

Partie 2. Techniques algorithmiques

6. Recherche exhaustive

Bruno Grenet

Université Grenoble Alpes – IM²AG

L3 Mathématiques et Informatique

UE Algorithmique

<https://membres-ljk.imag.fr/Bruno.Grenet/Algorithmique.html>

Table des matières

1. Exemple 1 : SAT

2. Principes de la recherche exhaustive

3. Exemple 2 : le voyageur de commerce

Table des matières

1. Exemple 1 : SAT

2. Principes de la recherche exhaustive

3. Exemple 2 : le voyageur de commerce

Le problème SAT

Définition

Entrée : une formule logique φ à n variables booléennes, sous *forme normale conjonctive* (CNF)

Sortie : une affectation des variables qui satisfasse φ ; « insatisfiable » sinon

Formule logique CNF : *conjonction de disjonction de littéraux*

- ▶ **Littéraux :** $x_1, \neg x_1, \dots, x_n, \neg x_n$
- ▶ **Disjonction :** $C = x_1 \vee \neg x_3 \vee \neg x_4$ (clause)
- ▶ **Conjonction :** $C_1 \wedge C_2 \wedge \dots \wedge C_k$

$$\varphi(x_1, x_2, x_3) = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge \neg x_2$$

Affectation satisfaisante ou non

- ▶ $(x_1, x_2, x_3) = (\text{FAUX}, \text{FAUX}, \text{FAUX})$ satisfait φ
- ▶ $(x_1, x_2, x_3) = (\text{VRAI}, \text{FAUX}, \text{VRAI})$ ne satisfait pas φ

SAT : résolution par recherche exhaustive

Algorithme: tester toutes les affectations possibles

Questions

- ▶ Comment parcourir toutes les affectations possibles ?
- ▶ Comment tester si une affectation satisfait la formule ?
- ▶ Quelle est la complexité de cet algorithme ?

Question préalable

- ▶ Quelle représentation informatique pour les formules et les affectations ?

SAT : représentation informatique

Représentation d'une formule CNF

- ▶ Conjonction $C_1 \wedge \dots \wedge C_k \rightarrow$ tableau de clauses
- ▶ Clause $C = \ell_1 \vee \dots \vee \ell_t \rightarrow$ tableau de littéraux
- ▶ Littéral $x_j \rightarrow$ entier i ; $\neg x_j \rightarrow$ entier $-i$

Représentation de φ : tableau de tableaux d'entiers

Exemple

$$\varphi(x_1, x_2, x_3) = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge \neg x_2 \quad \rightarrow \quad \varphi = [[-1, 2], [1, 2, -3], [-2]]$$

Représentation d'une affectation

- ▶ Tableau de booléens [FAUX, VRAI, FAUX]
- ▶ Plus pratique : tableau $\pm 1 \rightarrow$ VRAI = 1 ; FAUX = -1 [-1, 1, -1]

SAT : tester une affectation

Idée de l'algorithme

- ▶ Parcourir toutes les clauses \rightarrow elles doivent toutes être satisfaites
- ▶ Clause satisfaite : (au moins) un littéral est satisfait
- ▶ Littéral satisfait :
 - ▶ Littéral non nié : affectation VRAI $\rightarrow \ell > 0$ et $A_{[\ell-1]} = 1$
 - ▶ Littéral nié : affectation FAUX $\rightarrow \ell < 0$ et $A_{[-\ell-1]} = -1$

SATISFAIT(φ, A) :

1. Pour C dans φ :
2. OK \leftarrow FAUX
3. Pour ℓ dans C :
4. Si $\ell \times A_{[|\ell|-1]} > 0$:
5. OK \leftarrow VRAI
6. Si NON(OK) : Renvoyer FAUX
7. Renvoyer VRAI

Complexité

Linéaire en la taille de φ
= somme des tailles des clauses

SAT : parcourir les affectations

Affectations = mots binaires

- ▶ Affectation : tableau de n valeurs ± 1
- ▶ *Bijection* avec les mots binaires de longueur n : $1 \mapsto 1, -1 \mapsto 0$
- ▶ Analogie : parcourir les affectations \Leftrightarrow compter de 0 à $2^n - 1$
- ▶ Opération nécessaire : AFFSUIVANTE \Leftrightarrow incrémenter un compteur binaire

AFFSUIVANTE(A) :

1. $i \leftarrow 0$
2. Tant que $i < n$ et $A_{[i]} = 1$:
3. $A_{[i]} \leftarrow -1$
4. $i \leftarrow i + 1$
5. Si $i = n$: renvoyer « Fin »
6. $A_{[i]} \leftarrow 1$
7. Renvoyer A

Propriétés

- ▶ Si on part de $[-1, \dots, -1]$, AFFSUIVANTE parcourt toutes les affectations
- ▶ Complexité :
 - ▶ $O(n)$ dans le pire cas
 - ▶ $O(1)$ amortie (chap. 3)

SAT : algorithme de recherche exhaustive

RECHERCHEEXHAUSTIVE(φ) :

1. $A \leftarrow$ tableau de longueur n , initialisé à -1
2. Tant que NON(SATISFAIT(φ, A)):
3. $A \leftarrow$ AFFSUIVANTE(A)
4. Si AFFSUIVANTE a renvoyé « Fin » : Renvoyer « Insatisfiable »
5. Renvoyer A

Propriétés

Correction : conséquence de la correction de SATISFAIT et AFFSUIVANTE

Complexité : nombre d'itérations $\leq 2^n$; coût d'une itération : $O(|\varphi| + n) = O(|\varphi|)$

Remarque : permet d'obtenir toutes les affectations satisfaisantes, ou leur nombre, ...

Théorème

L'algorithme RECHERCHEEXHAUSTIVE trouve une affectation satisfaisante s'il en existe une, et renvoie « Insatisfiable » sinon, en temps $O(|\varphi|2^n)$.

Table des matières

1. Exemple 1 : SAT

2. Principes de la recherche exhaustive

3. Exemple 2 : le voyageur de commerce

Recherche exhaustive

Deux ingrédients

- ▶ Parcourir toutes les solutions possibles
- ▶ Tester chaque solution

En pratique

- ▶ Tester une solution est souvent *facile*
- ▶ Parcourir toutes les solutions peut être complexe

Objectifs

- ▶ Trouver une solution correcte parmi un ensemble de solutions possibles
- ▶ Compter le nombre de solutions, trouver toutes les solutions
- ▶ Trouver la meilleure solution, la plus petite, au contraire la plus grande, ...

Analyse de complexité

$$O(\text{NOMBRESOLUTIONS} \times (\text{COÛTTEST} + \text{COÛTPASSAGESUIVANT}))$$

Ensembles de solutions

Les ensembles de solutions ont souvent une structure mathématique à exploiter

- ▶ Exemple : affectations \leftrightarrow mots binaires
- ▶ Concevoir un algorithme de parcours des solutions demande de :
 - ▶ exhiber la structure mathématique
 - ▶ trouver une façon de parcourir la structure

Quelques exemples de structures

- ▶ Mots binaires, mots k -aires
- ▶ Suites d'entiers, suites *croissantes* d'entiers
- ▶ Sous-ensembles, combinaisons (k parmi n), permutations
- ▶ Arbres binaires, arbres plus généraux
- ▶ ...

(ça peut être difficile !)

Problèmes d'efficacité

La recherche exhaustive est en général exponentielle : soyons efficaces !

Deux façons de produire les solutions

- ▶ Algorithme pour passer d'une solution à la suivante
 - ▶ situation favorable si algorithme efficace
 - ▶ complexité en espace réduite (stockage d'une seule solution)
- ▶ Algorithme pour produire la liste de toutes les solutions, puis parcours
 - ▶ par exemple *via* un algorithme récursif
 - ▶ problèmes de mémoire (ex.: permutations à 12 éléments $\rightarrow > 20\text{Go}$)

Passer rapidement d'une solution à la suivante

- ▶ Ordre optimisé d'énumération des solutions
 - ▶ Réutilisation du test d'une solution pour la suivante
- \rightarrow Questions complexes, au delà de ce cours, évoquées en TD

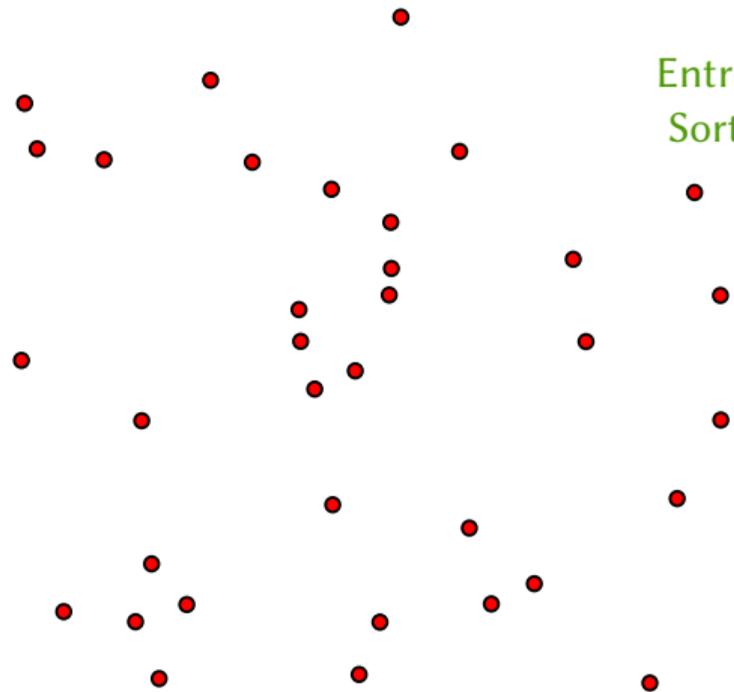
Table des matières

1. Exemple 1 : SAT

2. Principes de la recherche exhaustive

3. Exemple 2 : le voyageur de commerce

Le voyageur de commerce

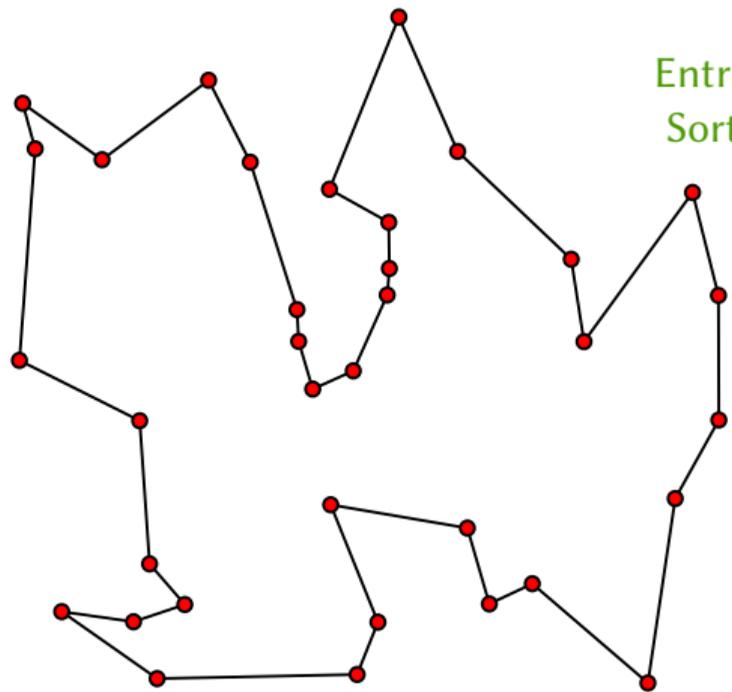


Entrée : Un ensemble de points du plan

Sortie : Un ordre de parcours des points

$u_0 \rightarrow u_1 \rightarrow \dots \rightarrow u_{n-1} \rightarrow u_0$ qui minimise la distance totale

Le voyageur de commerce



Entrée : Un ensemble de points du plan

Sortie : Un ordre de parcours des points

$u_0 \rightarrow u_1 \rightarrow \dots \rightarrow u_{n-1} \rightarrow u_0$ qui minimise la distance totale

Formalisation du problème

Définition

Entrée : Graphe $G = (S, A)$ avec une longueur $\ell(u, v)$ pour chaque arête

Sortie : Une numérotation u_0, \dots, u_{n-1} des sommets qui minimise la longueur totale
$$\sum_{i=0}^{n-1} \ell(u_i, u_{i+1}) + \ell(u_{n-1}, u_0)$$

Remarques

- ▶ Plus général : $\ell(u, v)$ n'est pas forcément une distance
- ▶ Numérotation des sommets = permutation des éléments de S

Algorithme par recherche exhaustive

- ▶ Parcours des solutions : permutations d'un ensemble $\rightarrow \{0, \dots, n-1\}$
- ▶ Test d'une solution : calcul de la longueur totale \rightarrow simple boucle

Générer les permutations d'un ensemble

- ▶ Comment passer d'une permutation à la suivante ?
- ▶ Comment définir « la suivante » ? → ordre sur les permutations

Définitions

- ▶ Permutation de $\{0, \dots, n-1\}$: n -uplets d'entiers tous distincts entre 0 et $n-1$
- ▶ Ordre lexicographique : $\pi^0 < \pi^1$ s'il existe j tq $\pi^0_{[j]} = \pi^1_{[j]}$ pour $i < j$ et $\pi^0_{[j]} < \pi^1_{[j]}$

Exemple : permutations de $\{0, 1, 2, 3\}$ dans l'ordre lexicographique

0123 → 0132 → 0213 → 0231 → 0312 → 0321
→ 1023 → 1032 → 1203 → 1230 → 1302 → 1320
→ 2013 → 2031 → 2103 → 2130 → 2301 → 2310
→ 3012 → 3021 → 3102 → 3120 → 3201 → 3210

Permutations : passer à la suivante

Trois conditions à respecter

1. π' est une permutation : $\pi \rightarrow \pi'$ en échangeant des valeurs
2. $\pi' > \pi$: début de π' égal à π , puis valeur plus grande
3. π' suit π dans l'ordre : début égal à π le plus long possible

Exemple : permutation suivant $\pi = 431520$

Permutations : principe de l'algorithme

Idée de l'algorithme

1. Trouver l'indice maximal j tq $\pi_{[j]} < \pi_{[j+1]}$
 - ▶ j est l'indice *le plus à droite* qu'on peut incrémenter
2. Échanger $\pi_{[j]}$ avec le plus petit $\pi_{[\ell]} > \pi_{[j]}$ pour $\ell > j$
 - ▶ ne pas toucher à $\pi_{[0]}, \dots, \pi_{[j-1]}$
 - ▶ incrément de $\pi_{[j]}$ le plus petit possible
3. *Retourner* la fin $\pi_{[j+1, n[}$
 - ▶ avant retournement : $\pi_{[j+1]} > \pi_{[j+2]} > \dots > \pi_{[n-1]}$
 - ▶ ordre lexicographique commence par $\pi_{[j+1]} < \pi_{[j+2]} < \dots < \pi_{[n-1]}$

Exemple : permutation suivant $\pi = 431520$

Permutations : l'algorithme

PERMSUIVANTE(π) :

0. Si $\pi_{[0]} > \dots > \pi_{[n-1]}$: renvoyer « Fin »
1. $j \leftarrow n - 2$; $\ell \leftarrow n - 1$
2. Tant que $\pi_{[j]} > \pi_{[j+1]}$: $j \leftarrow j - 1$
3. Tant que $\pi_{[\ell]} < \pi_{[j]}$: $\ell \leftarrow \ell - 1$
4. $\pi_{[j]} \leftrightarrow \pi_{[\ell]}$
5. Pour $k = 1$ à $\lfloor \frac{n-1-j}{2} \rfloor$: $\pi_{[j+k]} \leftrightarrow \pi_{[n-k]}$
6. Renvoyer π

$$j \text{ max tq } \pi_{[j]} < \pi_{[j+1]}$$
$$\ell \text{ max tq } \pi_{[\ell]} > \pi_{[j]}$$

Lemme

La complexité de PERMSUIVANTE est $O(n)$

Preuve

Correction de PERMSUIVANTE

Lemme

Si π est une permutation, $\text{PERMSUIVANTE}(\pi)$ renvoie la permutation suivant π dans l'ordre lexicographique (ou « Fin »)

Preuve

Retour au voyageur de commerce

Formalisation du graphe

- ▶ G : matrice $n \times n \rightarrow G_{[i,j]}$ = longueur entre sommets i et j (symétrique)
- ▶ Remarque : si graphe non complet $\rightarrow G_{[i,j]} = +\infty$ si pas d'arête entre i et j

VOYAGEURDECOMMERCE(G) :

1. $\pi \leftarrow$ tableau de taille n , initialisé à $[0, 1, \dots, n - 1]$
2. $L_{\min} \leftarrow +\infty$; $\pi_{\min} \leftarrow \pi$
3. Répéter :
4. $L \leftarrow G[\pi_{[n-1]}, \pi_{[0]}] + \sum_{i=0}^{n-2} G[\pi_{[i]}, \pi_{[i+1]}]$
5. Si $L < L_{\min}$: $(L_{\min}, \pi_{\min}) \leftarrow (L, \pi)$
6. $\pi \leftarrow \text{PERMSUIVANTE}(\pi)$
7. Si PERMSUIVANTE a renvoyé « Fin » : renvoyer π_{\min}

Propriétés

- ▶ Complexité : $O(n \times n!)$
- ▶ Correction : déduite de celle de PERMSUIVANTE

Conclusion sur la recherche exhaustive

Atouts

- ▶ Technique algorithmique conceptuellement simple : on teste toutes les possibilités
- ▶ Analyse de complexité simple : essentiellement le nombre de solutions
- ▶ Parfois le mieux qu'on sache faire !
- ▶ Point de départ d'algorithmes plus sophistiqués (*backtrack*, ...)

Limites

- ▶ Solution algorithmiquement coûteuse (quasiment toujours exponentiel)
- ▶ Écriture en détail et implantations parfois difficiles
- ▶ Problèmes éventuels de mémoire

Pour aller plus loin

- ▶ Techniques d'*élagage* de l'ensemble des solutions (dont *backtrack*)
- ▶ Optimisation du passage d'une solution à la suivante
 - ▶ *Algorithmes d'énumération*

Pour aller plus loin

Backtrack et Branch-and-bound

- ▶ Parcours *récuratif* des solutions, avec test de *solutions partielles* → *arbre des solutions*
 - ▶ Si la solution partielle est *contradictoire* → pas d'exploration des sous-arbres
- ▶ Ex. des problèmes de minimisation (trouver la plus petite solution)
 - ▶ *Backtrack* : arrêt si la solution partielle est déjà trop grande
 - ▶ *Branch-and-bound* : arrêt si la solution partielle ne peut mener qu'à une solution trop grande

Exemple de SAT

- ▶ Parcours récuratif : pour chaque x_i , essayer $x_i = \text{VRAI}$ et $x_i = \text{FAUX}$
- ▶ *Backtrack* : est-ce qu'il existe une clause (déjà) insatisfaite ?

Exemple du Voyageur de Commerce

- ▶ Parcours récuratif : $(n - k)$ possibilités pour $\pi_{[k]}$, si $\pi_{[0]}, \dots, \pi_{[k-1]}$ fixés
- ▶ *Backtrack* : si $\sum_{i=0}^{k-1} G[\pi_{[i]}, \pi_{[i+1]}] > L_{\min} \rightarrow \text{stop}$
- ▶ *Branch-and-bound* : si longueur partielle + coût du retour $> L_{\min} \rightarrow \text{stop}$