Backtracking

Lycée Thiers

Crédits

1. Un cours de Solène Mirliaz ;

Crédits

1. Un cours de Solène Mirliaz ;
2. Cette page de Wikipedia

Problème d'exploration

Définition

Un *problème d'exploration* (ou de *recherche*) est un problème algorithmique $\mathcal{P}$ donné par une relation binaire $\mathcal{R} \subset E \times (S \sqcup \{\textbf{None}\})$ telle qu'il existe deux sous-ensembles notés E^+, E^- de E formant une partition de E et vérifiant :

$$\mathcal{R} \subset (E^+ \times S) \sqcup (E^- \times \{\textbf{None}\})$$

Remarque

- Les éléments de E sont appelés des *entrées* et ceux de S des *solutions* ;

Problème d'exploration

Définition

Un *problème d'exploration* (ou de *recherche*) est un problème algorithmique $\mathcal{P}$ donné par une relation binaire $\mathcal{R} \subset E \times (S \sqcup \{\textbf{None}\})$ telle qu'il existe deux sous-ensembles notés E^+, E^- de E formant une partition de E et vérifiant :

$$\mathcal{R} \subset (E^+ \times S) \sqcup (E^- \times \{\textbf{None}\})$$

Remarque

- Les éléments de E sont appelés des *entrées* et ceux de S des *solutions* ;
- Le fait que l'union soit disjointe interdit d'avoir une entrée e et une solution s telles que $(e, s) \in \mathcal{R}$ et $(e, \textbf{None}) \in \mathcal{R}$.

Vocabulaire : solution

Avec les notations de la définition précédente :

- si $(e, s) \in E \times S$, on dit que s est une *solution* de l'*entrée* e ;

Vocabulaire : solution

Avec les notations de la définition précédente :

- si $(e, s) \in E \times S$, on dit que s est une *solution* de l'*entrée* e ;
- si $(e, \textbf{Node}) \in \mathcal{R}$, on dit que l'entrée e *n'a pas de solution*.

Backtracking : présentation informelle

- Le *backtracking* (ou *retour sur trace*) est une classe d'algorithmes cherchant la solution de problèmes d'exploration.

Backtracking : présentation informelle

- Le *backtracking* (ou *retour sur trace*) est une classe d'algorithmes cherchant la solution de problèmes d'exploration.
- Notamment les problèmes avec *contraintes de satisfactions* comme le Sudoku où les N-reines sont résolubles par backtracking.

Backtracking : présentation informelle

- Le *backtracking* (ou *retour sur trace*) est une classe d'algorithmes cherchant la solution de problèmes d'exploration.
- Notamment les problèmes avec *contraintes de satisfactions* comme le Sudoku où les N-reines sont résolubles par backtracking.
- Le backtracking construit incrémentalement des *candidats-solutions partiels* qui sont abandonnés dès lors qu'on établit qu'ils ne peuvent pas être complétés en une solution.

Backtracking (suite)

- Conceptuellement, les candidats-solutions partiels sont représentés comme des nœuds d'une structure arborescente : l'*arbre de recherche potentiel*.

Backtracking (suite)

- Conceptuellement, les candidats-solutions partiels sont représentés comme des nœuds d'une structure arborescente : l'*arbre de recherche potentiel*.
- Chaque candidat-solution partiel est le parent d'autres candidats-solutions qui diffèrent de lui par une simple étape d'extension (par exemple ajout d'un élément unique à une liste).

Backtracking (suite)

- Conceptuellement, les candidats-solutions partiels sont représentés comme des nœuds d'une structure arborescente : l'*arbre de recherche potentiel*.
- Chaque candidat-solution partiel est le parent d'autres candidats-solutions qui diffèrent de lui par une simple étape d'extension (par exemple ajout d'un élément unique à une liste).
- Les feuilles sont les candidats-solutions qui ne peuvent pas être développés plus avant (ce sont des *candidats-solutions complets*). Certains candidats-solutions complets sont effectivement des solutions, d'autres non.

Backtracking : arbre de décision

- L'algorithme de backtracking parcourt l'arbre de recherche potentiel par un DFS.

Backtracking : arbre de décision

- L'algorithme de backtracking parcourt l'arbre de recherche potentiel par un DFS.
- A chaque nœud c (donc, un candidat-solution), l'algorithme cherche si c peut être complété en une solution valide.

Backtracking : arbre de décision

- L'algorithme de backtracking parcourt l'arbre de recherche potentiel par un DFS.
- A chaque nœud c (donc, un candidat-solution), l'algorithme cherche si c peut être complété en une solution valide.
  - Si ce n'est pas possible, le sous-arbre de racine c dans l'arbre solution est supprimé.

Backtracking : arbre de décision

- L'algorithme de backtracking parcourt l'arbre de recherche potentiel par un DFS.
- A chaque nœud c (donc, un candidat-solution), l'algorithme cherche si c peut être complété en une solution valide.
  - Si ce n'est pas possible, le sous-arbre de racine c dans l'arbre solution est supprimé.
  - Par ailleurs, si c est une feuille, l'algorithme cherche si c lui-même peut être considéré comme une solution valide (exemple une grille complètement remplie constitue-t-elle une solution au problème de Sudoku ?).

Backtracking : arbre de décision

- L'algorithme de backtracking parcourt l'arbre de recherche potentiel par un DFS.
- A chaque nœud c (donc, un candidat-solution), l'algorithme cherche si c peut être complété en une solution valide.
  - Si ce n'est pas possible, le sous-arbre de racine c dans l'arbre solution est supprimé.
  - Par ailleurs, si c est une feuille, l'algorithme cherche si c lui-même peut être considéré comme une solution valide (exemple une grille complètement remplie constitue-t-elle une solution au problème de Sudoku ?).
- L'arbre effectivement parcouru par le backtracking est un sous-arbre (souvent strict) de l'arbre de recherche potentiel.

Algorithme de Backtracking

Listing 1 – Algo Backtracking

```
1  fonction backtrack(e,c):
2    entree : e entrée du problème; c candidat-solution partiel
3    sortie : une solution de e s'il en existe, None sinon
4    debut
5      si c est un candidat-solution complet (une feuille) alors
6        si c est une solution de e alors renvoyer c
7        sinon renvoyer None;
8      sinon
9        pour tout c' fils candidat-solution POSSIBLE de c faire
10         v ← backtrack(e, c');
11         si v ≠ None alors renvoyer v
12       renvoyer None;
13   fin
```

Remarque

Un candidat-solution *impossible* est un candidat dont on se rend compte qu'il ne peut pas être complété en une solution.
La détection précoce des candidats-solutions impossibles permet de limiter le coût de l'exploration.

Complexité

La complexité d'un algorithme de backtracking de recherche de solution pour une entrée e dépend en particulier de la taille de l'arbre exploré.
On se place dans le cas le pire où aucune détection précoce de candidat impossible n'est détectée.
Il faut alors explorer tout l'arbre de décision.

Complexité

- On note h la hauteur de l'arbre et p, l'arité maximum d'un nœud. L'arbre est alors un sous-arbre de l'arbre parfait d'arité p et de hauteur h : sa taille est $\sum_{i=0}^{h} p^i = O(p^h)$ (h est en fait le nombre d'étapes élémentaires pour construire un candidat-solution complet à partir de rien).

Complexité

- On note h la hauteur de l'arbre et p, l'arité maximum d'un nœud. L'arbre est alors un sous-arbre de l'arbre parfait d'arité p et de hauteur h : sa taille est $\sum_{i=0}^{h} p^i = O(p^h)$ (h est en fait le nombre d'étapes élémentaires pour construire un candidat-solution complet à partir de rien).
- La complexité du test de détection précoce d'impossibilité pour un candidat c et de celui de vérification qu'un candidat complet est une véritable solution dépendent de la taille de c, laquelle peut être majorée par une fonction de $|e|$.
  On peut majorer la complexité de ces tests par une fonction $f(|e|)$ où f dépend du problème étudié.

Complexité

- On note h la hauteur de l'arbre et p, l'arité maximum d'un nœud. L'arbre est alors un sous-arbre de l'arbre parfait d'arité p et de hauteur h : sa taille est $\sum_{i=0}^{h} p^i = O(p^h)$ (h est en fait le nombre d'étapes élémentaires pour construire un candidat-solution complet à partir de rien).
- La complexité du test de détection précoce d'impossibilité pour un candidat c et de celui de vérification qu'un candidat complet est une véritable solution dépendent de la taille de c, laquelle peut être majorée par une fonction de $|e|$.
  On peut majorer la complexité de ces tests par une fonction $f(|e|)$ où f dépend du problème étudié.
- En résumé, le backtracking a une complexité en $O(p^h f(|e|))$ où e est une entrée, h la hauteur de l'arbre de décision et f dépend du problème étudié.