Optimisation combinatoire – Métaheuristiques

Original Pierre Brezellec

Laboratoire Génome et Informatique, Evry

(modifié par Joël Pothier)

Présentation informelle du problème - Définition formelle du problème

LE PROBLEME DU VOYAGEUR DE COMMERCE

50 villes -> 49! solutions possibles, i.e. $6{,}08.10^{62}$ tournées possibles

Prenons un ordinateur de π millimètres calculant un milliard de solutions par seconde. Sachant que le diamètre équatorial de la terre est de 12 756 kilomètres, on peut mettre 12 756 000 000 de ces ordinateurs les uns à la suite des autres sur l'équateur. On peut ainsi calculer 12 576 000 000 000 000 000 solutions par seconde ($1{,}25.\ 10^{19}$)

Pour être certain de trouver la tournée la plus courte, il faut considérer toutes les tournées possibles. Il nous faudra alors $4{,}766.10^{43}$ secondes avec le super ordinateur que l'on vient de présenter. Note : $4{,}766.10^{43}$ est l'équivalent de $1{,}51.10^{34}$ siècles

CARACTERISATION DU PROBLEME

Etant donné une fonction f à valeurs numériques définie sur X et un ensemble fini F(P) inclus dans X, trouver un élément x de F(P) qui atteint l'optimum de f

Les éléments de F(P) sont appelées solutions réalisables

CARACTERISATION DU PROBLEME

Optimiser (maximiser ou minimiser) la fonction objectif f définie sur X, l'optimum - appelé x* - devant appartenir au domaine $F(P) \subset X$

La condition $x \subset F(P) \subset X$ est appelée contrainte

SAT (problème SAT: satisfiable):

étant donnée une formule logique : existe-t il une interprétation qui la satisfasse ?

x1 x2 booléens: expression (x1 ou x2) et (non(x1) ou non(x2))

complexité en 2^n

Problème du sac à dos:

Tableau d'objets/propriétés

	O1	O2	O3...	On					
P1					≤ b1				
P2					≤ b2				
...									
Pm					≤ bm				

max $\sum c_j x_j$

$x_i = 0$ si O_j n'est pas dans le sac

$x_i = 1$ si O_j est dans le sac

$\sum a_{ij} x_j \leq b_j$ (contraintes)

Le fait que les x_i vaillent 0 ou 1 empêche une résolution numérique.

Résolution approchée:

- méthodes de relaxation: exemple: $x_i \in [0,1[$ dans le problème du sac à dos, résolution numérique, puis choix si $x_i > 0.5$ alors $x_i = 1$, sinon $x_i = 0$.

- heuristiques: guidées par astuces
- méta-heuristiques : indépendantes du problème traité

P ?= NP

Cook (1971): tout problème peut se réduire à SAT en temps polynomial (reformulation).

(Informatique quantique ?)

Problèmes de décision = problèmes d'optimisation

Problèmes de décision : 2 réponses : oui/non

Classes P et NP

Exemples :

- a) parité d'un nombre entier,

- b) plus court chemin G dans un graphe, a et b sommets de G et B un nombre entier. Question : existe t'il un chemin entre a et b de longueur inférieure à B.

-c) Question : existe t'il un chemin élémentaire (cad ne passant pas 2 fois par un même sommet) entre a et b de longueur supérieure à B ?

-d) G admet il un cycle hamiltonien (cad passant une fois et une seule par chaque sommet) ?

Classe P : classes de problèmes de décision polynomiaux. Un problème appartient à P si il peut être résolu par un algorithme A de complexité $O(N^k)$ où k est une constante et N la taille du problème.

Problèmes ci dessus :

- a) O(1) -> division par 2

- b) $O(N^2)$

- c) et d) jusqu'ici pas trouvé

NP : classes de problème de décision pouvant être résolus en temps polynomial par un algo non déterministe

Déf : un problème appartient à NP ssi pour tout jeu de données du problème ayant pour réponse « oui », il existe un « certificat » permettant avec un algorithme polynomial, de vérifier que la réponse au problème est effectivement « oui ».

Problèmes c) et d) : la vérification, si on donne le chemin ou l'hamiltonien est faite en temps polynomial.

- problème de la « non primalité » : « n » est il divisible par un nombre autre que 1 et lui même ? Ce problème appartient à NP car :

- si on considère une instance où la réponse est « oui », cad que « n » n'est pas premier, il admet un diviseur « a » et en prenant « a » comme certificat, la division par « a » (en temps polnomial) vérifie la réponse au problème, donc ce problème appartient à NP

Complémentaire : « n » est il premier ? : Ici la détermination d'un certificat est moins aisée (mais la démonstration existe…).

Note : P est inclus dans NP

NP complets : en 1970, Cook a montré que parmi les problèmes de décision, certains étaient plus difficiles à résoudre que d'autres. Conjecture : seuls des algorithmes énumératifs de complexité au moins $O(2^N)$ permettent la résolution de ces problèmes « NP complets ». Les problèmes c) et d) dans un graphe quelconque sont NP complets. Par contre, dans un graphe sans circuit, le problème c) est en O(m), « m » étant le nombre d'arcs.

Problèmes d'optimisation NP difficiles :

Problème b) : trouver un chemin entre a et b de longueur minimale

Problème c) : trouver un chemin entre a et b de longueur maximale

Les versions « optimisation » sont au moins aussi difficiles à résoudre que les versions « décision ». Si le problème de décision est « NP complet », la version « optimisation » sera qualifiée de « NP difficiles » (énumération de type séparation et évaluation).

TECHNIQUES DE RESOLUTION EXACTES DU PROBLEME

Recherche exhaustive (ou énumération explicite)

Technique de complexité exponentielle

Branch-and-bound (ou énumération implicite)

Complexité importante (bien que le nombre de solutions considérées est moindre que dans une recherche exhaustive)

Programmation dynamique

TECHNIQUES DE RESOLUTION APPROCHEE DU PROBLEME : RECHERCHE D'UN OPTIMUM LOCAL

Méthodes de relaxation

On relâche le problème pour rendre plus aisé sa résolution

La solution obtenue n'est pas nécessairement la solution optimale du problème

Heuristiques

Recherche guidée par des "astuces" qui dépendent du problème traité

Méta-heuristiques

Méthodes de recherche indépendantes du problème traité (recuit simulé, tabou search, ...)

Pour les problèmes intéressants, i.e. les problèmes NP,on ne connait pas d'algorithmes exacts et rapides permettant de résoudre la question posée

EXPONENTIEL versus POLYNOMIAL

	n=1	n=10	n=100	n=500	n=1000
Exponentielle	1,1	2593	13780,61	$4{,}969.10^{20}$	$2{,}469.10^{41}$
Polynomiale	1	10^{10}	10^{20}	$9{,}765.10^{26}$	10^{30}

CADRE DE TRAVAIL:

LE PROBLEME A "RESOUDRE"

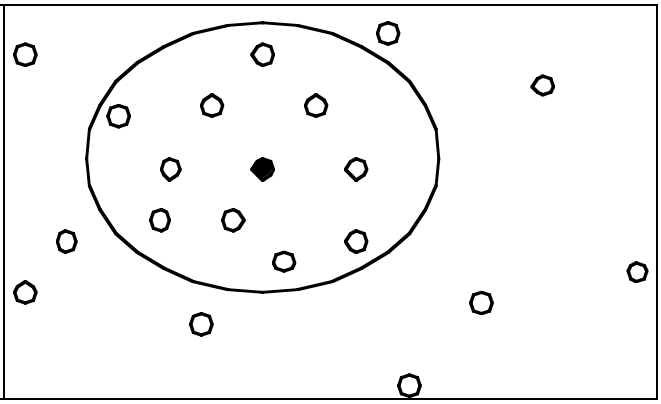
f : $\mathcal{E}$ –> $\mathcal{R}$

$\mathcal{E}$ est couramment appelé "espace de recherche". On cherche l'optimum de f, i.e. l'élément x de $\mathcal{E}$ minimisant (ou maximisant) f. On suppose que l'espace $\mathcal{E}$ est de grande taille

LE VOISINAGE

Notion de voisinage : l'espace est structuré

A chaque élément *x* de $\mathcal{E}$, on associe un voisinage. Un voisinage est un ensemble d'éléments de $\mathcal{E}$

$\mathcal{E}$ = espace des chaines binaires de longueur 6 $\mathcal{E} = 2^6 = 64$ $x \in E$	

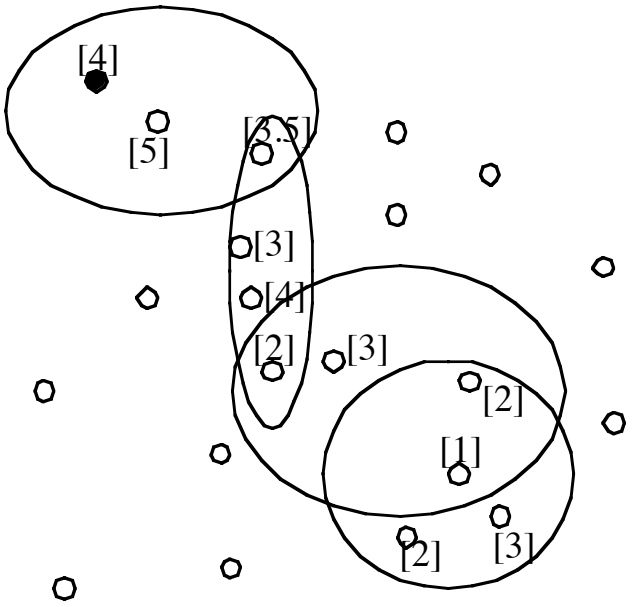
Voisinage de x: inverser successivement chacun des bits	0 1 0 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 0 1 1 1 0 1 0 1 0 0

LA DESCENTE (recherche d'un minimum)

choisir une solution initiale *s*

```
fin <-  FALSE
WHILE {fin = FALSE}
soit s' le voisin de s qui minimise f
Δ <- f(s') - f(s)
IF Δ ≤ 0
s <- s'
ELSE
fin <- TRUE
ENDIF
ENDWHILE
```

LA DESCENTE (EXEMPLE)



IDEES A LA BASE DES META-HEURISTIQUES

Il n'existe pas de technique générale pouvant réduire l'écart entre un optimum local et un optimum global en améliorant systématiquement le coût d'une solution}

Accepter provisoirement une mauvaise solution pour trouver une meilleure solution

- pour éviter de rester bloqué sur un optimum local

Eviter de boucler

- pour parcourir le plus d'espace possible

RECUIT SIMULE (Metropolis [1953], Kirkpatrick, Gelatt, Vecci [1982], indépendamment Cerny en 1985)

Méthode inspirée d'une analogie avec un phénomène physique (exemple de l'aimantation). Analogie avec Boltzmann.

Ne dispose pas en elle-même de mécanismes "anti-bouclage"

RECUIT SIMULE (recherche d'un minimum)

```
choisir une solution initiale s
choisir une température initiale T_i > 0
choisir une température finale T_f > 0      // T_f < T_i
choisir un nombre d'itération NB (à une température donnée)
choisir le coefficient de diminution de la température Φ ∈ [0,1[
 T <- T_{i}$
WHILE { T_f < T}
FOR {k = 1 to NB}
S' <- voisin aléatoire de s
Δ <- f(s') - f(s)$
IF {Δ ≤ 0}
     s <- s'
ELSE
     s <- s' avec la probabilité e^{-Δ/T}
ENDIF
ENDFOR
T <- T*Φ
ENDWHILE
```

Explication :

- si T grande, exp(-Δ/T) est de l'ordre de 1, on garde toujours le mouvement, même si il est mauvais.

- si T très petit, exp(-Δ/T) est de l'ordre de 0, donc les mouvements qui augmentent l'énergie (la différence) sont disqualifiés.

- méthode : on tire au hasard dans l'intervalle [0,1[, si le nombre est < exp(-Δ/T) , on garde sinon on jette.

Amélioration: stocker la meilleure solution !!!

C'est une bonne méthode pour le voyageur de commerce, mais pas très bonne pour les problèmes d'ordonnancement.

LE TABOU (Fred Glover [1989], « tabu search »)

Garder des traces du passé pour mieux s'orienter dans le futur

L'idée est d'utiliser une (petite) mémoire (la liste tabou) pour éviter de tomber dans un optimum local et/ou pour éviter de boucler (cycles de petite taille). Dans la liste tabou, on peut garder des configurations, des points ou des régions visitées, ou plus généralement des attributs, qui vont éviter des « mouvements » déjà faits.

Algorithme :

On va garder tous les points du voisinages - ou une partie échantillonnée de ces points si l'ensemble est trop grand (cas d'une fonction continue par exemple) – qu'on appelle N(s)). On va retirer de ces points ceux qui sont dans la liste tabou T(s,k) pour obtenir N(s,k)=N(s)-T(s,k). On calcule la valeur de la fonction f à minimiser pour chacun de ces points de N(s,k), on trie les points par ordre de f croissant (le 1^{er} est donc le meilleur du voisinage, mais pas forcément meilleur que le point courant qui n'est évidemment pas inclus dans le voisinage). Si le 1^{er} point est meilleur que la meilleure solution obtenue jusqu'ici, on le garde. Ensuite on met tous les points du voisinage dans la liste tabou. Et on itère jusqu'à par exemple ne plus avoir de changement (ou un autre critère).

Ainsi, à chaque itération, l'algorithme tabou choisit le meilleur voisin non tabou, même si celui-ci dégrade la fonction.

On peut aussi ajouter un critère d'« aspiration », qui déterminera si un point de la liste tabou (ou plusieurs, donc un sous ensemble A(s,k)) peut quand même être utilisé. qui permette d'utiliser un point tabou car il remplirait quand même une condition désirée (par exemple, si un mouvement interdit par la liste tabou conduit à une valeur de la fonction f qui serait meilleure que celle obtenue jusqu'ici. Evidemment, cette configuration ne peut pas survenir si on garde des points dans la liste tabou : si un point de la liste tabou a la meilleure valeur de f, il a forcément été compté auparavant. Des critères d'aspirations plus sophistiqués peuvent être utilisés pour ce critère d'aspiration.

```
choisir la taille k de la liste tabou L_TABOU
choisir un nombre d'itération NB
choisir une solution initiale s
meilleure_evaluation <- f(s)
meilleure_solution <- s
change <- TRUE
WHILE {change = TRUE}
   change <- FALSE
   FOR {iteration = 1 to NB}
      identifier le voisinage N(s)
      T(s,k) <- les points de N(s) de la liste L_TABOU
      N(s,k) <- N(s)-T(s,k)+A(s,k)
      trier le voisinage en fonction de la fonction f
      s' = élément de N(s,k) tel que f(s') minimum
      IF {f(s') < meilleure_evaluation}
        meilleure_solution <- s'
        meilleure_evaluation <- f(s')
        change = TRUE
      ENDIF
      mettre à jour la liste tabou,
         (i.e. ajouter N(s,k) à L_TABOU)
      s <- s'
   ENDFOR
ENDWHILE
```

Note : la gestion de la liste tabou est de type FIFO (First In First Out)

La liste tabou ne doit pas être trop grande (sinon, on bloque les mouvements), ni trop petite. En général, k=7 est pas mal. Néanmoins, la taille de la liste doit être proportionnelle à la taille du problème.

Algorithme tabou de base : mémoire à court terme (liste taboue), assure une diversification à court terme

Algorithme tabou évolué : mémoire à court terme (liste taboue) + mémoire à long terme pour assurer l'intensification et/ou la diversification

LES ALGORITHMES GENETIQUES (1ère conférence en 1986)

Basés sur une analogie avec l'évolution (reproduction/selection, mutations, croisements). John Holland (1960/1970), David Goldberg (1980/1990).

La notion de voisinage est remplacée par l'application d'opérateurs de "mutation" et de "croisement" (cross-over).

Etude simultanée d'un ensemble de solutions *versus* étude d'une solution pour le recuit et le tabou

Codage :

On cherche à avoir un codage sur une chaîne de 0/1 pour pouvoir effectuer les croisements (et mutations).

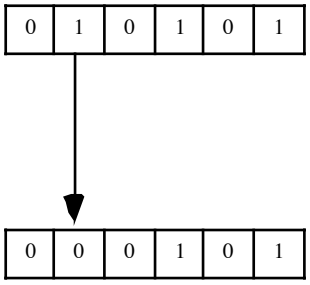
Si x est une variable de f(x) à optimiser sur l'intervalle $[x_{min},x_{max}]$.

On re-écrit x :2^n $(x-x_{min})/(x_{max}-x_{min})$

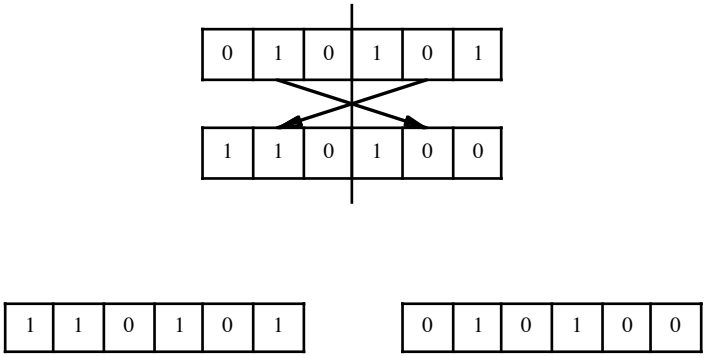
Ce qui donne une chaîne de n bits (on garde une précision de n digits en base 2) puisque $(x-x_{min})/(x_{max}-x_{min})$ est dans l'intervalle [0,1].

Une population complète of N individus de n bits est générée.

Mutation



Croisement



Algorithme général :

```
Choisir la taille t de la population
Générer la population initiale P      // i.e. un ensemble de solutions
WHILE {le critère d'arrêt n'est pas satisfait}
      Mutations et Recombinaisons par croisements
      Calcul pour chaque individu x_i de la fonction de fitness f(x_i)
      (et de S, la somme totale des f(x_i))
      Pour chaque x_i calcul de p(x_i)=f(x_i)/S
      Reproduction des individus de P selon les p(x_i),
            c'est à dire sélection probabiliste de t
            individus selon leur p(x_i)
      Mise à jour de P
ENDWHILE
```

Le choix de la représentation en bits est primordial, car les croisements peuvent être inadaptés.

On peut être amené à choisir par exemple un code où les chiffres successifs ne diffèrent que d'un bit (utile en électronique, où on veut éviter que plusieurs contacts ne s'inversent lorsqu'on passe d'un nombre à un nombre juste supérieur, car on passe alors par des intermédiaires). Le code de Gray (Franck Gray, 1953) est un tel code.

Le **code de Gray**, également appelé **binaire réfléchi**, est un type de codage binaire permettant de ne modifier qu'un seul bit à la fois quand un nombre est augmenté d'une unité.

Codage décimal	Codage binaire naturel	Codage Gray ou binaire réfléchi
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

Pour passer d'une ligne à la suivante, on inverse le bit le plus à droite possible conduisant à un nombre nouveau.

Une autre méthode de calcul permettant de passer d'un nombre de Gray au suivant, et qui présente l'avantage de ne pas nécessiter de connaître l'ensemble des nombres précédents est la suivante :

si le nombre de 1 est pair, il faut inverser le dernier chiffre.

si le nombre de 1 est impair, il faut inverser le chiffre situé à gauche du 1 le plus à droite.

Et enfin, la méthode la plus simple consiste à calculer le OU Exclusif (symbolisé ici par le caractère ^) entre le binaire de départ et ce même binaire décalé d'un rang à droite.

Exemples : On veut représenter 7 en code de Gray.
7 s'écrit 0111 en base 2.

```
  0111
^ 0011
 -----
  0100
```

Pour des fonctions de variables réelles, on emploie le croisement barycentrique (généralement).

Le « crossover » est alors :

$x'_1 = \alpha\, x_1 + (1- \alpha)\, x_2$

$x'_2 = (1- \alpha)\, x_1 + \alpha\, x_2$

où α est tiré au hasard dans l'intervalle [-0.5,1.5] (et donc $(1- \alpha)$ est dans [1.5,-0.5])

Et une mutation est: $x'_1 = x_1 + B(0,\sigma)$

où $B(0,\sigma)$ est un bruit gaussien centré en 0 et d'écart type σ

Pour les algorithmes génétiques, le codage et les opérateurs choisis conditionnent grandement la qualité des solutions qu'on peut obtenir. Il ne suffit pas d'introduire une fonction pour en retirer l'optimum. Sans un codage et des opérateurs appropriés, les résultats seront au mieux médiocres.

Il existe une technique (« sharing ») qui permet de modifier la fitness d'un individu selon le nombre d'individus semblables (i.e. similaires dans l'espace du problème) qui l'entourent : la fitness d'un individu dans une zone peuplée est diminuée. Ceci pour éviter trop de représentants d'une même solution, et favoriser les représentants uniques - ou peu nombreux - d'une solution.

HARMONY SEARCH

La recherche d'harmonie (« harmony search », Geem et al., 2001) est inspirée par le processus d'improvisation des musiciens, et ressemble aux algorithmes génétiques. Elle fait partie de la même famille des algorithmes évolutionnaires.

Il s'agit donc de trouver le vecteur **x** qui minimise/maximise la fonction f(**x**) (**x** est un vecteur de taille *n*).

Algorithme :

Pas 0 : On génère aléatoirement *hms* vecteurs *n* (*hms*: harmony memory size) qu'on garde en mémoire.

Pas 1 : on génère un nouveau vecteur **x'** dont les composantes x'_i sont tirés comme suit :

- avec une probabilité *hmcr* (« harmony memory considering rate; $0 \leq hmrc \leq 1$), on prend la valeur x_i d'un vecteur choisi au hasard uniformément parmi les vecteurs en mémoire
- avec une probabilité *1-hmcr* on tire le x'_i au hasard dans l'intervalle permis

Pas 2 : si le x'_i tiré au pas 1 provient de la mémoire (d'un vecteur existant) :

- avec une probabilité *par* (pitch adjusting rate; $0 \leq par \leq 1$), modifier x'_i d'une petite valeur $\pm\delta$ pour une variable discrète, ou $fw \times \text{uniform}(-1,1)$ (avec δ l'espace entre deux variables discrètes voisines, et *fw* (fret width, formerly bandwidth) le changement maximum dans l'ajustement du pitch, en général 0.01 à 0.001 × l'intervalle permis
- avec la probabilité *(1-par)* on ne fait rien

Pas 3 : si **x'** est meilleur que le plus « mauvais » vecteur en mémoire, on le garde et on retire le plus « mauvais » vecteur.

Pas 4 : on répète les pas 1 à 3 jusqu'à ce que le critère d'arrêt (maximum d'itérations) soit atteint.

Les paramètres de l'algorithme « harmony search » sont :

- *hms* taille de la mémoire, varie de 1 à 100 (valeur typique 30)
- *hmcr* la probabilité de choisir une valeur déjà en mémoire. Varie généralement de 0.7 à 0.99 (valeur typique 0.9)
- *par* le taux de choix d'une valeur voisine. Varie généralement de 0.1 à 0.5 (valeur typique 0.3)

Il est possible

CONCLUSION

Convergence (à la limite) des Méta-heuristiques

Robustesse aux conditions initiales

Nécessité d'automatiser le processus d'initialisation des paramètres

Plus on injecte de la "culture" dans la définition du voisinage, meilleurs sont les résultats

Où l'heuristique irrigue la Méta-heuristique

Combiner les Méta-Heuristiques entre elles

"Au pays des Z'heuristiques, l'inceste n'est pas tabou"

Existence d'autres Meta-Heuristiques (Bruitage, etc.)