Backtracking

(a.k.a. résoudre un Sudoku)

Michaël Rao

CNRS - ENS Lyon
LIP - Laboratoire de l'Informatique du Parallélisme
équipe MC2

De nombreuses questions/conjectures en combinatoire :

Pour tout $x \in X$, on a $P(x)$

où X est un ensemble (dénombrable) et P est une propriété.
(Par exemple, pour le 4CT : X= graphes planaires, P=4-colorable))

De nombreuses questions/conjectures en combinatoire :

Pour tout $x \in X$, on a $P(x)$

où X est un ensemble (dénombrable) et P est une propriété.
(Par exemple, pour le 4CT : X= graphes planaires, P=4-colorable))

L'ordinateur peut nous servir (de façon triviale) :

- Essayer sur de nombreux $x \in X$ et se faire une intuition
- Si X est fini et elle est vraie : prouver la conjecture
- Si la conjecture est fausse : trouver un contre exemple

De nombreuses questions/conjectures en combinatoire :

Pour tout $x \in X$, on a $P(x)$

où X est un ensemble (dénombrable) et P est une propriété.
(Par exemple, pour le 4CT : X= graphes planaires, P=4-colorable))

L'ordinateur peut nous servir (de façon triviale) :

- Essayer sur de nombreux $x \in X$ et se faire une intuition
- Si X est fini et elle est vraie : prouver la conjecture
- Si la conjecture est fausse : trouver un contre exemple

Si la conjecture est vraie et X est infini, il faudra "retravailler" le problème...

Attaquer avec un ordinateur...

Première idée : parcourir X, et tester P sur chaque élément.
Une solution est un $x \in X$ t.q. $P(x)$

Attaquer avec un ordinateur...

Première idée : parcourir X, et tester P sur chaque élément.
Une solution est un $x \in X$ t.q. $P(x)$

Backtracking (ou "Retour sur trace") :

Essayer toutes les possibilités, en faisant des suppositions, et des retours en arrière.

$\Rightarrow$ explorer "l'arbre" des possibilités

Attaquer avec un ordinateur...

Première idée : parcourir X, et tester P sur chaque élément.
Une solution est un $x \in X$ t.q. $P(x)$

Backtracking (ou "Retour sur trace") :

Essayer toutes les possibilités, en faisant des suppositions, et des retours en arrière.

$\Rightarrow$ explorer "l'arbre" des possibilités

Si X est grand ou infini, il faut parcourir intelligemment X, c.à.d. éviter des sous ensembles où on sait qu'il n'y aura pas de solution

On peut se servir de P pour guider la recherche.

Problème des Dames

Soit un damier $n \times n$, est il possible de placer n dames qui ne peuvent pas s'attaquer 2 à 2 ?
(n est un entier fixé)

X : l'ensemble des placement de n dames sur $n \times n$.
Taille : $\binom{n^2}{n}$

Problème des Dames

Soit un damier $n \times n$, est il possible de placer n dames qui ne peuvent pas s'attaquer 2 à 2 ?
(n est un entier fixé)

X : l'ensemble des placement de n dames sur $n \times n$.
Taille : $\binom{n^2}{n}$

Remarque : une dame par ligne.
X : $[1..n]^n$ Taille : n^n

Problème des Dames

Soit un damier $n \times n$, est il possible de placer n dames qui ne peuvent pas s'attaquer 2 à 2 ?
(n est un entier fixé)

X : l'ensemble des placement de n dames sur $n \times n$.
Taille : $\binom{n^2}{n}$

Remarque : une dame par ligne.
X : $[1..n]^n$ Taille : n^n

Remarque 2 : une dame par ligne et colonne
X : $\mathbb{S}_n$ Taille : $n!$

Dames : premier algorithme

- pour toute permutation $x \in \mathbb{S}_n$
  - tester si $\forall i, j$ avec $i \neq j$, on a :
    - $x_i \neq x_j$
    - $x_i + (i - j) \neq x_j$
    - $x_i + (j - i) \neq x_j$

Dames : premier algorithme

- pour toute permutation $x \in \mathbb{S}_n$
  - tester si $\forall i, j$ avec $i \neq j$, on a :
    - $x_i \neq x_j$
    - $x_i + (i - j) \neq x_j$
    - $x_i + (j - i) \neq x_j$

Temps : $n!$ essais $\times$ tests en n^2

Dames : premier algorithme

- pour toute permutation $x \in \mathbb{S}_n$
  - tester si $\forall i, j$ avec $i \neq j$, on a :
    - $x_i \neq x_j$
    - $x_i + (i - j) \neq x_j$
    - $x_i + (j - i) \neq x_j$

Temps : $n!$ essais $\times$ tests en n^2

C'est bête... Par exemple on pourrait écarter toutes les permutations avec $x_1 = 1$ et $x_2 = 2$...

Dames : premier algorithme

- pour toute permutation $x \in \mathbb{S}_n$
  - tester si $\forall i, j$ avec $i \neq j$, on a :
    - $x_i \neq x_j$
    - $x_i + (i - j) \neq x_j$
    - $x_i + (j - i) \neq x_j$

Temps : $n!$ essais $\times$ tests en n^2

C'est bête... Par exemple on pourrait écarter toutes les permutations avec $x_1 = 1$ et $x_2 = 2$...
[IRL]

Sudoku

Voir mieux ce qui se passe : Sudoku

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

remplir chaque case restante avec des entiers entre 1 et 9, tel qu'il n'y ait pas deux mêmes entiers sur chaque ligne, colonne, ou carré 3×3

Sudoku

Grille partiellement remplie = fonction partielle de $[1..9]^2 \to [1..9]$

Backtrack(T)

- Entrée : une grille T partiellement remplie
- Si la grille est complète : gagné !
- Sinon
  - prendre une case vide c dans T
  - pour chaque entier $x \in [1..9]$, Backtrack($T|_{c \to x}$)

Sudoku

Grille partiellement remplie = fonction partielle de $[1..9]^2 \to [1..9]$

Backtrack(T)

- Entrée : une grille T partiellement remplie
- Si la grille est complète : gagné !
- Sinon
  - prendre une case vide c dans T
  - pour chaque entier $x \in [1..9]$, Backtrack($T|_{c \to x}$)

[IRL]

Où brancher ?

Pour le moment, notre algorithme est stupide...

Où brancher ?

Pour le moment, notre algorithme est stupide...

Comment l'humain résout un Sudoku ?
Il commence par remplir les cases où il n'y a qu'une possibilité.

$\Rightarrow$ brancher où il y a le moins de possibilités

Où brancher ?

Pour le moment, notre algorithme est stupide...

Comment l'humain résout un Sudoku ?
Il commence par remplir les cases où il n'y a qu'une possibilité.

$\Rightarrow$ brancher où il y a le moins de possibilités

Mais d'autres heuristiques sont possibles, et peuvent dépendre du problème.

CSP

Le backtraking fonctionne bien pour les problèmes de satisfaction de contraintes (CSP)

Fameux cas particulier des CSP : SAT

Une solution au lieu de programmer soi-même son backtrack : transformer le problème en un problème SAT ou CSP, et le donner à un solveur

Exemple de solveur SAT : glucose