TPs : Métaheuristiques

Les exercices suivant peuvent être programmés dans *votre langage favori* ou dans *le langage qui vous semble le plus adapté* au sujet. L'assistance de l'enseignant privilégiera donc les questions liées à l'algorithmique et aux structures de données.

Exercice 1 ("Unconstrained Binary Quadratic Problem"). Un UBQP (Unconstrained Binary Quadratic Problem) a pour but de **minimiser** le résultat d'une fonction f pouvant s'écrire sous la forme :

$$f(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij}.X[i]X[j]$$

où Q est une matrice **symétrique** $n \times n$ de constantes et X est un n-uplet de variables binaires. On peut par exemple considérer la matrice Q suivante :

$$Q = \begin{pmatrix} -17 & 10 & 10 & 10 & 0 & 20 \\ 10 & -18 & 10 & 10 & 10 & 20 \\ 10 & 10 & -29 & 10 & 20 & 20 \\ 10 & 10 & 10 & -19 & 10 & 10 \\ 0 & 10 & 20 & 10 & -17 & 10 \\ 20 & 20 & 20 & 10 & 10 & -28 \end{pmatrix}$$

Pour la solution $X = \boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{0}$, $f(X)$ vaut 6.

Les UBQP sont reconnus pour leur capacité à représenter une grande variété de problèmes importants [?], qui vont de l'analyse financière à des problèmes combinatoires sur les graphes en passant par l'ordonnancement de tâches, l'allocation de fréquence etc.

Nous désirons trouver une solution X qui minimise la fonction $f(X)$ pour une matrice Q fournie dans un fichier. Pour cet exemple le fichier `partition6.txt` contient en premier la taille n (6) puis un nombre p (2) (on expliquera ce qu'il représente par la suite) puis les 6×6 éléments séparés par des blancs. Le fichier `graphe12345.txt` est un autre exemple.

Pour réaliser les premières questions vous pouvez créer un tableau Q contenant les valeurs données dans l'énoncé.

Question 1.1. Écrire une fonction qui renvoie *une solution initiale* au hasard pour ce problème, c'est-à-dire qui renvoie un vecteur de n bits tirés aléatoirement (la taille du vecteur sera paramètrable selon la valeur lue dans le fichier utilisé en entrée).

Question 1.2. Écrire une fonction qui *calcule la valeur* de cette solution par f dont Q est décrit dans le fichier en entrée.

Question 1.3. Programmer une fonction `meilleur_voisin` qui renvoie la *meilleure solution voisine* de X où un voisin X' de X est une séquence de bits qui ne diffère de X que par un seul bit.
Optimisation : S'il y a plusieurs meilleurs voisins votre fonction devra choisir aléatoirement parmi eux.

On rappelle l'Algorithme `Steepest Hill-Climbing` :

- `on part d'une solution` s
- $nb\_depl \leftarrow 0$; $STOP \leftarrow false$

`Repeat`

- $s' \leftarrow$ `meilleur_voisin(`s`)`
- `if meilleur(`$f(s'), f(s)$`)` `then` $s \leftarrow s'$
  `else` $STOP \leftarrow true$ `/* optimum local */`
- nb_depl++

`Until` $nb\_depl =$ `MAX_depl or` $STOP$
`Return(`s`)`

Question 1.4. Programmer la méthode du `Steepest Hill-Climbing`. Nous choisissons, ici, la version du Steepest Hill-Climbing dans laquelle "meilleur" signifie "strictement meilleur".

Question 1.5. Programmer une variante avec redémarrages, c'est-à-dire faire une boucle externe autour du `Steepest Hill-Climbing` permettant de partir d'une nouvelle solution tirée au hasard, puis d'effectuer un essai : c'est-à-dire faire au plus `MAX_depl` déplacements vers des meilleurs voisins (un essai peut s'arrêter avant d'avoir fait les `MAX_depl` déplacements vers des meilleurs voisins, puis redémarrer avec une nouvelle solution au hasard. Cette boucle externe devra être faite `MAX_essais` fois.

Question 1.6. Faîtes tourner vos programmes sur les deux fichiers `partition6.txt` et `graphe12345.txt`.

On rappelle la méthode *Tabou* :

- on part d'une solution s
- Tabou $\leftarrow []$
- $nb\_depl \leftarrow 0$; $msol \leftarrow s$; $STOP \leftarrow false$

Repeat
  - if voisins_non_Tabou(s) $\neq \emptyset$
    then $s' \leftarrow$ meilleur_voisin_non_Tabou(s)
    else $STOP \leftarrow true$ /* plus de voisin non tabou*/
  - Tabou $\leftarrow$ Tabou + $\{s\}$
  - if meilleur(s',$msol$) then $msol \leftarrow s'$ /* stockage meilleure solution courante*/
  - $s \leftarrow s'$
  - nb_depl++

Until $nb\_depl =$ MAX_depl or $STOP$
Return($msol$)

où la liste Tabou est implémentée en FIFO de taille k fixée.

Question 1.7. Programmer la méthode tabou en essayant différentes tailles pour la liste Tabou.

Question 1.8. On impose maintenant une contraintes sur les solutions, on cherche une séquence binaire dont la somme des bits est supérieure ou égale à p. Modifiez votre fonction meilleur_voisin afin qu'elle prenne en compte cette contrainte où p est le deuxième nombre donné dans le fichier décrivant le problème (pour `partition6.txt` ce nombre est 2, pour `graphe12345.txt` c'est 4). Testez le Steepest-Hill-Climbing avec redémarrage sous cette contrainte.

Exercice 2 (Voyageur de commerce). Le célèbre problème du Voyageur de commerce (Travelling Salesman Problem TSP) consiste à trouver une tournée passant par toutes les villes d'un ensemble donné et ayant la plus courte distance totale. Les n villes sont numérotées de 1 à n et leurs coordonnées sont données dans un fichier : chaque ligne a la forme `Id x y` : `Id` est le numéro de la ville, et `x` et `y` sont les coordonnées de la ville. La première ligne indique le nombre n de villes.

Une solution de ce problème est une séquence de n villes, elle représente une tournée partant du point (0,0) puis passant par chacune des villes de la séquence puis revenant au point (0,0). Pour simplifier, on assimilera la distance entre deux villes à leur distance euclidienne : $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Par exemple, en utilisant le fichier `tsp5.txt` décrivant les coordonnées de 5 villes, la solution

5	3	4	1	2

correspond à une tournée de longueur 263.88 km.

Le problème consiste à trouver une tournée de longueur minimale.

Question 2.1. Écrire une fonction qui renvoie *une solution initiale* au hasard pour ce problème. Comme les villes sont numérotées de 1 à n, on veut donc générer une permutation aléatoire de la séquence $1, 2, \ldots, n$, qui indiquera dans quel ordre visiter les n villes. Un algorithme pour faire cela est le suivant :

Algorithme : Génération d'une permutation aléatoire de $[1, 2, \ldots, n]$

1. $S \leftarrow [1, 2, \ldots, n]$;
2. pour i allant de 1 à n faire :
   (a) $j \leftarrow \text{random}(1, n)$ . . . . . *# on tire un entier aléatoire entre 1 et n*
   (b) échanger $S[i]$ et $S[j]$

Question 2.2. Écrire une fonction qui *calcule la valeur* de cette solution c'est-à-dire la distance que le voyageur de commerce doit parcourir en partant de (0,0) puis en passant par toutes les villes $S[1] \ldots S[n]$ et en revenant en (0,0).

Question 2.3. Programmer une fonction `meilleur_voisin` qui renvoie la *meilleure solution voisine* de S. On pourra étudier plusieurs notions voisinage, notamment :

2-swap : on échange deux villes ; pour toute paire i, j, avec $1 \leq i < j \leq n$, un voisin S' de S est défini en échangeant $S[i]$ et $S[j]$;

2-opt : on inverse un segment de la séquence S ; plus précisément, pour toute paire i, j, avec $1 \leq i < j \leq n$, un voisin S' de S est défini en inversant les villes du segment $S[i] \ldots S[j]$:

$$\begin{aligned} S = \quad & S[1], \ldots, S[i-1], S[i], S[i+1], \ldots, S[j-1], S[j], S[j+1], \ldots, S[n] \\ S' = \quad & S[1], \ldots, S[i-1], S[j], S[j-1], \ldots, S[i+1], S[i], S[j+1], \ldots, S[n]. \end{aligned}$$

Cela revient à échanger $S[i]$ et $S[j]$, mais aussi $S[i+1]$ et $S[j-1]$, $S[i+2]$ et $S[j-2]$, ...

(On constate expérimentalement que le voisinage 2-opt permet en général d'arriver à de meilleurs solutions que 2-swap, mais coûte plus cher en temps de calcul : on opère 1 échange de 2 villes avec 2-swap, alors qu'on fait de l'ordre de $(j-i)/2$ échanges avec 2-opt.)

Question 2.4. Utilisez la méthode du Steepest Hill-Climbing sur les fichiers `tsp5.txt` et `tsp101.txt` et les méthodes avec redémarrages en redémarrant `MAX_essais` fois. Pour chaque essai, vous devrez être capable d'afficher la solution initiale tirée au hasard, la solution atteinte et le nombre de déplacements effectués depuis la solution initiale jusqu'à la solution atteinte.

Question 2.5. Utilisez la méthode tabou en essayant différentes tailles pour la liste tabou et pour `MAX_depl`. De la même manière, vous devrez pouvoir afficher la solution initiale, la solution atteinte, le nombre de déplacements effectués jusqu'à la solution atteinte, la meilleure solution rencontrée, le contenu de la liste tabou à la fin de la recherche.

Références

[KGAR04] Gary A. Kochenberger, Fred W. Glover, Bahram Alidaee, and César Rego. A unified modeling and solution framework for combinatorial optimization problems. *OR Spectrum*, 26(2) :237–250, 2004.