

TECHNIQUES D'OPTIMISATION COMBINATOIRE

INTRODUCTION

Laurent Périody

IMA-UCO
3 place A. Leroy
BP 808
49000 Angers Cedex 1
email: laurent.peridy@ima.uco.fr



1/44





Table des matières

1	Introduction à la Théorie de la Complexité	5
2	Classes $\mathcal{P}$ et $\mathcal{NP}$	11
3	Typologie de Problèmes	16
4	Méthodes Heuristiques	19
5	Méthodes Exactes	30
6	Problèmes de tournées	36
7	Parcours eulériens	37



8	Problème dérivé : le problème du postier chinois	38
9	Parcours hamiltoniens	39
10	Heuristiques	40
11	Recherches locales	42
12	Recherches globales	43
13	Problèmes dérivés et extensions du PVC	44



3/44



Liste des Algorithmes

1	— APPARTIENT(x, J)	7
2	Méthode à seuil	26
3	— B&B($x^*, f, \mathcal{S}, evd$)	32





Introduction à la Théorie de la Complexité

Objectif : bref aperçu de la Théorie de la Complexité.

- les cours de L.L.A. en IMA3
- le cours de Recherche Opérationnelle IMA4.

Notions :

- efficacité pratique d'un algorithme,
- difficulté *intrinsèque* d'un Problème d'Optimisation Combinatoire.



Complexité d'un algorithme

Soit $T_A(n)$ le nombre d'opérations élémentaires nécessaires à l'exécution de l'algorithme pour une donnée de taille n .

Définition 1.

On dit qu'un programme est de complexité $O(P(n))$ s'il existe $\alpha < \infty$ tel que :

$$\lim_{n \rightarrow \infty} \frac{T_A(n)}{P(n)} = \alpha$$

Si en outre $P(n)$ est borné supérieurement par un polynôme d'ordre r , on dira que l'algorithme est polynomial d'ordre r .



6/44



Exemple 1. *Considérons l'algorithme A suivant qui renvoie VRAI si l'information x figure dans l'ensemble J .*

Algorithme 1: — $APPARTIENT(x, J)$

Précondition : $J \neq \emptyset$

$i \leftarrow 1$

tant que $i < |J|$ **et** $x \neq J_i$ **faire**

$i \leftarrow i + 1$

fin tant que

si $x = J_i$ **alors**

Retourner VRAI

sinon

Retourner FAUX

fin si



7/44



Remarque 1.

Soit $P(n) = \sum_{i=1}^r \alpha_i \times n^i$. Si $T_A(n) = O(P(n))$, alors on a $T_A(n) = O(n^r)$.
il est inutile de conserver les termes d'ordre inférieur.

S'il n'existe pas de polynôme $P(n)$ tel que $T_A(n) = O(P(n))$
(par exemple, si $T_A(n) = 2^n$ ou $T_A(n) = n!$) alors l'algorithme A
est dit de complexité *exponentielle*



8/44



Complexité	Ordinateur de référence	Ordinateur 100 fois plus rapide	Ordinateur 1 000 fois plus rapide
n	N_1	$100 \cdot N_1$	$1000 \cdot N_1$
n^2	N_2	$10 \cdot N_2$	$31,6 \cdot N_2$
n^3	N_3	$4,64 \cdot N_3$	$10 \cdot N_3$
n^5	N_4	$2,5 \cdot N_4$	$3,98 \cdot N_4$
2^n	N_5	$N_5 + 6,64$	$N_5 + 9,97$
3^n	N_6	$N_6 + 4,19$	$N_6 + 6,29$

TAB. 1 – *Taille de la plus grande instance résolue en 1 heure*

Compte tenu de ces différences importantes, deux questions s'imposent :

- Existe-t-il des POC qui n'admettent pas d'algorithme polynomial ?
- Si oui, comment peut-on déterminer qu'un problème n'admet pas d'algorithme polynomial ?

Réponse(s)??? cf. la suite...



Complexité d'un Problème de Décision



10/44

- Problèmes d'existence (OUI-NON).
- Basée sur un outil théorique — les machines de TÜRING —

Exemple : le problème “ L'élément x figure-t-il dans J ? ” abordé au paragraphe précédent est de complexité $O(n)$.





Classes $\mathcal{P}$ et $\mathcal{NP}$

La Théorie de la Complexité décompose la résolution d'un Problème d'Existence en 2 phases :

1. la recherche d'une solution ;
2. la vérification de la solution.

⇒ 2 classes importantes de Problème d'Existence :

- la classe $\mathcal{P}$,
- la classe $\mathcal{NP}$.



Définition 2.

Un Problème d'Existence appartient à $\mathcal{NP}$ si et seulement si il existe un algorithme polynomial permettant de vérifier toute solution OUI.

Définition 3.

Un Problème d'Existence appartient à $\mathcal{P}$ si et seulement si il existe un algorithme polynomial permettant de répondre OUI au problème (phases 1 et 2 réalisables en temps polynomial).

Evidemment,

$$\mathcal{P} \subset \mathcal{NP}$$

On subodore,

$$??? \mathcal{P} \neq \mathcal{NP} ???$$



12/44



Sous-classe de $\mathcal{NP}$: la classe des problèmes $\mathcal{NP}$ -complet.

Définition 4.

Un Problème d'Existence est dit $\mathcal{NP}$ -complet ssi il appartient à $\mathcal{NP}$ et si tout problème de $\mathcal{NP}$ peut être transformé de façon polynomiale en ce problème.



13/44



Complexité d'un POC

Un Problème d'Optimisation Combinatoire consiste à chercher le minimum x^* d'une fonction f dans un domaine $\mathcal{S}$ fini.

$$f(x^*) = \min_{x \in \mathcal{S}} f(x)$$

Problème d'Existence associé à un POC :

(PE) Existe-t-il x tel que $f(x) \leq k$?

Classe des problèmes $\mathcal{NP}$ -difficile :

Définition 5.

Un POC est dit $\mathcal{NP}$ -difficile ssi le Problème d'Existence associé est $\mathcal{NP}$ -complet.



14/44



Si un POC est $\mathcal{NP}$ -difficile, alors 2 choix sont possibles :

- soit on définit un algorithme polynomial (*heuristique*) qui permet de donner une solution approchée au problème ;
- soit on définit un algorithme exponentiel qui permet d'atteindre l'optimum (*méthode exacte*), mais en ce cas on ne pourra traiter que des problèmes de taille réduite.



15/44





Typologie de Problèmes

Problèmes de “ Routing “

- Problèmes Hamiltoniens ;
- Problèmes Eulériens ;
- Problèmes à contraintes supplémentaires (calendrier, capacité ...)

Programmation Linéaire

- Programmation Linéaire classique ;
- Programmation Linéaire en Nombre Entiers ;
- Programmation Linéaire en 0-1 ;
- Programmation Linéaire Mixte.



Problèmes d'Affectation

- Problème de Sac-à-dos ;
- Problème de Bin-Packing ;
- Problème d'Affectation Généralisée.

Problèmes de Découpe

- Problèmes de type *Bin Packing* ;
- Problèmes de type *Cutting Stock* ;
- Problèmes de type *Knapsack* ;
- Problèmes de type *Pallet Loading*.



17/44



Problèmes d'Ordonnancement

- notion de tâches ;
- notion de ressource :
 - renouvelable / consommable ;
 - disjonctive / parallèle / cumulative ;
- notion de composante temporelle :
 - horizon d'ordonnancement ;
 - fenêtres temporelles ;
- relations de précédences ;
- critères d'optimisation ;
- etc.



18/44





Méthodes Heuristiques

- Théorie de la Complexité $\Rightarrow$ problèmes sans algorithmes polynomiaux (si $\mathcal{P} \neq \mathcal{NP}$),
- optimum en un temps raisonnable $\Rightarrow$ **IMPOSSIBLE**.



Solution approchée



Méthode heuristique



Introduction

Soit un POC du type :

$$f(x^*) = \min_{x \in \mathcal{S}} f(x)$$

Objectif : calculer une solution la moins éloignée possible de la solution optimale avec un algorithme polynomial.

L'ensemble des solutions optimales sera noté $\mathcal{S}^*$.



20/44



Algorithmes construisant une solution

- construction progressive d'*une* solution
- sans remise en cause des choix effectués précédemment.

On distingue :

- les algorithmes gloutons ;
- les méthodes à regret (ou pénalité) ;
- les algorithmes de liste.



21/44



Les algorithmes gloutons

Définition 6.

A chaque itération, on effectue le choix le plus intéressant, relativement à un critère donné.



22/44



Les méthodes à regret (ou pénalité)

Définition 7.

Le choix retenu est celui qui coûte le plus cher si on ne l'effectue pas.



23/44



Les algorithmes de liste

Définition 8.

Ces algorithmes sont dédiés aux problèmes d'ordonnancement et construisent une solution relativement à une liste de priorité donnée.



24/44



Techniques de Voisinage

Principe : partir d'une solution initiale $\rightarrow$ améliorer en examinant des solutions voisines.

La notion de Voisinage

Définition 9 (Voisinage). Le voisinage $V(x)$ d'une solution x appartenant à $\mathcal{S}$ est une fonction $V : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ qui associe à x un sous-ensemble d'éléments de $\mathcal{S}$.

Définition 10 (Minimum local/global). Une solution x de $\mathcal{S}$ est appelée minimum local, respectivement au voisinage V , si

$$f(x) \leq f(x'), \forall x' \in V(x)$$

Tout élément de $\mathcal{S}^*$ sera appelé a contrario minimum global.



25/44





Initialiser(x_0);

$x := x_0$;

$k := 0$;

répéter

 Generer(x' de $V(x)$);

si $f(x') - f(x) < t_k$ **alors**

$x := x'$;

fin si

$k := k + 1$;

jusqu'à Stop

Algorithme 2: *Méthode à seuil*



Soit t_k le seuil à l'itération k .

1. Amélioration itérative.

$$t_k = 0, k = 0, 1, \dots$$

2. Seuil d'acceptation.

$$t_k = c_k, k = 0, 1, \dots \text{ où } c_k \geq 0, c_k \geq c_k + 1.$$

3. Recuit simulé.

Les t_k suivent une distribution F_{c_k} sur R^+ . La probabilité qu'un seuil t_k soit inférieur à une valeur y est donnée par $P_{c_k}\{t_k \leq y\} = F_{c_k}(y)$ où c_k est un paramètre de contrôle.

Exemple :

$$P_{c_k}\{\text{accepter } x'\} = \begin{cases} 1 & \text{si } f(x') \leq f(x) \\ \exp\left(\frac{f(x) - f(x')}{c_k}\right) & \text{si } f(x') > f(x) \end{cases}$$

Attention : réglage difficile, c_0 initial, distribution des c_k , fonction de probabilité...).



Méthode Taboue

Objectif : éviter les mimina locaux.

Choisir une solution initiale $x \in \mathcal{S}$.

$$x^* = x$$

tant que non fini **faire**

Générer un sous-ensemble V^* de solutions non taboues de $V(x)$

Choisir la meilleure solution $x' \in V^*$ par rapport à f .

$$x = x'$$

si $f(x) < f(x^*)$ **alors**

$$x^* = x$$

fin si

Mettre à jour les conditions taboues.

fin tant que

Longueur de la liste taboue : entre 7 et 30.



28/44



Algorithmes Génétiques

1. Définir une représentation génétique du problème.
2. Créer une population initiale $P(0) = x_1^0, \dots, x_N^0$. Soit $t = 0$.
3. Affecter à chaque individu la valeur normalisée $f(x_i)/\bar{f}(t)$ avec $\bar{f}(t) = \sum_{i \in [N]} f(x_i)/N$.
4. Affecter à chaque individu une probabilité $p(x_i, t)$. En utilisant cette distribution, choisir N vecteurs de $P(t)$.
5. Choisir $N/2$ paires. Appliquer le crossing-over avec une certaine probabilité à chaque paire et d'autres opérateurs génétiques tels que la mutation, pour former une nouvelle population $P(t + 1)$.
6. Soit $t = t + 1$ et retourner à l'étape 2.

L'opérateur génétique *mutation* change avec une certaine probabilité m chaque "gène" de l'individu sélectionné.

L'opérateur crossing-over génère un individu à partir d'une paire de parents.



29/44



Méthodes Exactes

- La Recherche Arborescente
 - Fin du cours de programmation linéaire (PLNE)
- la Programmation Dynamique
 - Cours de recherche Opérationnelle, IMA IV



30/44



Recherche Arborescente (Branch Bound)

Objectif : effectuer une exploration *implicite* de l'espace des solutions optimum du problème.

2 phases : **séparation, évaluation.**

Soit x^* une solution optimale du (POC) : $\min_{x \in \mathcal{S}} f(x)$.

Les éléments caractéristiques d'une Recherche Arborescente sont :

- une fonction évaluation par défaut $evd : \mathcal{P}(\mathcal{S}) \rightarrow \mathbb{R}, \forall \mathcal{S}' \subset \mathcal{S}, \forall x \in \mathcal{S}', evd(\mathcal{S}')$
- une évaluation par excès UB de la solution optimale ;
- une liste L de parties de $\mathcal{S}$ non encore explorées ;
- une procédure de séparation $\text{Sep}(\mathcal{S}', (\mathcal{S}'_1, \dots, \mathcal{S}'_r))$ associant à toute partie $\mathcal{S}'$ de $\mathcal{S}$ un r -recouvrement $(\mathcal{S}'_1, \dots, \mathcal{S}'_r)$ (en fait, Sep vérifie $\mathcal{S}' = \bigcup_{i=1}^{i=r} \mathcal{S}'_i$, avec, en outre, $\mathcal{S}'_i \neq \mathcal{S}'$ pour tout $i \in [1, r]$).



31/44



Algorithme 3: — $B\&B(x^*, f, \mathcal{S}, evd)$

$UB \leftarrow +\infty$

$L \leftarrow \{\mathcal{S}\}$

tant que $L \neq \emptyset$ **faire**

choisir $\mathcal{S}' \in L$

$L \leftarrow L \setminus \{\mathcal{S}'\}$

$\text{Sep}(\mathcal{S}', (\mathcal{S}'_1, \dots, \mathcal{S}'_r))$

pour tout $\mathcal{S}'_i$ **tel que** $\mathcal{S}'_i \neq \emptyset$ **et** $evd(\mathcal{S}'_i) < UB$ **faire**

si $\mathcal{S}'_i = \{x\}$ **alors**

$x^* \leftarrow x$

$UB \leftarrow f(x)$

sinon

$L \leftarrow L \cup \{\mathcal{S}'_i\}$

fin si

fin pour

fin tant que



32/44



Différentes explorations

Suivant le choix de $\mathcal{S}'$, on distingue :

- la recherche en profondeur d'abord
si L est une pile (*depth-first search*) ;
- la recherche en largeur
si L est une file (*breadth-first search*) ;
- la recherche meilleur d'abord
si L est une liste de priorité (*best-first search*).



33/44



Programmation Dynamique

La Programmation Dynamique → R.E. BELLMAN

Principe d'Optimalité :

Une politique optimale est telle que, quels que soient l'état initial et la décision initiale, les décisions suivantes doivent constituer une politique optimale par rapport à l'état résultant de la première décision.



34/44



Exemple d'application :

- la recherche d'un chemin minimal par l'algorithme de BELLMAN.



35/44

Principe d'optimalité : tout chemin extrait d'un chemin minimal est un chemin minimal.

Soit $G = (X, U, W)$ un graphe pondéré. La valeur $l_{s,i}$ d'un chemin de valeur minimale du sommet s à tout sommet i (sous réserve d'existence d'un tel chemin) est donc donnée par : $\{ l_{s,s} = 0$

$$l_{s,i} = \min_{j \in \Gamma_i^{-1}} (l_{s,j} + w_{j,i})$$

- Problème de Knapsack





Problèmes de tournées

- Parcours eulériens
- Parcours hamiltoniens
- Problème du voyageur de commerce
- Problème de tournées de véhicules





Parcours eulériens

Définition 11. *Un parcours eulérien est un chemin visitant une et une seule fois chaque arête (arc) du graphe.*

Résolution cas non orienté

- *Il existe un parcours eulérien dans un graphe G non orienté si et seulement si G est connexe et possède 0 ou 2 sommets de degré impair.*

Exemple 2. *La lettre*





Problème dérivé : le problème du postier chinois

Définition 12. *Le problème du postier chinois consiste à construire un circuit visitant au moins une fois chaque arc du graphe à coût total minimal.*

Exemple 3.

- ramassage des ordures ménagères,
- distribution du courrier,
- ...





Parcours hamiltoniens

Définition 13. *Un parcours hamiltonien est un chemin visitant une et une seule fois chaque sommet d'un graphe.*

Problème NP-Difficile

Problème du voyageur de commerce (PVC)

Définition 14 (Problème du voyageur de commerce). Soit $G = (X, U, C)$ un graphe valué complet. Le problème du voyageur de commerce consiste à trouver un circuit hamiltonien de coût minimal.





Heuristiques

- PLUS PROCHE VOISIN

connecter le dernier sommet atteint au sommet libre le plus proche (au sens des coûts).

- MÉTHODES D'INSERTION

Partir d'un circuit réduit à une boucle, choisir un sommet libre et l'insérer entre 2 sommets en minimisant l'augmentation de coût

$$\Delta C = C_{ij} + C_{jk} - C_{ik}$$

- Plus Proche Insertion

choisir le sommet le plus proche de la tournée contruite et l'insérer au meilleur endroit

- Plus Lointaine Insertion

choisir le sommet le plus loin de la tournée contruite et l'insérer



au meilleur endroit

– Meilleure Insertion

*tester tous les sommets sur toutes positions de la tournée et
choisir le meilleur couple (sommet,position)*

– MÉTHODE DE FLETCHER

Au départ aucune arête n'est prise.

A chaque étape, choisir l'arête de coût minimal ne créant ni **fourche**,
ni **cycle**.



41/44





Recherches locales

Objectif : Rechercher parmi les solutions “proches” d’une solution, une solution meilleure et ré-itérer.

- 2-OPT (k-OPT)

Remplacer 2 arêtes non consécutives $[i,j]$ et $[k,l]$ par $[i,k]$ et $[j,l]$.

- OR

Déplacer une chaîne de 1, 2, ou 3 sommets consécutifs pour l’insérer entre 2 autres sommets.





Recherches globales

Objectif : Ressortir des optima locaux.

– Recuit Simulé

1. choisir au hasard une solution proche
2. l'accepter en fonction d'une probabilité contrôlée par un réel positif décroissant lentement, appelé la température.
3. Ré-itérer jusqu'à ce que la température soit proche de 0.

– Méthode Taboue

1. explorer toutes les solutions proches
2. choisir la meilleure non taboue
3. interdire la transformation inverse (transformation “taboue”)





Problèmes dérivés et extensions du PVC

- visiter au moins une ville dans chaque région.
- DARP (Dial A Ride Problem) ou problème du coursier.

Définition 15 (Problème de tournées de véhicules). *Le problème de tournées de véhicules consiste à livrer n clients à partir d'un ou plusieurs dépôts et en utilisant des véhicules de capacité Q , tout en minimisant le coût total.*

Contraintes additionnelles :

- satisfaire les demandes des clients en une seule livraison
- fenêtres horaires
- chargements et déchargements possibles ...

