Heuristiques, Méta-heuristiques

ESP (Ecole Supérieure Polytechnique) de Dakar

Optimisation en informatique

Alain Faye

6 – Heuristiques

Plan

- Introduction
- Recherche locale
- Métaheuristiques
  - Recuit simulé
  - Recherche tabou
  - Algo. de population: algorithme de fourmis

Introduction

Résoudre un problème d'optimisation

- Algorithme de descente
  - on améliore une solution en cherchant dans son voisinage de meilleures solutions
  - on tombe dans optima locaux (donc sous-optimal)
- Sortir des optima locaux
  - Méthode probabiliste : recuit simulé
  - Interdire les mouvements récents : tabou
  - Agrandir les voisinages
- Algorithmes de population
  - Plusieurs solutions qui s'améliorent en coopérant

Cadre commun : optimisation combinatoire

- Définition

  Chercher un élément A* dans un ensemble discret E qui **minimise** une fonction objectif f(A)

  $A^* \in E \ / \ f(A^*) = \min_{A \in E} f(A)$

  A* est une solution optimale du problème

  f(A*) est l'optimum

Exemples

- Minimiser une fonction quadratique en variable binaire

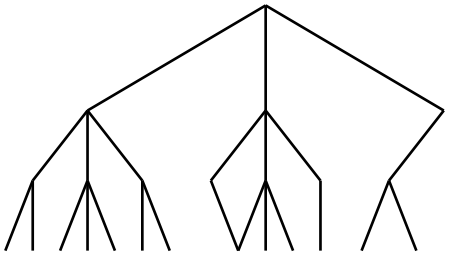
  Min $x^tQx + c^tx$ avec $x \in \{0,1\}^n$

- Problème du voyageur de commerce

  trouver un cycle passant exactement une fois chez chaque client et de coût minimum

Limites des méthodes exactes

- Les **méthodes exactes** sont basées sur énumération implicite
- L'encadrement de la valeur optimale permet de limiter l'énumération



  - Séparation :
    décomposer une instance en sous instances indépendantes
  - Evaluation :
    interdire des affectations partielles qui sont infaisables ou sous optimales

- Limite : temps de calcul prohibitif quand la taille du problème est grande

Heuristique

Objectif : calculer une solution approchée (sous-optimale) du problème en un temps raisonnable

Question: qualité de la solution fournie. Est-elle éloignée ou pas de l'optimum?

- Heuristique spécifique
  - dédiée à un problème
  - Qualité de la solution ? Elle dépend de l'heuristique et de l'instance du problème
- Méta-heuristique
  - méthode générale s'appliquant à tout problème d'optimisation discrète
  - Qualité de la solution bonne en pratique
- Algorithme approché avec garantie de performance
  - heuristique spécifique et qui fournit une solution avec une certaine garantie de qualité (distance à l'optimum bornée)

Recherche locale

Recherche locale : voisinage d'une solution

- Défini à l'aide d'une **transformation élémentaire** (ou locale)
- Définition : on appelle transformation toute opération permettant de changer une solution A de E en une solution A' de E (ensemble de toutes les solutions). Une transformation sera considérée comme élémentaire (ou locale) si elle ne modifie que faiblement la structure de la solution à laquelle on l'applique.

Recherche locale : voisinage d'une solution

- Définition (voisinage) : étant donnée une transformation locale, le voisinage V(A) d'une solution A est l'ensemble des solutions que l'on peut obtenir en appliquant à A cette transformation locale.

Exemples

- les vecteurs 0-1

A=(0,1,1,0,1)

On change un bit ou deux de A

A ’=(1,0,1,0,1)

La distance de Hamming (on compte les bits différents) $d(A, A') = 2$

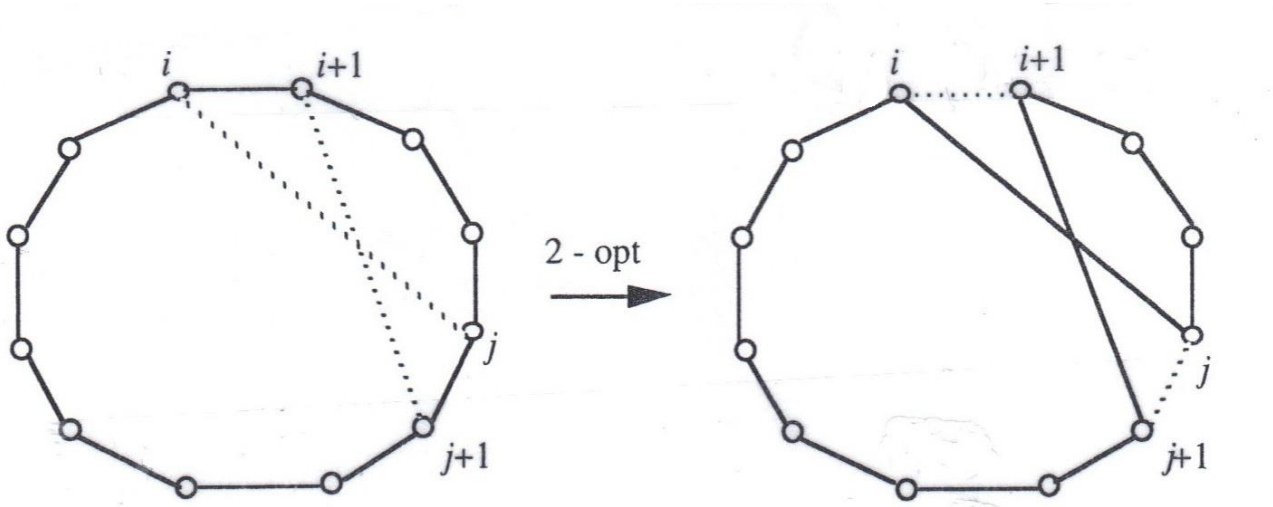
$V(A)=\{A' \in E: d(A, A') \leq 2\}$

Exemple

- Voyageur de commerce

  A = un cycle passant par tous les clients

  échanger deux arêtes. On choisit 2 arêtes non adjacentes dans le cycle et on les remplace par celles qui permettent de reconstituer un autre cycle . On parle de **voisinage 2-opt**.



Méthode de descente

A ← Génère solution initiale ()

Pour m = 1 à *Max Mouvements* Faire

A' ← Sélectionne dans voisinage (A)

Si f(A') < f(A) Alors A ← A'

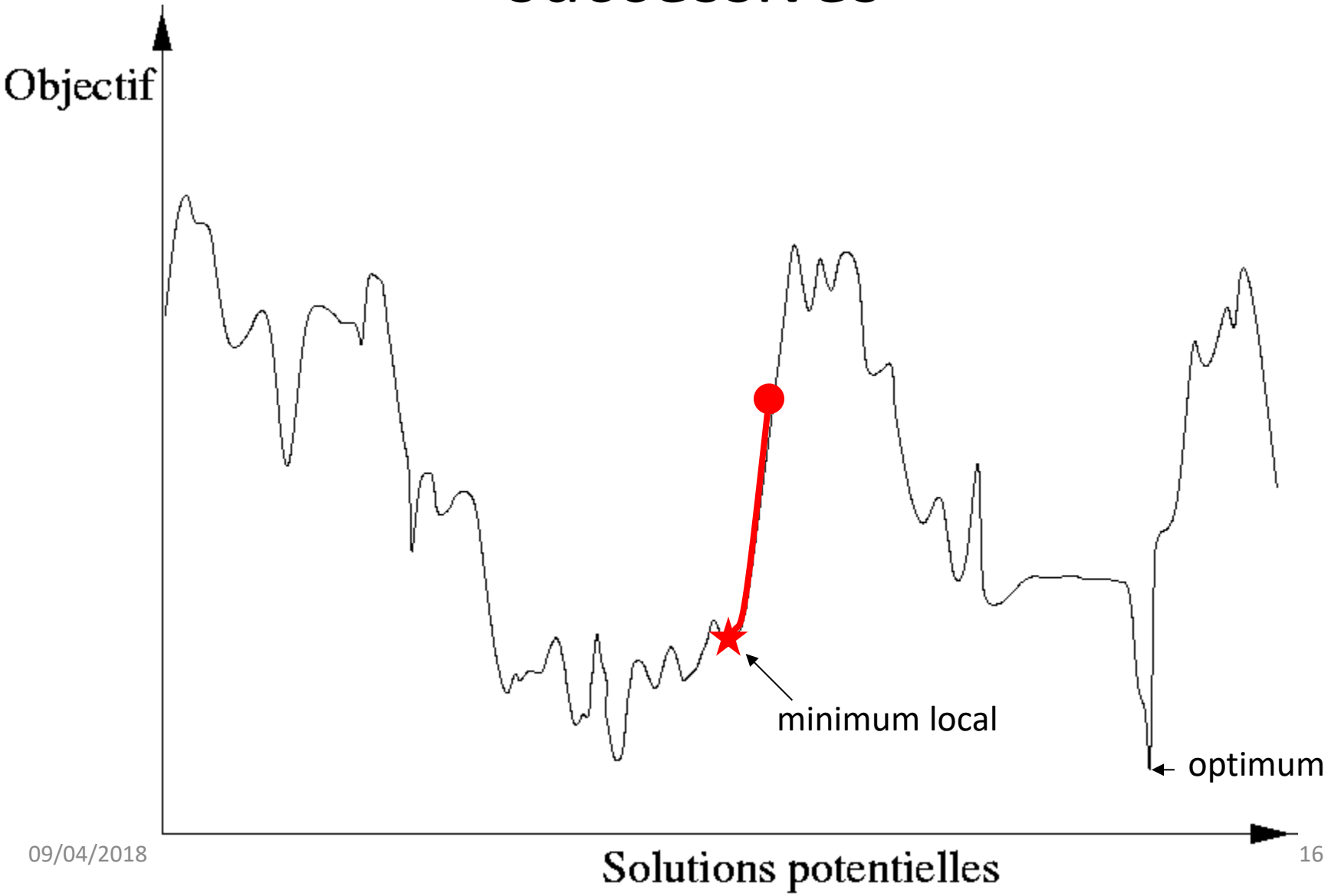
retourner A , f(A)

Modes de sélection

- Sélection aléatoire dans le voisinage

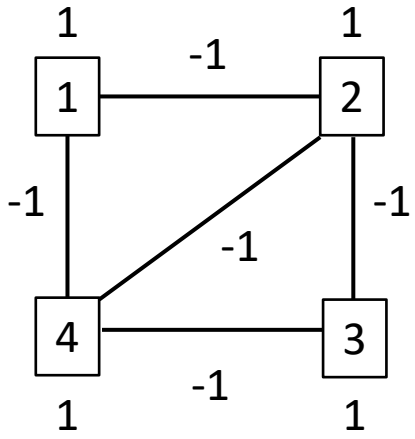
- Parcours exhaustif du voisinage et on retient le meilleur voisin

Recherche par améliorations successives



Exemple de min local

- Fonction quadratique en 0-1



$f(x)=x_1+\ldots+x_4 - x_1x_2 - x_1x_4 - x_2x_3 - x_2x_4 - x_3x_4$

2-voisinage : on modifie au plus 2 bits

A=(0,0,0,0) est minimum local : $\forall x \in V(A)\backslash\{A\}$ $f(x)>f(A)$

A*=(1,1,1,1) est minimum global f(A*)=-1

Minima locaux

- **Comment sortir des minima locaux ?**
- Dégradation de la solution courante
  - Marche aléatoire
  - Recuit Simulé
  - Recherche Tabou
- Modification du voisinage – Recherche à voisinages variables
- Méthodes exploitant une population de solutions

Marche aléatoire

A ← Génère solution initiale ()

Pour m = 1 à *Max Mouvements* Faire

A' ← Sélectionne dans voisinage (A)

Si f(A') < f(A) Alors A ← A'

Sinon Faire A ← A' avec une probabilité p

retourner A , f(A)

Comment choisir p ?

Métaheuristiques

L’algorithme de recuit-simulé : historique

- Le recuit physique
  - Ce processus est utilisé en métallurgie pour améliorer la qualité d'un solide.
  - On cherche à atteindre un état d'énergie minimale qui correspond à une structure stable du métal.
  - En partant d'une haute température à laquelle la matière est devenue liquide, la phase de refroidissement conduit la matière à retrouver sa forme solide par une diminution progressive de la température.

L’algorithme de recuit-simulé : historique

- Le recuit simulé (Simulated Annealing)
  - Expériences réalisées par Metropolis et al. dans les années 50 pour simuler l'évolution de ce processus de recuit physique (Metropolis53).
  - L’utilisation pour la résolution des problèmes d'optimisation combinatoire est beaucoup plus récente et date des années 80 (Kirkpatrick83,Cerny85).
  - Le recuit simulé est la première métaheuristique qui a été proposée.

Recuit-simulé : principes généraux

- **critère de Metropolis** : on accepte de dégrader la solution selon une probabilité qui dépend de la température T.
  - $\Delta = f(A') - f(A)$
  - A' est accepté avec une probabilité $e^{-\Delta/T}$
    - Si $\Delta<0$ A' améliore et est accepté
    - Si $\Delta\geq0$ A' dégrade et est accepté avec une probabilité d'autant plus forte que T est grande
- On part avec une température élevée que l'on baisse progressivement

Schéma d'algorithme de recuit-simulé

A ← Génère solution initiale ()

initialiser T

Répéter

A' ← Sélection aléatoire dans voisinage (A)

Δ= f(A') - f(A)

Si CritèreMetropololis (Δ, T) = accepté

alors A ← A'

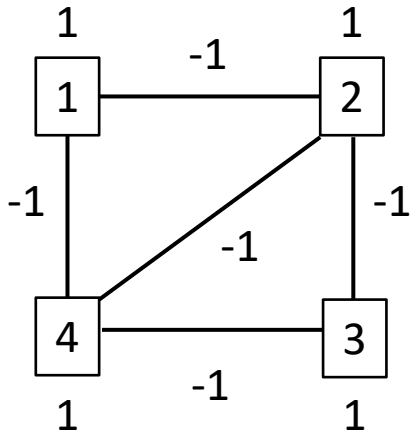
mettre T à jour en fonction du schéma de refroidissement

jusqu'à condition de fin

retourner la meilleure solution trouvée

Exemple sortir min local

- Fonction quadratique en 0-1



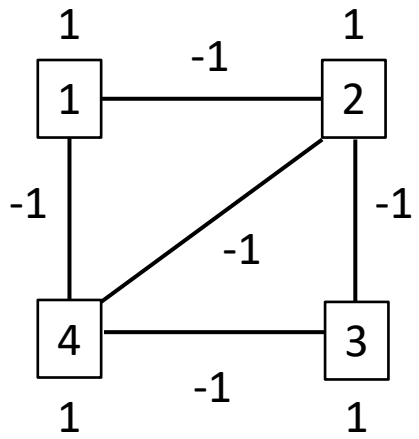
$f(x)=x_1+\ldots+x_4 - x_1x_2 - x_1x_4 - x_2x_3 - x_2x_4 - x_3x_4$

2-voisinage : on modifie au plus 2 bits

A=(0,0,0,0) est minimum local : $\forall x \in V(A)\backslash\{A\}$ $f(x)>f(A)$

A*=(1,1,1,1) est minimum global f(A*)=-1

Exemple sortir min local



$f(x)=x_1+\ldots+x_4 - x_1x_2 - x_1x_4 - x_2x_3 - x_2x_4 - x_3x_4$

- On modifie 2 bits de A=(0,0,0,0) choisis aléatoirement, par exemple 1 et 3
  A'=(1,0,1,0), f(A')=2, Δ=f(A')-f(A)=2
- Critère de Metropolis: A' accepté avec proba = $e^{-\Delta/T}$
  Supposons T=2.
  On tire un nombre aléatoire entre 0 et 1. Par exemple a=0.15
  a=0.15<$e^{-2/2}$=0.37 $\Rightarrow$ le mouvement, bien que dégradant, est accepté.
- On se retrouve sur le point A=(1,0,1,0).
  Maintenant A est dans le 2-voisinage de A* l'optimum.

Recuit-simulé : schéma de refroidissement

- La fonction qui spécifie l'évolution de température est le schéma de refroidissement
- La température décroît par palier
  - On fait des itérations à température constante
- La température décroit selon une loi géométrique $T \leftarrow \rho T$ avec $\rho \in [0.9 , 0.99]$

Algorithme de recuit-simulé avec paliers et refroidissement géométrique

A ← Génère solution initiale ()

T ← T_0 température initiale

Répéter

nb_move ← 0

Pour i=1 à iter_palier

A' ← Sélection aléatoire dans voisinage (A)

Δ= f(A') - f(A)

Si CritèreMetropololis (Δ, T) = accepté

alors A ← A' ; nb_move ← nb_move + 1

Taux_accepté ← nb_move/iter_palier

T ← ρT

jusqu'à condition de fin

retourner la meilleure solution trouvée

Réglage des paramètres du recuit-simulé

- Température initiale.
  Choisir T_0 telle que taux_accepté > 50% au début

- Critères d'arrêt
  - Taux_accepté proche de 0
  - Pas d'amélioration de la meilleure solution trouvée

- Implémentation de Johnson et al.
  - Un compteur est initialisé à 0
  - Il est incrémenté quand le taux_accepté < seuil
  - Il est remis à 0 quand on améliore la meilleure solution
  - Si le compteur est trop élevé on stoppe

Le recuit-simulé : conclusion

- Avantages
  - Méthode qui donne de bons résultats
  - Simple à implémenter

- Inconvénients
  - réglage des paramètres : T_0, nombre d'itérations par palier, seuils. Pas toujours facile et propre à chaque problème

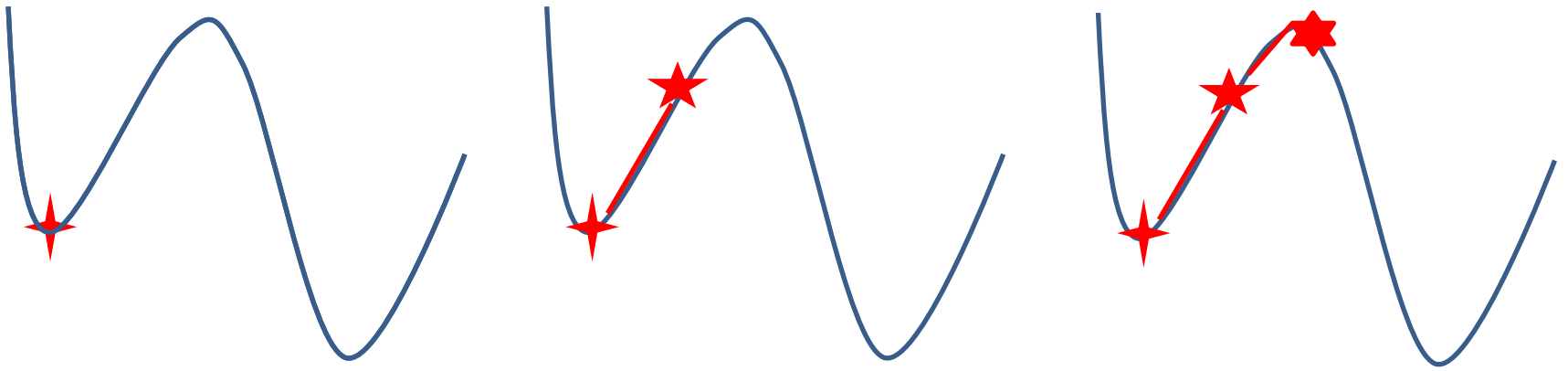
La recherche tabou: historique

- Initialement proposée par F. Glover 86
- Depuis de nombreuses variantes et extensions ont été proposées
- Nouveau concept pour sortir des minimas locaux : la liste tabou

La recherche tabou: principes

- On mémorise dans une liste les mouvements que l'on a fait : la liste tabou
- Ces mouvements sont interdits pendant un certain nombre d'itérations : longueur de la liste tabou
- A chaque itération l'algorithme choisit le meilleur voisin non tabou.

La recherche tabou: principes



On est dans min local
On choisit le mouvement
dans le voisinage
qui dégrade le moins

→

On choisit le mouvement
dans le voisinage
qui dégrade le moins et qui
n'est pas tabou

→

On est sorti du min local

La recherche tabou: la liste tabou

- Divers contenus possibles de la liste tabou
- On peut mettre les solutions sur lesquelles on est passé (liste de solutions)
- On peut mettre le type de mouvement que l'on a fait (liste d'attributs)

La recherche tabou: exemple de liste tabou

- Minimisation d'une fonction quadratique en 0-1.
- Quand on passe d'un vecteur 0-1 à un voisin

  on modifie un bit (ou plus) et on indique pour combien de temps on ne peut plus y toucher

La recherche tabou: exemple de liste tabou

- On définit un tableau IsTabuUntil(i) (de longueur n=nombre de variables)
- IsTabuUntil(i) indique jusqu'à quelle itération le bit i est interdit

- **Condition pour déterminer si le bit i est tabou**
  IsTabuUntil(i)>iter (iter numéro de l'itération courante)

- **Procédure pour rendre le bit i tabou pendant h itérations**
  IsTabuUntil(i):=iter+h

Recherche tabou: algorithme

Fonction **RechercheTabou** ( )

A* ← Génère solution initiale ()

ListeTabou ← ∅

Pour iter = 1 à *Max Mouvements* Faire

A ← Sélection du meilleur dans voisinage(A)

excepté voisins dans ListeTabou

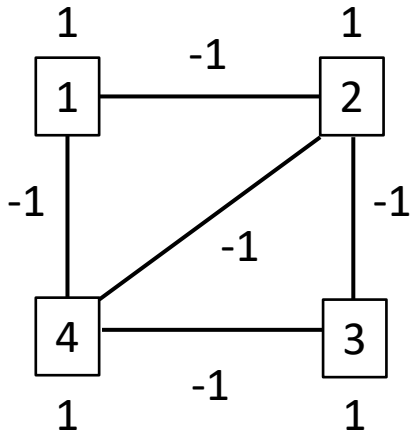
MiseAJour(ListeTabou)

Si f(A) < f(A*) Alors A* ← A

retourner A* , f(A*)

Exemple sortir min local – liste tabou

- Fonction quadratique en 0-1



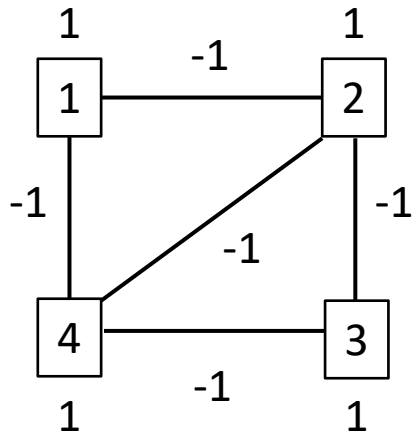
$f(x)=x_1+\ldots+x_4 - x_1x_2 - x_1x_4 - x_2x_3 - x_2x_4 - x_3x_4$

2-voisinage : on modifie au plus 2 bits

A=(0,0,0,0) est minimum local : $\forall x \in V(A)\backslash\{A\}$ $f(x)>f(A)$

A*=(1,1,1,1) est minimum global f(A*)=-1

Exemple sortir min local – liste tabou



$f(x)=x_1+...+x_4 - x_1x_2 - x_1x_4 - x_2x_3 – x_2x_4 - x_3x_4$

- sol. initiale A=(0,0,0,0) , f(A)=0, durée tabou h=2
- iter=1.
  Dans le 2-voisinage de A, on choisit le meilleur point: A'=(1,0,0,0), f(A')=1
  Le bit 1 est tabou jusqu'à l'itération 1+2=3.
- iter=2. On pose A=A'
  Dans le 2-voisinage de A (sans changer le bit 1), meilleur point A'=(1,1,0,1), f(A')=0
  Les bits 2 et 4 tabou jusqu'à l'itération 2+2=4
- iter=3. On pose A=A'
  Dans le 2-voisinage de A (sans changer bits 2 et 4), meilleur point A'=(1,1,1,1), f(A')=-1

  On est sur l'optimum

Recherche Tabou : améliorations

- Liste tabou
  - Réglage dynamique de la longueur de la liste tabou
    - à augmenter si solutions fréquemment répétées (diversification)
    - à diminuer en cas d'absence d'améliorations (intensification)
    - réglage aléatoire périodique parmi [Lmin, Lmax]
- Critère d'aspiration
  - Autoriser des mouvements tabou qui mènent à une meilleure solution

Recherche Tabou : conclusion

- Avantages
  - Méthode qui donne de bons résultats
  - Simple à implémenter

- Inconvénients
  - réglage des paramètres : longueur liste tabou
  - Contenu de la liste tabou : qu'est-ce qu'on y met ? spécifique à chaque problème

Algorithmes de population

- Dans les méthodes précédentes on part d'une solution et à partir de cette solution on essaie de se rapprocher d'une solution optimale
- Les algorithmes de populations:
  - Algo. génétique: à partir d'une population de solutions, on fait des croisements entre les meilleurs individus de cette population dans l'objectif de créer un individu proche de l'optimum
  - Algo. animalier (fourmis, essaim d'abeilles): individus (solutions) qui échangent des infos pour trouver une bonne solution

Algorithme de fourmis

- Dorigo et al. (1991)
- Simuler le comportement observé des fourmis
- Une fourmi limitée en intelligence
- mais collectivement les fourmis construisent des solutions optimales pour trouver le plus court chemin pour aller chercher nourriture
- Partage d'info par la phéromone

Phéromone

- Une fourmi qui se déplace laisse sur son passage une trace de phéromone
- Phéromone est une substance odorante que peuvent détecter les autres fourmis
- Les fourmis choisissent le chemin le plus concentré en phéromone c'est-à-dire sur lequel le plus d'aller-retours ont été effectués
- Le plus d'aller-retours signifie donc que c'est le plus court chemin (le plus rapide)

Algorithme ACO (Ant Colony Optimisation)

- Initialiser les traces de phéromones
- Boucle t = 1 à n
  - Pour chaque fourmi k = 1 à m
    - Construire une solution en s'appuyant sur les traces de phéromones
  - Mettre à jour les traces de phéromones
    - on ajoute les dépôts de chaque fourmi et on retranche l'évaporation des phéromones déposées à l'étape précédente

Application au voyageur de commerce (TSP)

Règle aléatoire de déplacement d'un sommet i à j pour une fourmi k
Probabilité d'aller de i à j pour une fourmi k à l'itération t

$$p_{ij}{}^{k}(t) = \begin{cases} \dfrac{\sigma_{ij}{}^{\alpha}(t)\eta_{ij}{}^{\beta}}{\sum_{l \in J_i{}^{k}} \sigma_{il}{}^{\alpha}(t)\eta_{il}{}^{\beta}} & si\ j \in J_i{}^{k} \\ 0 & sinon \end{cases}$$

$J_i{}^{k}$ est l'ensemble des sommets atteignables lorsque la fourmi k se trouve sur le sommet i

$\sigma_{ij}\ \ (t)$ quantité de phéromone sur l'arc (i,j) à l'itération t

$\eta_{ij}\ \ = \frac{1}{d_{ij}}$ inverse de la distance de i à j (visibilité)

α, β paramètres réglant l'importance entre la phéromone et la visibilité de l'arc

Quantité de phéromone déposée

Quantité de phéromone déposée par une fourmi k sur son parcours à l'itération t
Plus le parcours est court plus la quantité est importante

$$\Delta\sigma_{ij}{}^{k}(t) = \begin{cases} \dfrac{Q}{L^{k}(t)} & si\ (i,j) \in T^{k}(t) \\ 0 & sinon \end{cases}$$

$T^{k}(t)$ est l'ensemble des arcs parcourus par la fourmi k à l'itération t
$L^{k}(t)$ est la longueur du parcours fait par la fourmi k à l'itération t
Q est un paramètre de réglage

Mise à jour phéromone

Sur chaque arc, on enlève l'évaporation et
on ajoute les traces laissées par toutes les fourmis

$$\sigma_{ij}(t+1) = (1-\rho)\sigma_{ij}(t) + \sum_{k=1}^{m} \Delta\sigma_{ij}{}^{k}(t)$$

m est le nombre total de fourmis
$0 < \rho < 1$ est le taux d'évaporation des phéromones

Exemple TSP - 5 sommets

distance	1	2	3	4	5
1	x	1	2	2	2
2	1	x	1	2	2
3	2	1	x	1	2
4	2	2	1	x	10
5	2	2	2	10	x

ACO avec 4 fourmis, $\alpha=\beta=1$, Q=1, $\rho=½$, 8 itérations
Phéromone initiale = 1 sur chaque arc
Les fourmis partent toutes du sommet 1

Itérations 1-3

Tour des fourmis

itération	Fourmi 1	Fourmi 2	Fourmi 3	Fourmi 4
1	1-2-4-3-5-1 = 8	1-3-4-5-2-1 = 16	1-3-2-4-5-1 = 16	1-2-3-5-4-1 = 16
2	1-2-3-4-5-1 = 15	1-3-4-5-2-1 = 16	1-4-2-5-3-1 = 10	1-3-2-5-4-1 = 17
3	1-4-5-2-3-1 = 17	1-2-5-4-3-1 = 16	1-2-4-5-3-1 = 17	1-3-4-2-5-1 = 9

phéromone fin itération 3

σ(4)	1	2	3	4	5
1	x	0.023	0.058	0.005	0.000
2	0.000	x	0.004	0.015	0.067
3	0.028	0.002	x	0.056	0.000
4	0.002	0.057	0.008	x	0.019
5	0.056	0.004	0.016	0.010	x

Itérations 4-6

Tour des fourmis

itération	Fourmi 1	Fourmi 2	Fourmi 3	Fourmi 4
4	1-3-4-2-5-1 = 9	1-3-4-2-5-1 = 9	1-3-4-2-5-1 = 9	1-2-5-3-4-1 = 8
5	1-3-4-2-5-1 = 9	1-2-5-3-4-1 = 8	1-2-5-3-4-1 = 8	1-3-4-2-5-1 = 9
6	1-2-5-3-4-1 = 8	1-2-5-3-4-1 = 8	1-3-4-2-5-1 = 9	1-2-5-3-4-1 = 8

phéromone
fin itération 6

σ(7)	1	2	3	4	5
1	x	0.089	0.031	0.000	0.000
2	0.000	x	0.000	0.000	0.121
3	0.000	0.000	x	0.120	0.000
4	0.089	0.031	0.000	x	0.000
5	0.031	0.000	0.089	0.000	x

Itérations 7-8

Tour des fourmis

itération	Fourmi 1	Fourmi 2	Fourmi 3	Fourmi 4
7	1-2-5-3-4-1 = 8	1-2-5-3-4-1 = 8	1-2-5-3-4-1 = 8	1-2-5-3-4-1 = 8
8	1-2-5-3-4-1 = 8	1-2-5-3-4-1 = 8	1-2-5-3-4-1 = 8	1-2-5-3-4-1 = 8

Toutes les fourmis font le tour optimal !

phéromone fin itération 8

σ(9)	1	2	3	4	5
1	x	0.124	0.000	0.000	0.000
2	0.000	x	0.000	0.000	0.124
3	0.000	0.000	x	0.124	0.000
4	0.124	0.000	0.000	x	0.000
5	0.000	0.000	0.124	0.000	x

Métaheuristiques - Conclusion

- Méthodes générales s'appliquant à n'importe quel problème d'optimisation discrète
- Fournissent de bonnes solutions approchées en général
- Inconvénients :
  - réglage des paramètres
  - Garantie de la qualité de la solution ne peut se faire que a posteriori à l'aide de bornes