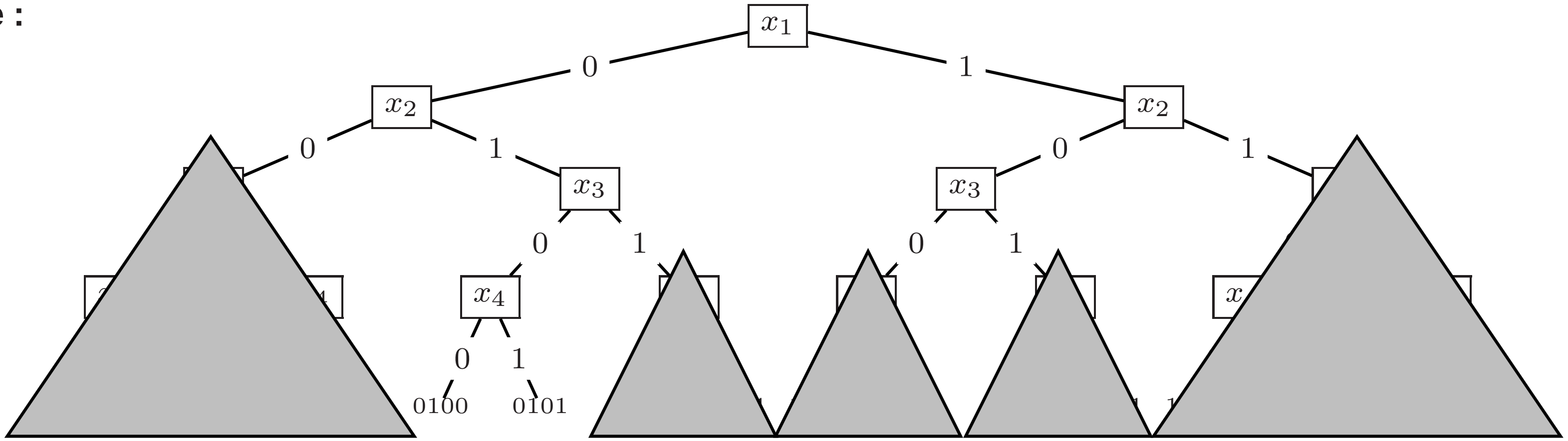
$$\mid \mathcal{E} \mid= 2^4 = 16$$

$\hookrightarrow$ **Étendre uniquement les affectations partielles qui "satisfont" les contraintes**

On teste les affectations partielles; dès qu'une contrainte est invalide, on arrête d'étendre l'affectation.

La génération de l'affectation et le test des contraintes sont entremélés

Exemple :



Taille de l'espace de recherche réduit (environ) de 16 à 2

$\hookrightarrow$ coût = tests ! mais c'est gagnant

```
Algo Backtrack-Rec
```

Entrée : $\vec{X}, \vec{D}, \vec{C}$: CSP de taille n;

A une affectation partielle de taille $k \leq n$

Sortie : B booleen % vrai si A peut être étendue en une solution, faux sinon

---

$B \leftarrow false$;

if testpartiel(A,k,$\vec{C}$) **then** % A est une solution partielle

  if $k = n$ **then** $B \leftarrow true$ **else** % l'affectation est totale donc c'est une solution

    $k \leftarrow k+1$ % extension de l'affection A

    while $\neg B$ and "nouveau" v dans D_k **do**

      choisir un "nouveau" v dans D_k;

      $A \leftarrow Ajout(A, v, k)$; % par une affectation pour x_{k+1}

      $B \leftarrow$ Backtrack-Rec($\vec{X}, \vec{D}, \vec{C}$,$A$, k)

    endwhile;

  endelse;

endthen

Exemple (fait en cours): Simuler l'algorithme de backtracking simple pour le problème suivant:

$n = 4 \qquad D_i = \{0, 1\}$ pour $i = 1..4, \qquad C = \{x_1 \neq x_2,\ x_3 \neq x_4,\ x_1+x_3 < x_2\}$

Backtracking simple

Exercice : Raffiner (choisir des structures de données et préciser les fonctions `Testpartiel`, ...) l'algorithme de backtracking simple ci-dessus de manière à pouvoir traiter le problème CSP de l'exemple du cours.

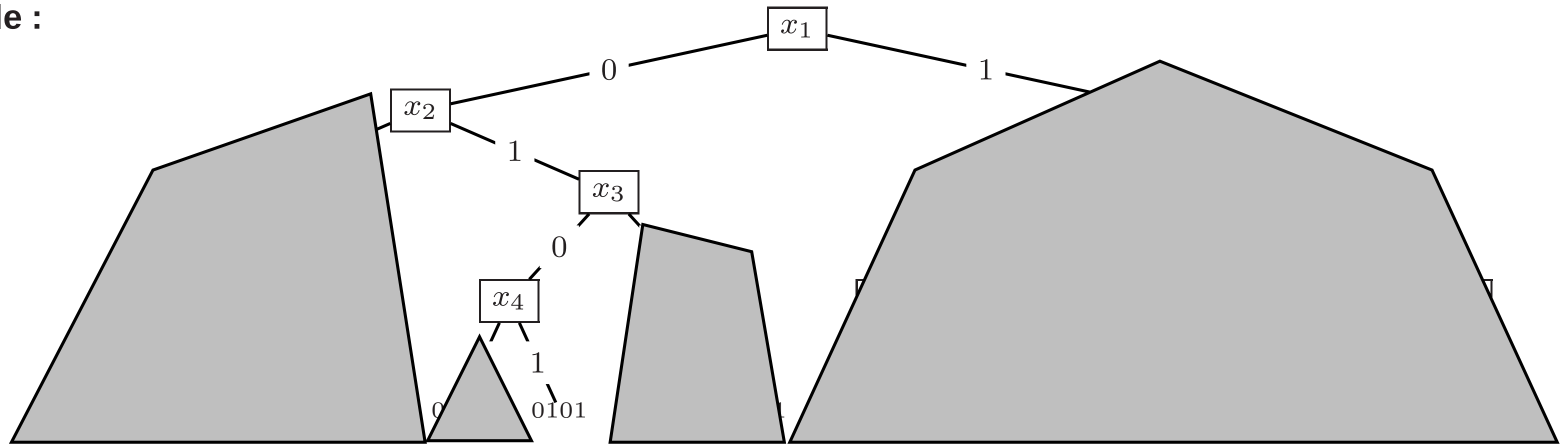
Exercice : Modifier l'algorithme de backtracking simple de manière à obtenir toutes les solutions d'un problème.

$\hookrightarrow$ **Réduire la taille des domaines des variables**

Pendant la génération des affectations partielles, on filtre les domaines des variables restant à affecter pour ne garder que les valeurs localement valides.

Les tests sont anticipés (look ahead).

Exemple :



L'affectation $[x_1 \leftarrow 0]$ élimine immédiatement la valeur 0 de D_2 pour x_2 (contrainte $x_1 \neq x_2$)

L'affectation $[x_1 \leftarrow 1]$ élimine immédiatement les valeurs 0 et 1 de D_2 pour x_2 (contrainte $x_1 + x_3 < x_2$)

L'affectation $[x_2 \leftarrow 1]$ élimine immédiatement les valeurs 0 et 1 de D_3 pour x_3 (contrainte $x_1 + x_3 < x_2$)

L'affectation $[x_3 \leftarrow 0]$ élimine immédiatement la valeur 0 de D_4 pour x_4 (contrainte $x_3 \neq x_4$)

```
Algo Look-Ahead-Rec
```

Entrée : $\vec{X}, \vec{D}, \vec{C}$: CSP de taille n;

A une affectation partielle de taille $k \leq n$

Sortie : B booleen % vrai si A peut être étendue en une solution, faux sinon

---

$B \leftarrow false$;

if $k = n$ **then** $B \leftarrow true$ **else** % l'affectation est totale donc c'est une solution

$k \leftarrow k+1$; % extension de l'affection A par une affectation pour x_{k+1}

while $\neg B$ and "nouveau" v dans D_k **do**

choisir un "nouveau" v dans D_k;

$A \leftarrow Ajout(A, v, k)$; $vide \leftarrow false$; $j \leftarrow k+1$;

while $j \leq n$ and $\neg vide$ **do**

$DF_j \leftarrow$ Filtrer($\vec{C}, D_j, A, k, j$); % filtrage des domaines des variables non affectées

if $DF_j = \emptyset$ **then** $vide \leftarrow true$ **else** $j \leftarrow j+1$;

endwhile

if $\neg vide$ then $B \leftarrow$ Look-Ahead-Rec($\vec{X}, \vec{DF}, \vec{C}$,$A$, k)

endwhile;

endelse;

Exemple (fait en cours): Simuler l'algorithme de backtracking avec filtrage pour le problème suivant:

$$n = 4 \qquad D_i = \{0, 1\} \text{ pour } i = 1..4, \qquad C = \{x_1 \neq x_2,\ x_3 \neq x_4,\ x_1+x_3 < x_2\}$$

Réduire l'espace de recherche – Élagage de l'arbre de décision

$\hookrightarrow$ **Utiliser des heuristiques pour guider la recherche, etc ...**

Essayer d'explorer d'abord certaines affectations en s'appuyant sur des critères dépendant du domaine et qui, "si ça marche" conduit "vite" vers une solution.

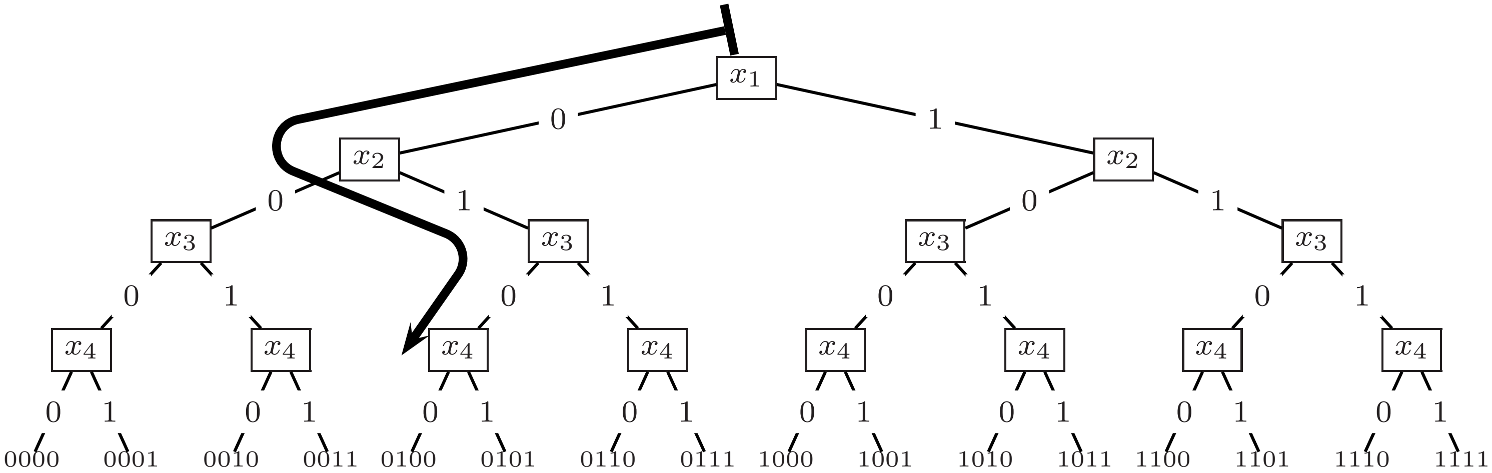
Exemple : (un peu simple pour illustrer les heuristiques)

choisir une valeur de x_1 petite car x_1 à gauche de $<$ dans la contrainte $x_1 + x_3 < x_2$

choisir une valeur de x_2 grande car x_2 à droite de $<$ dans la contrainte $x_1 + x_3 < x_2$

choisir une valeur de x_3 petite car x_3 à gauche de $<$ dans la contrainte $x_1 + x_3 < x_2$

Exemple :



Ordre d'examen des variables quelques pistes ”classiques”

$\hookrightarrow$ considérer les variables **critiques** en premier

x_i critique ?

(1) x_i apparaît dans *beaucoup* de contraintes et/ou

(2) x_i est très sélective (le domaine D_i est *petit*)

Le choix de l'ordre d'examen des variables peut être

statique ie déterminé a priori (critère 1 par exemple)

dynamique ie fait à chaque étape (critère 2 dans le cas du back-tracking avec filtrage)

Ordre d'affectation des valeurs dépendant de l'application, non généralisable

```
Algo Backalg-Rec
 Entrée : ...
 Sortie : ...
```

$B \leftarrow false;$

...

$k \leftarrow k+1$ % extension de l'affection A

while $\neg B$ and ”nouveau” v dans D_k **do**

choisir un ”nouveau” v dans D_k;

...

Liste de problèmes

Problème 1 : construction de tous les sous-ensembles

Problème 2 : construction d’un dérangement (une permutation d’une liste L telle qu’aucun éleément n’est à sa place initiale)

Problème 3 : les 8 reines

Une rapide comparaison des stratégies