Le voyageur de commerce

Algorithme “branch and bound”, Algorithme Glouton,

Méthode de recherche locale

Michel Van Caneghem

Décembre 2002

Le problème

Trouver le trajet de longueur minimale passant par toutes les villes et revenant au point de départ (distance euclidienne)

Recherche exhaustive

Il suffit de construire tous les chemins possibles et de calculer leurs longueurs. Avec n villes il y a $(n-1)!/2$ chemins possibles. Avec 36 villes on trouve :

516657398319307246483332566876160000000 0

Cette méthode est donc inutilisable, sauf si le nombre de villes est très petit.

Les méthodes

Méthodes exactes	**"Branch and bound" [*record 13509 villes*], approximations garanties.**
Méthodes approchées	**Algorithme gloutons, Méthodes de descente, Recuit simulé, Algorithme tabou, Algorithmes génétique, Algorithme de Lin et Kerningham, Colonies de fourmis, [*quelques millions de villes*].**

"Branch and bound"

En français : Algorithmes d’énumération par séparation et évaluation. Voyons un premier exemple (problème de set partitionning) :

$$min \quad z = 3x_1 + 7x_2 + 5x_3 + 8x_4 + 10x_5 + 4x_6 + 6x_7 + 9x_8$$

$$\begin{cases} x_1 + x_2 = 1 \\ x_3 + x_4 + x_5 = 1 \\ x_5 + x_6 + x_7 = 1 \\ x_7 + x_8 = 1 \\ x_2 + x_4 + x_6 = 1 \\ x_i = 0 \text{ ou } 1 \end{cases}$$

On peut explorer l’arbre complet (256 noeuds)

"Branch and bound" (2)

évaluation : (souvent le simplex)

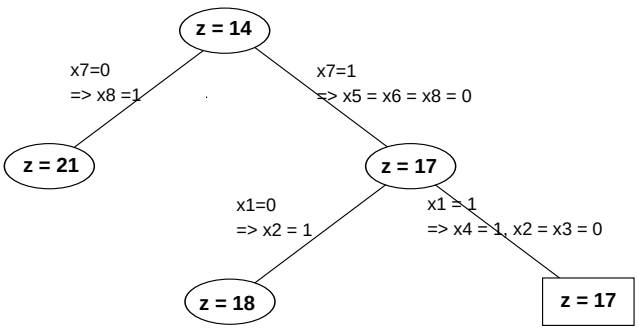
$$\begin{aligned} z &= 3x_1 + 7x_2 + 5x_3 + 8x_4 + 10x_5 + 4x_6 + 6x_7 + 9x_8 \\ &= 3(x_1 + x_2) + 5(x_3 + x_4 + x_5) + 4(x_5 + x_6 + x_7) + \dots \\ &= 14 + 4x_2 + 3x_4 + x_5 + 7x_8 \end{aligned}$$

On en déduit donc que $\min(z) \geq 14$

séparation : (choix d’une variable)

Variables de coût nul	Pénalité associée
$x_1 = 0$	**4 car** $x_2 = 1$
$x_3 = 0$	**min(3,1) = 1 car** $x_4 = 1$ **ou** $x_5 = 1$
$x_6 = 0$	**min(4,3) = 3 car** $x_2 = 1$ **ou** $x_4 = 1$
$x_7 = 0$	**7 car** $x_8 = 1$

"Branch and bound" (3)



"Branch and bound" (4)

On veut minimiser $z = f(x), x \in \Omega$ ensemble des solutions réalisables (qui satisfont les contraintes). Il faut donc définir :

- **Une procédure de séparation : qui va choisir la prochaine variable à énumérer (pénalité, taille du domaine, ...)**
- **Définir un minorant de z sur $S \subset \Omega$ sous-ensemble de solution (On utilise le simplex ou des relaxations lagrangiennes).**

L’objectif est de faire la plus grande coupure possible dans l’arbre de recherche. *Cette méthode est donc une méthode exacte*

Algorithme de Little

Revenons à notre voyageur et examinons ce principe sur un exemple plus petit (Algorithme de Little 1963) : Un voyageur doit visiter successivement chacune des 6 villes suivantes : A, B, C, D, E, F et revenir à son point de départ.

	A	B	C	D	E	F
A	∞	1	7	3	14	2
B	3	∞	6	9	1	24
C	6	14	∞	3	7	3
D	2	3	5	∞	9	11
E	15	7	11	2	∞	4
F	20	5	13	4	18	∞

Algorithme de Little (2)

Evaluation

On soustrait de chaque ligne le plus petit élément et on fait de même pour les colonnes. D'ou une première évaluation minimale de la longueur du chemin : 16 = 13 + 3.

	A	B	C	D	E	F
A	∞	0	3	2	13	1
B	2	∞	2	8	0	23
C	3	11	∞	0	4	0
D	0	1	0	∞	7	9
E	13	5	6	0	∞	2
F	16	1	6	0	14	∞

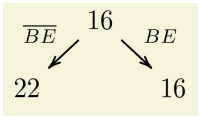
Algorithme de Little (3)

Séparation : On essaye de choisir au mieux si un couple de deux villes est ou n'est pas dans le chemin optimal. Ex : Si un chemin contient AB le cout augmente de 0. Si un chemin ne contient pas AB (noté $\overline{AB}$) le cout est de 1 + 1 = 2. On obtient le choix de BE de cout 6.

	A	B	C	D	E	F
A	∞	0(2)	3	2	13	1
B	2	∞	2	8	0(6)	23
C	3	11	∞	0(0)	4	0(1)
D	0(2)	1	0(2)	∞	7	9
E	13	5	6	0(2)	∞	2
F	16	1	6	0(1)	14	∞

Algorithme de Little (4)

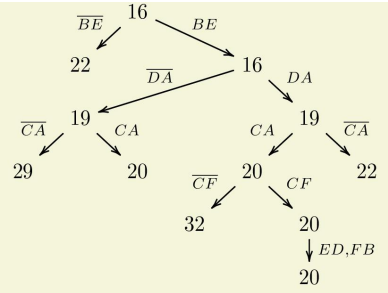
Le premier arbre



	A	B	C	D	F
A	∞	0(2)	3	2	1
C	3	11	∞	0(0)	0(1)
D	0(3)	1	0(3)	∞	9
E	13	∞	6	0(2)	2
F	16	1	6	0(1)	∞

Algorithme de Little (5)

L'arbre final



Le chemin est : F, B, E, D, A, C, F : Cout 20

Algorithme de Little (6)

Quand on exclu un couple du circuit

Par exemple nous avons à exclure DA du circuit. Il suffit de rendre son coût infini. Nous avons alors la matrice suivante :

	A	B	C	D	F
A	∞	0	3	2	1
C	3	11	∞	0	0
D	∞	1	0	∞	9
E	13	∞	6	0	2
F	16	1	6	0	∞

Une approximation exacte

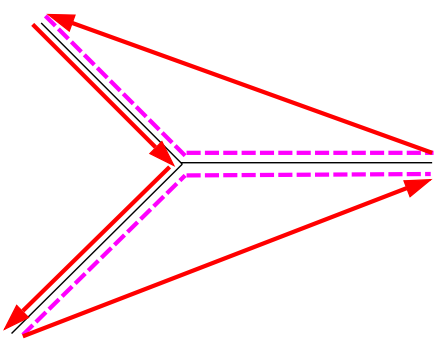
On part de l'arbre de recouvrement minimal du graphe construit à l'aide de la méthode de Prim ou de Kruskal (complexité $n \log n$)

On Construit un chemin à partir de cet arbre R en doublant chaque arête et en court-circuitant les sommmets déjà rencontrés dans le parcours. Soit Z_{opt} la longueur du plus cours chemin et Z_{app} la longueur construite à partir de l'arbre. Alors $Z_{opt} > Z_{opt} - e > lg(R)$ et $2lg(R) > Z_{app}$, donc ;

$$Z_{opt} < Z_{app} < 2 * Z_{opt}$$

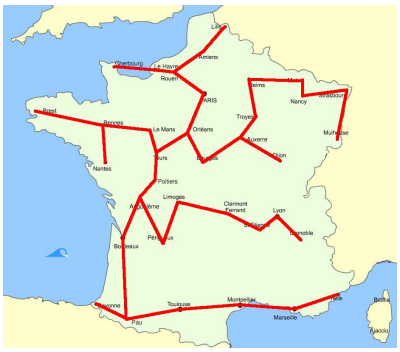
Il y a une meilleure approximation.

Une approximation exacte (2)



Une approximation exacte (3)

lg(R) = 3883 Km
Zapp = 6069Km
Zopt = 4441Km



Un algorithme glouton

C'est un algorithme qui fait un choix à chaque étape, sans jamais remettre en cause ce choix.

Algorithme du plus proche voisin (Nearest Neighbour) : On choisit une première ville au hasard et on construit un chemin en allant vers la ville la plus proche n'appartenant pas déjà au chemin.[demo]

Cet algorithme donne de meilleures solutions que la solution approchée précédente, mais n'est pas très bon. Il est parfois utilisé comme point de départ d'autres méthodes. (*Remarque : pour d'autres problèmes il y a parfois de très bons algorithmes gloutons*)

Recherche locale

Les méthodes de recherche locale partent d'une configuration initiale et appliquent successivement des transformations à la solution courante tant qu'un critère d'arrêt n'est pas vérifié. Leur mise en œuvre nécessite donc le choix :

- **d'une (ou plusieurs) solution initiale ;**
- **d'une (ou plusieurs) transformation locale, on parle aussi de mouvement**

Méthode de descente

Cette méthode est très simple : elle consiste à se déplacer dans l'espace de recherche en choisissant toujours la meilleure solution parmi le voisinage de la solution courante (Hill Climbing).

- ❑ ***Générer une solution initiale*** s **.**
- ❑ ***boucle d'itération***

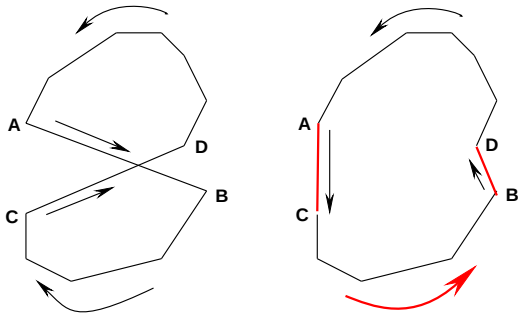
① **Générer le voisinage de** $V(s)$ **de** s**.**

② **Trouver** $s' \in V(s)$ **tel que**

$$f(s') = \min_{s \in V(S)} f(s).$$

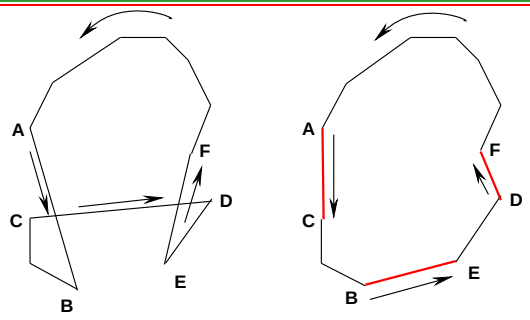
③ **si** $f(s') < f(s)$ **alors** $s := s'$ **et aller en ① sinon FIN.**

Voisinage 2 opt



Si lg(AB) + lg(CD) > lg(AC) + lg(BD)

Voisinage 3 opt



Si lg(AB) + lg(CD) + lg(EF) > lg(AC) + lg(BE) + lg(DF)

Remarque sur 2 opt et 3 opt

Ces voisinages ont été définis par Lin (1965).

2 opt : Nb de voisinages = $n(n-3)/2$

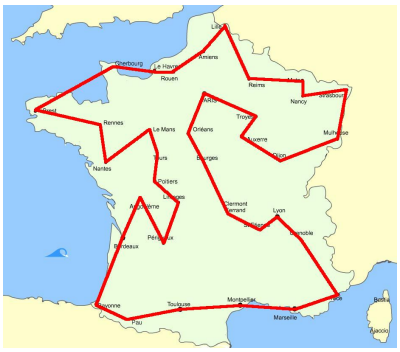
3 opt : Nb de voisinages = $n(n-3)(n-2)$

En général on part d'un chemin aléatoire (ou avec heuristique du plus proche voisin) et on applique une transformation tant que l'on peut. Quand on a fini, on peut recommencer...

En général *3 opt* est plus efficace que *2 opt*, mais est plus couteux en temps, ce qui fait que par la suite on va le plus souvent utiliser *2 opt*. [demo1], [demo2]

et en France ?

Zopt = 4441,3Km



Recherche locale (2)

L' existence de minima locaux, impose l'utilisation de méthodes d'exploration efficaces pour éviter de rester bloqué aux alentours de ces minima. Plusieurs méthodes ont été proposées et ont souvent été inspirées par des phénomènes naturels :

- les *algorithmes génétiques* font référence à la sélection, la mutation et le croisement des individus au sein d'une même espèce biologique ;
- le *recuit simulé* est basé sur les principes d'équilibre énergétique lors de la cristallisation des métaux ;
- la *méthode tabou* introduit la notion d'histoire (mémoire) dans la stratégie d'exploration des solutions.

Algorithmes génétiques

Les algorithmes génétiques forment une famille très intéressante d'algorithmes d'optimisation. Ils ont été proposés il y a une vingtaine d'années par J.H Holland ("Adaptation in Natural and Artificial Systems", 1976)

Leur fonctionnement est calqué sur les critères de sélection naturelle. Partant d'une population initiale constituée de solutions admissibles, on produit itérativement plusieurs générations. A chaque étape, afin de garder une population de taille raisonnable, on ne conserve que les meilleurs individus (solutions).

Algorithmes génétiques (2)

❑ *Initialisation :* Générer un ensemble de solutions initiales.

❑ *Boucle Principale*

① Evolution :
- Sélection : Choisir avec une probabilité proportionnelle à leur qualité une liste d'individus.
- Reproduction : Générer à partir de cette liste de nouveaux individus à l'aide des opérateurs génétiques
- Remplacement : Eliminer avec une probabilité inversement proportionnelle à leur qualité certains individus.

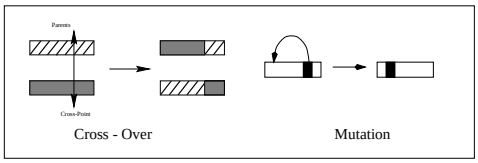
② Réactualisation de la meilleure solution

③ Allez en ① tant que Nombre de Générations $\leq$ Valeur pré-déterminée

La reproduction

La mutation permet de diversifier les solutions en altérant de façon non déterministe un individu.

Le cross-over consiste à générer à partir de deux individus, deux nouvelles solutions composées chacune d'une partie des caractéristiques de leurs parents.



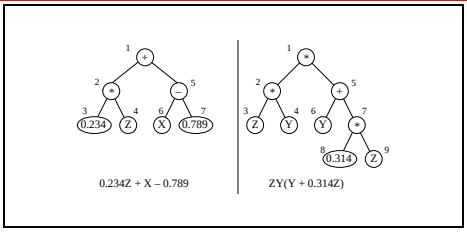
La programmation génétique

"Genetic programming" : il s'agit d'algorithmes génétiques travaillant sur une population de "programmes". On peut dire qu'il s'agit d'une programmation automatique.

On représente un programme par un arbre. L'opération de cross-over correspond à partir de 2 arbres (les parents) à tirer au hasard, un noeud dans le premier et dans le deuxième arbre et à permuter les sous arbres correspondant.

http://www.genetic-programming.org
http://www.smi.stanford.edu/people/koza

La programmation génétique (2)



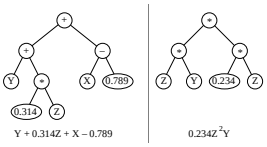
Parent 1:

```
(+ (* 0.234 Z) (- X 0.789))
```

Parent 2:

```
(* (* Z Y) (+ Y (* 0.314 Z)))
```


La programmation génétique (3)



Offspring 1:

```
(+ (+ Y (* 0.314 Z)) (- X 0.789))
```

Offspring 2:

```
(* (* Z Y) (* 0.234 Z))
```


Un joli cristal en métallurgie

Quand on chauffe un métal, il passe à l'état liquide c'est à dire : beaucoup de liberté pour les atomes.

Quand on abaisse la température il y a donc moins de degré de liberté, puis si l'on refroidit encore on passe à l'état solide. Suivant la manière dont la température baisse on obtient différents solides :

- Baisse brutale (trempe) : on obtient un "verre" structure amorphe (minimum local d'énergie).
- On abaisse progressivement la température pour avoir un minimum absolu d'énergie et donc un cristal.
- Si la baisse est trop rapide, il y a des défauts. Le "recuit" permet, si l'on réchauffe un peu, de redonner de la liberté aux atomes.

Le recuit simulé

Méthode découverte en 1982 par S. Kirkpatrick à partir de la méthode de Métropolis (1953) qui remarque que le problème des minima locaux est de même nature dans les deux cas.

Cela peut se modéliser à l'aide de la mécanique statistique : la distribution de Bolztmann mesure la probabilité $P(X)$ de visiter l'état X en fonction de son énergie $E(X)$ et de la température T :

$$P(X) = e^{-\frac{E(X)}{kT}}/N(T)$$

k est bien sur la constante de Bolztmann.

Le recuit simulé(2)

① **Calculer une solution initiale $k := 0$; et initialiser la température T_0.**
② **Modifier la solution courante à l'aide d'une transformation simple. Calculer son coût Δf.**
③ **Si $\Delta f > 0$ alors tirer un nombre aléatoire p entre 0 et 1. Si $p > \exp(-\Delta f/T_k)$ aller en ⑤ (la transformation est rejetée).**
④ **Adopter la nouvelle solution comme solution courante.**
⑤ **Si "l'équilibre statistique est atteint" alors incrémenter k et "abaisser" la température : $T_k = g(T_{k-1})$.**
⑥ **Si le système n'est pas "gelé" aller en ②.**
⑦ **FIN.**

Le recuit simulé(3)

Il faut quelques explications :

✔ **Solution initiale aléatoire ou solution obtenue avec un algorithme glouton pour démarrer avec une température initiale basse.**
✔ **Température initiale : difficile à choisir (j'ai pris $T_0 = 80$)**
✔ **Décroissance de T : $T_k = \mu T_{k-1}$ avec $0,5 < \mu < 0,9$. J'ai pris $\mu = 0.9$.**
✔ **Equilibre statistique : fixer le nombre d'itérations ou le nombre de transformation acceptées (J'accepte 1000 transformations).**
✔ **Gel du système : Fixer la température finale ($T = 5$).**

Le recuit simulé(4)

Voici des résultats :

```
Temp:  80.00,  Long: 15739.23 km
Temp:  72.00,  Long:  7331.35 km
Temp:  64.80,  Long:  8344.08 km
Temp:  58.32,  Long:  6970.56 km
Temp:  52.49,  Long:  5905.03 km
Temp:  47.24,  Long:  6205.65 km
...
Temp:  13.34,  Long:  4441.30 km
Temp:  12.01,  Long:  4458.37 km
Temp:  10.81,  Long:  4479.66 km
Temp:   9.73,  Long:  4441.30 km
```


Le recuit simulé(5)

Méthode délicate à utiliser, mais c'est la seule qui arrive à donner une solution pour des problèmes à 10000 villes.

Il semble que cette méthode soit utilisée pour :

✘ **l'optimisation combinatoire (ordonnancement)**
✘ **la CAO (conception de circuits, placement de composants)**
✘ **Le traitement d'images (restitution d¨images brouillées)**

➥ **Cette méthode est très utilisée dans les milieux industriels.**

Méthode de recherche Tabou

Cette méthode a été présentée pour la première fois par F. Glover ("Future Paths for Integer Programming and Links to Artificial Intelligence", 1986).

L'idée de base consiste à introduire la notion d'histoire dans la politique d'exploration des solutions. Le nom de baptême de *recherche tabou* donné en 1986 par Glover exprime l'interdiction de reprendre des solutions récemment visitées.

Un mouvement permet de passer d'une solution valide à une autre. On dit que la nouvelle solution est un voisin de la précédente. L'ensemble des solution que l'on peut atteindre à partir d'une solution s'appelle le voisinage.

Méthode de recherche Tabou (2)

A chaque itération, tous les mouvements possibles sont examinés et le "meilleur", ou plus exactement le moins mauvais est sélectionné.

Comme cette manière d'agir peut cycler c'est-à-dire répéter indéfiniment la même suite de mouvements, les t derniers mouvements effectués sont considérés comme interdits ("tabous"), t représentant la taille de la liste tabou. Le mouvement effectivement choisi à chaque itération, est donc le meilleur mouvement non tabou.

Méthode de recherche Tabou (3)

❑ ***Initialisation***

① **Trouver une solution initiale x_0. $x^* \leftarrow x_0$, $c^* \leftarrow f(x_0)$. x^* est la meilleure solution rencontrée, c^* sont coût et f la fonction économique.**
② **$0 \leftarrow k$, ListeTabou = $\emptyset$**

❑ ***Répéter tant qu'un critère de fin n'est pas véfifié***

① **Choisir parmi le voisinage de $x_k, V(x_k)$, le mouvement qui minimise f et qui n'appartient pas à ListeTabou, $meilleur(x_k)$.**
② **$x_{k+1} \leftarrow meilleur(x_k)$.**
③ **si ($c(x_{k+1}) < c^*$) alors $x^* \leftarrow x_{k+1}$, $c^* \leftarrow c(x_{k+1})$.**
④ **Mise à jour de ListeTabou.**

Méthode de recherche Tabou (4)

Diverses améliorations :

stratégie d'intensification : On mémorise les meilleures solutions rencontrées et l'on essaie d'en dégager quelques propriétés communes pour définir des régions *intéressantes* vers lesquelles on oriente la recherche (par exemple en rendant Tabou tous les mouvements qui font sortir de cette région).

stratégie de diversification : C'est le contraire. L'application de cette politique conduit à mémoriser les solutions les plus fréquemment visitées et imposer un système de pénalités, afin de favoriser les mouvements les moins souvent utilisés.

Méthode de recherche Tabou (5)

Aspiration : Il arrive qu'un mouvement Tabou, donc en principe interdit, se révèle intéressant. Le critère d'aspiration le plus fréquenment utilisé et le plus simple, consiste à regarder si le mouvement considéré ne conduit pas à une solution de coût inférieur à celui de la meilleure solution rencontrée jusqu'à présent.

Taille de la liste Tabou : La taille t de la liste tabou est à déterminer empiriquement, elle varie avec les problèmes, mais c'est une donnée primordiale. En effet une liste trop petite peut conduire à un cycle , alors qu'une liste trop grande peut interdire des transformations intéressantes (On trouve des règles statiques et des règles dynamiques).

Méthode de recherche Tabou (6)

Selection du meilleur voisin : La Politique du *Best Fit*, et la Politique du *First Fit* (on sélectionne le premier voisin qui satisfait les contraintes tabous. Cette politique est souvent utilisée lorsque la taille du voisinage ne permet pas d'effectuer une évaluation complète).

Critères d'arrêt : Le critère d'arrêt sert à déterminer le moment où l'on considère que la solution trouvée est d'assez bonne qualité pour être recevable.

➥ La méthode tabou, plus jeune que le recuit simulé donne quand elle est bien appliquée de meilleurs résultats. Elle est moins sensible aux paramètres de réglage.

Le Devoir 2

Etant donné m entrepôts et n clients. Le problème de la localisation d'entrepôts sans capacité ("UWLP : Uncapacited warehouse location problem") consiste à choisir un sous-ensemble d'entrepôts qui minimise le coût fixe des entrepôts (f_i) et le coût de transport entre les entrepôts et les clients (c_{ij}) [*Chaque client est affecté à l'entrepôt le plus proche*). Je vous propose de programmer une méthode tabou pour la résolution approchée de ce problème sur des grandes tailles.

Vous vous baserez sur l'article de Pascal Van Hentenryck : *A Simple Tabu Search for Warehouse Location* :

`http://www.cs.brown.edu/people/pvh/trwh.pdf.`

Le Devoir 2 (2)

- **On représentera une configuration par le vecteur booléen Y de taille m, Y_i est vrai si l'entrepôt i est choisi.**
- **On part d'une configuration aléatoire.**
- **Le voisinage est les m configurations ou on prend chaque fois le complémentaire d'un Y_i.**
- **Attention, pour chaque voisin, on recalculera à chaque fois le coût total. (je ne vous demande pas de faire le calcul incrémental du coût, comme cela est décrit dans l'article).**
- **Vous vous limiterez aux 5 exemples disponibles sur la page web.**

Le Devoir 2 (2)

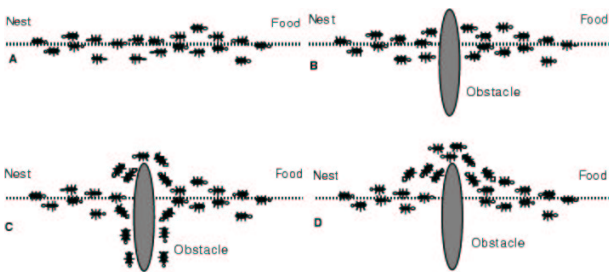
```
public static void tabou() {
    // On tire une solution au hasard
    for (int i = 0; i < M; i++) { y[i] = alea.nextBoolean(); tabou[i] = -1;}
    // Debut de la boucle
    int nbStable = 0; int it = 0; double obj = cout(y); double best = obj;
    while (nbStable < 500) {
        // On parcourt le voisinage
        for (int i = 0; i < M; i++) if (tabou[i] < it) {...} ==> bestObj
        //Est-ce que l'on n'a pas détérioré la solution
        if (bestObj <= old) {
            // On choisit un élément au hasard parmi les meilleurs : w
            // On met à jour
            obj = bestObj;
            // L'element w devient tabou
            tabou[w] = it + tLen;
        } else
            // Pas de bonne solution : on va diversifier en fermant un entrepot au hasard
        if (obj < best) nbStable = 0; else nbStable++;
    }
}
```


Le Devoir 2 (3)

cap41.txt

```
Solution initiale = 1130897.7375000003
bestSol = 1005323.3625,        it = 1, t = 2ms
bestSol = 973999.6625,         it = 2, t = 5ms
bestSol = 965453.2625000001,   it = 3, t = 8ms
bestSol = 957679.2375,         it = 4, t = 11ms
bestSol = 950998.0375000001,   it = 5, t = 15ms
bestSol = 946588.6625000001,   it = 6, t = 18ms
bestSol = 942665.5625,         it = 7, t = 20ms
bestSol = 938947.7874999999,   it = 8, t = 24ms
bestSol = 938222.0374999999,   it = 9, t = 27ms
bestSol = 935152.2874999999,   it = 10, t = 29ms
bestSol = 934199.1375,         it = 11, t = 32ms
bestSol = 932615.75,           it = 12, t = 38ms
C'est terminé
bestSol = 932615.75,           it = 269, t = 146ms
```


Les fourmis



Consultez la page Web :

http://iridia.ulb.ac.be/dorigo/ACO/ACO.html

Les fourmis (1)

On place des fourmis sur différentes villes et chaque fourmi à son tour choisit une nouvelle ville de manière probabiliste en fonction de :

- **La quantité de phéromone (!!)**
- **Une fonction de la distance à la prochaine ville**
- **Sa mémoire interne qui lui indique les villes déjà parcourues.**

Chaque fois qu'une fourmi se déplace elle laisse une trace (local trail updating). Celle qui a trouvé le chemin de longueur minimum laisse une trace plus importante (global trail updating).

Les colonies de fourmis

Soit $Dist(i,j)$ la distance entre les villes i et j et $Phero(i,j)$ la quantité de phéromone sur le chemin entre ces deux villes, N le nombre de villes et M le nombre de fourmis.

```
Initialisation
for iter:=1 to NbIter
    for i:=1 to N
        for k:=1 to M
            choix de la prochaine ville pour la fourmi k
            dépot de la trace locale
        endfor
    endfor
    Choix de la fourmi ayant le plus court chemin
    dépot de la trace globale
endfor
```


Les colonies de fourmis (2)

Initialisation

1. **Soit τ_0 une quantité de phéromone élémentaire calculée par : $\tau_0 = 1/(N * L_{nn})$. La distance L_{nn} est la distance calculée par l'alorithme glouton du plus proche voisin (nearest neighbor). On initialise le tableau des traces avec : $Phero(x,y) = \tau_0$**
2. **On tire au hasard les M villes de départ pour les M fourmis.**

Les colonies de fourmis (3)

Le choix de la ville

Soit q un nombre aléatoire entre 0 et 1 et soit v_k la position de la fourmi k

Si $q \leq q_0$ alors : On choisit parmi les villes candidates u celle qui maximise :

$$\max_u \frac{Phero(v_k,u)}{Dist(v_k,u)^2}$$

Sinon : On choisit une ville au hasard selon la loi de probabilité suivante :

$$p_k(u) = \frac{Phero(v_k,u)/Dist(v_k,u)^2}{\sum_v Phero(v_k,v)/Dist(v_k,v)^2}$$

Les colonies de fourmis (4)

La trace locale est est calculée pour chaque couple (x,y) de ville parcourue par

$$Phero(x,y) = Phero(x,y)(1-\rho) + \rho\tau_0$$

La trace globale : Pour chaque couple de ville (x,y) appartenant au meilleur chemin trouvé on a :

$$Phero(x,y) = Phero(x,y)(1-\alpha) + \alpha/L_{meilleur}$$

ou $L_{meilleur}$ est la longueur du meilleur chemin trouvé.

Référence

Beaucoup de références sur Internet, la meilleur page sur le voyageur de commerce :

http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html

mais aussi :

- ***Modern Heuristic Techniques for Combinatorial Problems*, Edited by C.R. Reeves, Backwell Scientific Publications – 1993**
- ***Local search in Combinatorial Optimization*, E. Aarts et K. Lenstra, Wiley – 1997**