CHAPTER 1

Conclusion

1.1. Qu'est-ce qu'un problème d'optimisation discrète ?

On a un ensemble fini, et on veut minimiser une fonction définie sur cet ensemble.

Facile, il suffit d'énumérer tous les éléments de l'ensemble !

Sauf que l'ensemble est habituellement très gros ...

1.2. Comment aborder un tel problème ?

L'essentiel est de trouver un *bon modèle :*

- Suffisament simple pour être traité (si possible déjà étudié dans la littérature !)
- Suffisament expressif pour bien représenter la réalité

Un tout petit changement de modèle peut entrainer de gros changements de complexité !

(Cf. ordonnancement).

Alors, quelles questions se poser ?

(1) Quelles sont les variables ? (comment mesurer une solution au problème)
(2) Quelles sont les contraintes ?
    - Peut-on l'exprimer en fonction des variables ?
    - Quelles contraintes sont essentielles ?
    - Quelles contraintes peuvent relaxées ?
(3) Quelle est la fonction à optimiser ?
    - Peut-on l'exprimer simplement en fonction des variables ?
    - Peut-on l'approcher par une fonction plus simple (par ex. linéaire) ?

Exemple. Problème du boucher.

1.3. Reconnaître les difficultés

1.3.1. Problèmes de décision.

Exemple. Problème du sac à dos

1.3.2. Problèmes en entiers (discret<-> continu).

Exemple. Résolution d'équations diophantienne (équations linéaires en entiers)

Une droite contient-elle un point à coordonnées entières ?

Démontrer que $\sqrt{2}$, e ou π sont irrationnels n'est pas trivial.

1.4. Modèles et méthodes polynomiales

1.4.1. Programation linéaire. Principe : simplifier le problème en le rendant continu.

EXEMPLE. Recherche de couplage max dans un graphe biparti.

Plusieurs modèles de plus en plus simples :

(1) Programmation linéaire générale
   - Méthode du simplexe
   - Méthode des points intérieurs
   - Méthode de l'ellipsoïde
   - Résolution polynomiale
   - Théorème de dualité

(2) Problèmes de transport, avec ou sans contraintes
   - Algorithme du simplexe pour les réseaux
   - Théorème d'intégralité !
   - Beaucoup plus rapide *en pratique*.

(3) Problèmes de flot
   - Algorithme de Ford-Fulkerson (parcours en profondeur dans un graphe)
   - Beaucoup plus rapide : $O(n^3)$

1.4.2. Algorithmes dans les graphes, ...

1.5. Problèmes non-polynomiaux

La plupart du temps, il n'y a pas d'algorithme polynomial, et on est obligé de passer à des méthodes énumératives.

Comment gérer l'explosion combinatoire ?

En fait la plupart des idées ci-dessous sont parfaitement naturelles : vous les utilisez déjà intuitivement dans la vie de tous les jours.

1.5.1. Méthodes arborescentes. Organiser la recherche exhaustive, de façon à examiner des solutions proches les unes des autres, et limiter les recalculs.

1.5.2. Restreindre le domaine de recherche. Rendre faisable une recherche exhaustive en restreignant le domaine de recherche.

1.5.2.1. *Minorer et majorer la fonction objectif.*

(1) Résolution approchée par des heuristiques -> minoration

   EXEMPLE. Utiliser Coffman-Graham pour ordonnancer des tâches unitaires sur 3 processeurs ou plus.

(2) Méthodes de relaxation -> majoration

   Relâcher certaines contraintes

   EXEMPLE. Enlever la contrainte d'intégralité dans un problème de programmation linéaire général.

(3) Trouver un problème dual -> majoration

   Si on a un théorème min-max, c'est encore mieux

1.5.2.2. *Restreindre le domaine de recherche à un sous-ensemble dominant.*

EXEMPLE. Pour l'ordonnancement de tâches unitaires, on peut toujours trouver un ordonnancement optimal calé à gauche.

Un tel sous-ensemble de solutions est *dominant.*
On peut restreindre le domaine de recherche à ce sous-ensemble.

EXEMPLE. Pour l'ordonnancement de tâches sans contraintes de resources, cette approche est suffisante pour donner un algorithme polynomial (méthode PERT).

1.5.2.3. *Programmation dynamique.* Utiliser l'indépendances de certaines décisions pour contracter des branches similaires dans un arbre de recherche.

1.5.3. *«C'est pas parfait, mais ca fera l'affaire».* Pour beaucoup de problèmes, il existe des algorithmes polynomiaux donnant une solution approchée avec garantie de qualité (à moins de $x\%$ de la solution optimale).
C'est très souvent largement suffisant dans les applications.
Voire mieux : une famille d'algorithmes A_n de complexité polynomiale tels que :
– Le degré de complexité tends vers l'infini avec n.
– La garantie de qualité x_n tends vers 0 avec n.
– L'algorithme limite A_∞ est optimal.

EXEMPLE. Dans une méthode arborescente,

On peut donc complèment choisir, suivant les besoins, un compromis vitesse / qualité.

EXEMPLE. Algorithme LLL.

1.6. À vous de jouer !

Bibliography

[1] *Problèmes d'ordonnancement, Modélisation, Complexité, Algorithmes* ; J. Carlier, P. Chrétienne, Masson, Études et recherches en informatique, 1988.

[2] *Optimisation combinatoire* (2 tomes), Michel Sakarovitch, Hermann