Recherche locale en optimisation combinatoire : fondements et applications

Frédéric Gardi, Bouygues e-lab, Paris

Plan :

A) Fondements de la recherche locale

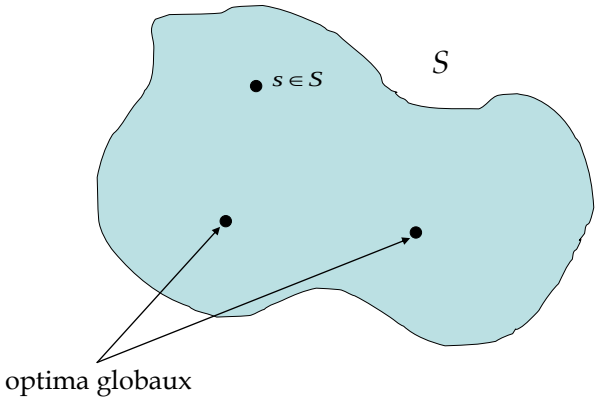
1) Optimisation combinatoire

a) Problème d'optimisation

Problème d'optimisation combinatoire :

$$\begin{array}{c}\text{Minimiser } f(s) \\ s \text{ dans } S\end{array}$$

S : ensemble des solutions du problème
f : fonction de coût



b) Problème de décision

Equivalence entre optimisation et décision.

Décision (existe-t-il une solution de coût c ?) implique optimisation (quelle est la solution de meilleur coût ?). Lorsque l'espace des solutions peut être vide, résoudre un problème d'optimisation (quelle est la meilleure solution ?) revient à résoudre un problème de décision (existe-t-il une solution ?).

Ici : problème d'optimisation. Si pas de solution évidente, alors problème d'optimisation visant à minimiser les contraintes violées, avec coût d'une solution = fonction du nombre et de la nature des contraintes violées.

Nombreux problèmes d'optimisation combinatoire NP-difficiles. Pas d'algorithme efficace de résolution (temps polynomial). Attention : problème NP-difficile ne signifie pas insoluble. Beaucoup d'instances ou classes d'instances de problèmes NP-difficiles sont solubles efficacement. Exemples : 2-SAT, Horn-SAT, coloration/clique/stable des graphes parfaits, etc. Mais aussi : phénomènes de seuil mis en évidence sur SAT.

c) Panorama et histoire des techniques de résolution

Problèmes d'optimisation combinatoires de plus en plus étudiés après guerre : avènement de la RO. Premières techniques avancées (et systématiques) de résolution : fin 1950, début 1960

Méthodes exactes (énumératives) : relaxation, coupes (PLNE), filtrage, propagation (PPC)
Méthodes approchées (heuristiques) : constructive, à base de recherche locale

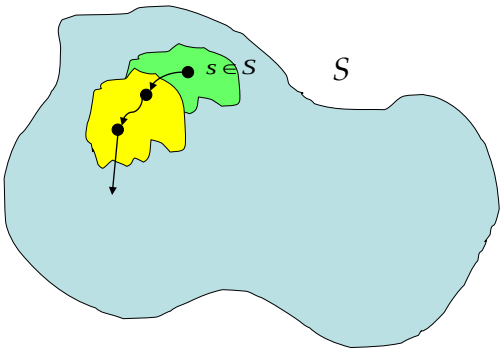
Beaucoup de travaux réalisés autour du problème du voyageur de commerce (TSP)

Succès de la recherche locale surtout pratique (Kernighan & Lin, 1973, TSP)
Resté informelle pendant longtemps, peu de résultats fondamentaux

Ces 10 dernières années + fécondes :

- premiers résultats fondamentaux
- nombreux succès pratiques (implémentation facile et flexible)
- apparition des métaheuristiques (méthodes sophistiquées, analogie avec phénomène naturel)
- diffusée auprès des ingénieurs (info, maths, RO)

2) Recherche locale = recherche par voisinage



Contrairement à ce beaucoup (enseignants, chercheurs, ingénieurs !) font croire, c'est l'ingrédient essentiel de la recherche locale (en anglais « neighborhood search »). Parler de recherche locale = parler de voisinage, et non pas parler de métaheuristique (taboo, recuit, etc).

Pour chaque problème, voire instance de problème, on peut définir des voisinages qui s'avèrent plus ou moins adaptés. Il n'y a pas de recette miracle (de métaheuristiques miracles !), certaines analogies et certains principes à respecter seulement.

a) Fonction de voisinage

Voisinage N(s) d'une solution s : ensemble des solutions que l'on peut « atteindre » à partir de cette solution

- Exemple : le tri (minimiser le nombre de couples dans le désordre)

i) une solution est voisine d'une autre si on l'obtient par permutation de 2 éléments
ii) voisinage + riche si on considère permutation de k éléments

Analyse de la taille du voisinage :
i) C(2,n) = n(n-1)/2 = O(n^2) voisins
ii) C(k,n) = O(n^k) voisins

Si k est de l'ordre de n, le voisinage est de taille exponentielle en n

On parle de grand voisinage ou voisinage large, lorsque la taille du voisinage défini est exponentiel en la taille des données d'entrée, ou polynomial avec un degré élevé (O(n^k) avec k grand). Par opposition, nous dirons petit voisinage, lorsque celui-ci est polynomial de faible degré.

- Autre exemple : voyageur de commerce

i) deux solutions sont voisines si 2 arêtes échangées : 2-opt
ii) deux solutions sont voisines si k arêtes échangées : k-opt

Voisinage « k-échange » apparaît dans quasiment tous les problèmes combinatoires.

+ sophistiqué : « Variable-depth search algorithm » (Lin & Kernighan, 1970 : uniform graph partitioning, 1973 : TSP)

Séquence de « best 2-exchange ». Exemple sur le TSP avec profondeur k : vous choisissez le meilleur 2-opt (même si négatif), vous simulez l'échange. Vous continuez k fois sans modifier aux arêtes déjà touchées. Soit g_i le gain à l'étape i ; le total G(i) des gains est la somme des gains des étapes 1 à i. Choisir la valeur de i pour laquelle G(i) est maximale. Si G(i) > 0 (ou G(i) >= 0), validez toutes les modifications.

Permet de passer outre un (voire plusieurs) optimum local apparent sur le 2-opt. Ne coûte pas beaucoup plus cher en temps de calcul qu'un 2-opt. 2 idées maîtresses : la fonction de gain et la règle bloquante (ne pas modifier une arête déjà touchée). Sur d'autres problèmes que le TSP, peut-être + difficile à formaliser !

- Autre exemple : ordonnancement de tâches sur des machines

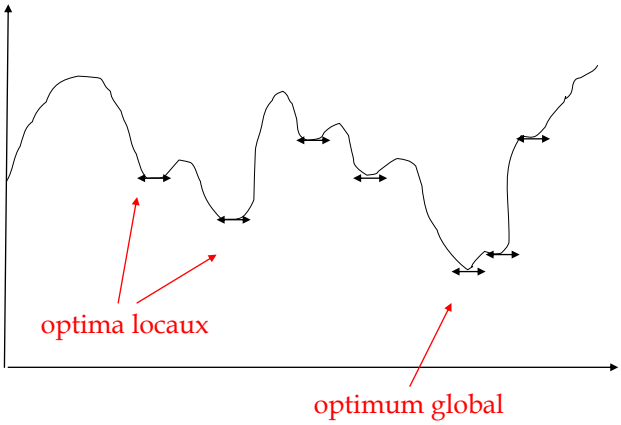
Si une machine : « sequencing »
i) « swap » : permutation de deux tâches dans la séquence
ii) « transpose » : permutation de deux tâches consécutives
iii) « block swap » : permutation de deux blocs de tâches consécutives
iv) « insert » : insertion d'une tâche à une autre position
v) « block insert » : insertion d'un bloc de tâches consécutives à une autre position
vi) « invert » : invertion d'un bloc de tâches consécutives

Si plusieurs machines : « scheduling » :
vii) « swap » : permutation de deux tâches sur deux machines
viii) « reassign / move » : insertion d'une tâche d'une machine sur une autre machine
ix) « 2-swap» : permutation d'1 tâche contre 2 sur machines différente
x) « 2-reassign » : insertion de 2 tâches sur machine différente

Nous appelons cela des *transformations* de la solution courante ou encore des *mouvements*. Une solution est voisine d'une autre, si on l'obtient par une transformation. Ces transformations définissent donc des voisinages.

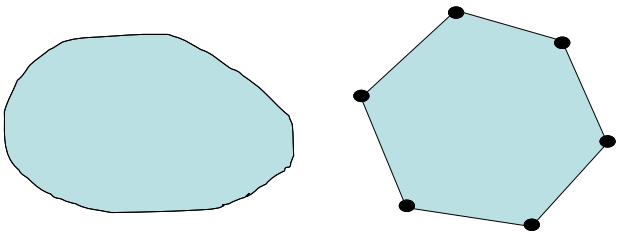
b) Optimum local vs. optimum global

Optimum local si s dans S est tel que f(s) <= f(s') quel que soit s' dans N(s)
Exemple sur une fonction (continue) en 2 dimensions :



Notons que la notion d'optimum local est fonction du voisinage (notion différente de l'analyse continue même si certaine analogie).

Structure de certains problèmes implique : local = global (propriété de convexité)



Convexité est une notion issue de l'analyse continue. Son équivalent en analyse combinatoire peut se voir comme : il existe un chemin entre toute solution et l'optimum global tel que le coût des solutions ne fait que diminuer le long du chemin. (En effet, on peut voir l'ensemble des solutions comme un graphe : deux sommets sont reliés si les solutions correspondants sont voisines.)

Même lorsque c'est le cas, il faut savoir trouver ce chemin : notion de voisinage exact. Pour une instance, un voisinage est exact lorsqu'il conduit à un optimum global.

- Exemple simple : le tri

k-échange exact, mais mieux 1-échange adjacent exact (« bubble sort »)

- Autre exemple : l'algorithme du simplexe pour la PL (Dantzig, 1949)

Il existe des choix de pivot exact : choix d'une variable entrante de coût réduit négatif (analogie avec le gradient en optimisation continue ?)

- Autre exemple : l'algorithme de Ford-Fulkerson (1962) pour le flot maximum (« augmenting path ») ou l'algorithme d'Edmonds pour le couplage maximum (« alternating path »). Provient de caractérisations fortes induisant la notion de voisinage : un flot n'est pas maximum s'il existe un chemin améliorant (soit, un chemin le long duquel on peut augmenter le flot d'une unité au moins) ; un couplage n'est pas maximum s'il existe une chaîne alternée.

Ces caractérisations déterminent des voisinages exacts : on converge vers un optimum global. Toutefois, cela n'implique pas une convergence en temps polynomial (simplexe exponentiel, Klee & Minty, 1971 ; Ford-Fulkerson exponentiel, etc).

Choix du voisin est alors déterminant. Pour le simplexe, on a toujours pas trouvé de voisin qui garantisse une convergence en temps polynomial. Pour le flot ou le couplage, oui. Flot : choisir le chemin améliorant le long du plus dans le graphe résiduel (O(n^3), Edmonds-Karp, 1972).

c) Exploration du voisinage

Soit le choix du voisin avec lequel on poursuit la recherche : très important !

Question du choix du voisin intimement liée avec la définition du voisinage :

- petit voisinage implique peu de choix, mais une exploration rapide
- grand voisinage implique beaucoup de choix, mais une exploration difficile

Règles basiques de sélection du voisin (appelé « pivoting rule ») :
- « first improvement » : le premier voisin améliorant le coût de la solution courante
- « best improvement » : le voisin de meilleur coût

(conduisent par définition à un optimum local)

Moins basiques :
- sélection d'un voisin détériorant le coût de la solution courante (avec une certaine probabilité) : principe au cœur du recuit simulé (« simulated annealing »)
- sélection d'un voisin qui n'appartient pas à une certaine liste de solutions interdites : principe au cœur de la méthode taboo (« tabu search »)

Disons plus sur les voisinages larges. A manier avec précaution, car difficile à mettre en œuvre. Le voisinage large doit être explorable efficacement, sinon convergence très lente.

En fait, trois façons de construire des voisinages larges explorables efficacement :
i) méthodes de type « variable-depth search » avec exploration de façon heuristique (plus généralement méthodes du type « ejection chains » de Glover)
ii) voisinages larges explorées par des techniques de flots, couplages ou programmation dynamique
iii) voisinages induit par des sous-problèmes polynomiaux

Exemple pour iii). Cornéjuols et al. (1983) donnent un algorithme en O(n) pour trouver un tour optimal dans les graphes de Halin. Graphe de Halin : prolongez un arbre sans sommet de degré 2 dans le plan et joignez les feuilles par un cycle de façon à qu'il reste planaire. On dit que H est une Halin extension du tour T si on peut construire H tel que T soit un sous-graph de H. Si on a cet algorithme qui crée la Halin extension, alors l'ensemble des tours dans la Halin extension est un voisinage du tour T. Un graphe de Halin peut posséder un nombre exponentiel de tours. Trouver le meilleur tour dans la Halin extension peut se faire en temps linéaire.

Nous verrons un autre exemple mélant ii) et iii) dans la section Applications au travers du car sequencing.

d) Questions fondamentales

Convergence des algorithmes de LS très difficiles à analyser, même avec des voisinages simples. Trois questions fondamentales :

i) converge-t-on vers un optimum global (local = global) ?
ii) combien d'itérations faut-il dans le pire des cas (ou en moyenne) pour converger vers un optimum local ?
iii) complexité globale (exploration + descente) de l'heuristique par LS

De telles analyses sont très difficiles à faire, encore plus sur des problèmes pratiques.
Toutefois quelques résultats théoriques sur la descente, pour le TSP notamment :

- Lueker (1976) construit des instances pour lesquelles l'heuristique 2-opt conduit à un nombre exponentiel d'itérations suivant une règle de pivot. Chandra & al (1997) étende la construction pour k-opt avec tout $k > 2$ fixé ;
- Papadimitiou & Steiglitz (1977) montre que si P différent NP, alors il n'existe pas de LS en temps polynomial par mouvement qui peut garantir un ratio d'approximation constant, même si un nombre exponentiel de mouvements est autorisé ;
- Papadimitiou & Steiglitz (1978) montre que pour 2-opt, 3-opt et k-opt avec $k < 3n/8$, il existe des instances avec un seul tour optimal et un nombre exponentiel de tours locallement optimal, chacun plus grand que l'optimal d'un facteur exponentiel ;

Depuis, une théorie est née : la classe PLS (Johson & al, 1988). Problème : une solution initiale, un voisinage défini et une règle de pivot.

Problème PLS : problème pour lequels l'optimalité locale peut être vérifiée en temps polynomial. Problèmes PLS-complets : les plus durs de PLS.

En particulier : TSP avec k-opt (k constant) ou le Lin-Kernighan voisinage ; on a alors que dans le pire des cas, la descente prend un temps exponentiel quel que soit la règle de pivot utilisée (même stochastique, même exponentielle).

Les études originales et récentes de Tovey (1997) sur des graphes de solutions tels que l'hypercube d-régulier montre que la descente prend un nombre exponentiel d'itérations pour beaucoup de voisinages, mais à un bon comportement en moyenne. C'est ce qui semble ressortir de la pratique : pire des cas mauvais, cas moyen excellent.

3) Stratégies de recherche

a) Stratégies basiques

Appelé « standard local search algorithm », « descent heuristic », « iterative improvement » :

Calculer une solution initiale s ;
Tant que
    Définir un voisinage N(s) de s ;

S'il existe un voisin s' dans N(s) tel que f(s') < f(s) alors
    s devient s' ;
Sinon
    Retourner s ;

Dans la littérature, on trouve généralement f(s') < f(s) (plus simple à analyser en théorie ?)
Retourne un optimum local, mais de faible qualité si N(s) petit voisinage.

Attention, nous préconisons : f(s') <= f(s) !
C'est-à-dire que toutes les solutions dans le voisinage sont plus coûteuses que s. Beaucoup moins restrictifs car en pratique beaucoup de solutions de même coût dans les voisinages.

A mon avis, cette méthode doit toujours être employée pour commencer à travailler, tester les voisinages, évaluer la difficulté de convergence.

- Approche « multistart » : appliquez l'heuristique de descente plusieurs fois à partir de solution initiale différente. Plusieurs personnes ont testé, n'apporte pas grand-chose (Johnson, 1990, TSP). Dernière idée que l'on observe des blocages tôt dans la descente alors que l'on a tout essayé (ce qui est peu probable !!!).

- Approche « multilevel » : appliquez l'heuristique de descente en combinant les voisinages.

Par exemple, descente avec un voisinage, puis nouvelle descente avec un voisinage différent à partir de l'optimum local précédent : appelé « iterated local search » (Johnson, 1990, TSP : « iterated Lin-Kernighan »).

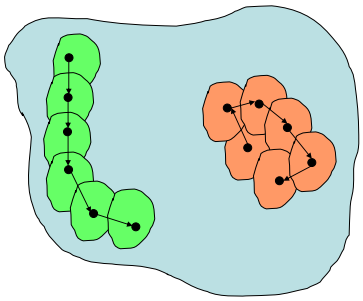
Les combinaisons de voisinages peuvent et doivent être plus fines, plus complexes. Nous le verrons par la suite dans les Applications. En fait, je pense que la combinaison de voisinage est l'un des moyens les plus efficaces pour descendre vers des optima locaux de bonne qualité.

Voir les travaux récents d'Helsgaun (1999) sur le TSP et nos travaux sur le « car sequencing », (en version académique ou industrielle RENAULT) et un problème de planification de RH chez France Telecom.

b) Stratégies avancées et métaheuristiques

Principe d'intensification / diversification de la recherche :

- intensification : exploration autour de la solution courante (rester dans une région de l'espace des solutions et y explorer les solutions)
- diversification : exploration visant à s'éloigner de la solution courante (vers de nouvelles régions de l'espace des solutions)



La clé est la diversification : ne pas rester « enfermer » dans une région de l'espace des solutions (TSP à + de 1000 sommets : 10^25000 tours >> 10^100 atomes dans l'univers).

Paradoxe de l'optimisation combinatoire :

- d'un côté, la combinatoire rend le problème difficile empêchant une approche exacte (énumération impossible)
- d'un autre côté, dans les grands problèmes combinatoires, il y a beaucoup de solutions de même coût : il est souvent possible d'aller de région en région en parcourant ces solutions de coût identique (convexité de certains espaces / régions de recherche ?)

Donc très important : lorsque le voisinage est petit, sélectionner un voisin de coût égal est primordial ! Sinon blocage très tôt de la recherche.

- Recuit simulé (Kirkpatrick & al 1983, Cerny, 1985) : appartient à une classe d'heuristique appelé « heuristique à seuil » (« threshold algorithms »).

Calculer une solution initiale s ;
k = 0 ;
Tant que
    Définir un voisinage N(s) de s ;
    S'il existe un voisin s' dans N(s) tel que f(s') - f(s) <= t_k alors
        s devient s' ;
        k = k + 1 ;
    Sinon
        Retourner s ;

Avec t_k >= 0, t_k >= t_{k+1} et lim_{k vers 0} t_k = 0. On peut imaginer tout simplement t_k égal à une constante. Cela signifie qu'à certaines itérations, on accepte une solution détériorant le coût de la solution courante dans le voisinage. Avec des t_k décroissants, on s'assure de ne plus détériorer après un certain nombre d'itération.

Dans le recuit simulé, t_k est une variable aléatoire. De là, t_k peut varier entre 0 et l'infini avec une certaine distribution de probabilité pour que t_k <= t (distribution exponentielle négative). Nous avons alors la définition du critère d'acceptation d'une solution j à partir de i après k itérations :

P(j accepté) = 1 si f(j) <= f(i)
= exp((f(i) – f(j)) / c_k) si f(j) > f(i)

avec c_k paramètre de contrôle (qui joue un rôle important dans l'analyse de la convergence)

Analogie avec la physique : le recuit, qui peut être simulé par des techniques de Monte Carlo (Metropolis et al., 1953). Critère de Metropolis : exp((E(i) – E(j)) / (k_B * T)) duquel découle l'algorithme de simulation de Métropolis. Ici énergie = coût et T dépend du nombre d'itérations.

En fait, en choisissant les c_k décroissant, on définit de façon implicite les notions de diversification / intensification dans l'heuristique. Diversification tôt dans la descente, puisque c_k grand et donc probabilité de choisir un voisin détériorant grand. Intensification plus tard puisque c_k petit et probabilité de retenir une solution moins bonne très faible. Pour résumer, de grandes détériorations peuvent être acceptées avec une probabilité de plus en plus petite (mais toujours positive).

Pour les algorithmes à seuil (version déterministe du recuit simulé), on ne possède pas de résultat général de convergence. Pour le recuit simulé, oui. Par modélisation du processus à l'aide de chaînes de Markov infinies, on a des résultats de convergence : sous certaines conditions, un optimum global est trouvé avec une probabilité 1 à l'infini.

Beaucoup de résultats de convergence depuis 20 ans sur le sujet, mais essentiellement théorique (valeurs des c_k pour assurer la convergence, l'accélerer, …). Nombreux résultats sur la réussite pratique du recuit simulé. Par exempel, succès pour le TSP, problèmes de scheduling, VLSI design.

- Tabu (Glover, 1986) : appartient à une classe d'heuristiques appelés « recherche locale dynamique », car des informations sont utilisées à l'itération k provenant des itérations précédentes.

L'idée de mémoire est au cœur de la méthode taboo.

```
Calculer une solution initiale s ;
Tant que
        Définir une restriction V du voisinage N(s) de s ;
        S'il existe un voisin s' dans V tel que f(s') <= f(s) alors
                s devient s' ;
        Sinon
                Retourner s ;
```

V est construit à partir d'une liste de solutions taboo, c'est-à-dire interdites. Cela permet d'éviter de choisir à nouveau des solutions déjà visitées récemment. En fait, comme le recuit simulé la notion de diversification est introduite dans l'heuristique même, avec le fait que l'on cherche à s'éloigner des solutions déjà rencontrées, au travers de cette liste taboo.

Définition du taboo peut être complexifié avec :

- le mécanisme dit d'aspiration : une solution devient taboo ou n'est plus considéré comme taboo sous certaines conditions, voire avec une certaine probabilité ;
- liste taboo variable ;
- combinaison avec recuit simulé ;

Pas de résultats de convergence. Succès pratiques : QAP, coloration, stable, vehicle routing, course scheduling.

- Recherche locale génétique (Muhlenbeim & al, 1988) : tirée des algorithmes génétiques et évolutionnaires (Holland, 1975) : gestion d'une population de solutions que l'on fait muter et croiser.

La recherche locale est ajoutée au processus d'évolution de façon à améliorer le coût des solutions sur lesquelles on travaille.

i) *Initialize* : générer une population de n solutions ;
ii) *Improve* : utiliser la recherche locale pour remplacer ces n solutions par leurs optima locaux respectifs ;
iii) *Recombine* : augmenter la population en ajoutant m solutions combinées à partir des n premières ;
iv) *Select* : réduire la population à sa taille originale en sélectionnant n solutions ;
v) *Improve* : Remplacer ces m solutions par leurs optima locaux respectifs ;
vi) Evolute : Répéter les étapes 3, 4, 5 ;

De bons résultats sont reportées sur problèmes de scheduling, en particulier de sequencing (car solution = séquence s'y prête). Plus ou moins facilement utilisable selon le contexte. D'après moi, très difficile à mettre en œuvre en pratique. Mais les idées de mutation et combinaison de solutions sont à retenir et peuvent être utiles.

- Autres approches (difficiles à mettre en œuvre) :

- réseau de neurones artificiels : preuve d'efficacité pratique sur TSP, course scheduling
- colonies de fourmis : véritablement efficace si croisée avec LS (efficacité sur TSP)
- optimisation par essaim particulaire (généralisation des fourmis et algorithmes à base d'agents) : efficacité sur problème discret/continu

A mon sens, de telles approches ne se justifient pas en pratique.