Projet d'Optimisation Stochastique

Implémentation de l'algorithme du Recuit Simulé

Problème du Voyageur de Commerce
Document organique

Auteur :
Jeremie Bosom
Erwan Chaussy

Professeur :
M.Abdel Lisser

19 janvier 2014

1 Présentation générale de la structure du programme

Pour ce projet nous avons repris comme base le projet codé en cours de Programmation en Nombre Entier. Néanmoins, nous avons eu beaucoup de modifications à effectuer car nous n'avions pas implémenté le recuit simulé et que nous ne gérions que la lecture des fichiers ".tsp" formatés selon le format EUC_2D. Nous avons donc modifié notre programme de façon à ce que nous puissions lire les fichiers ".xml", tout en gardant la compatibilité avec les fichiers ".tsp" formatés au format EUC_2D ou CEIL_2D, la différence entre les deux formats étant minime. Voici notre code sous forme de diagramme UML.

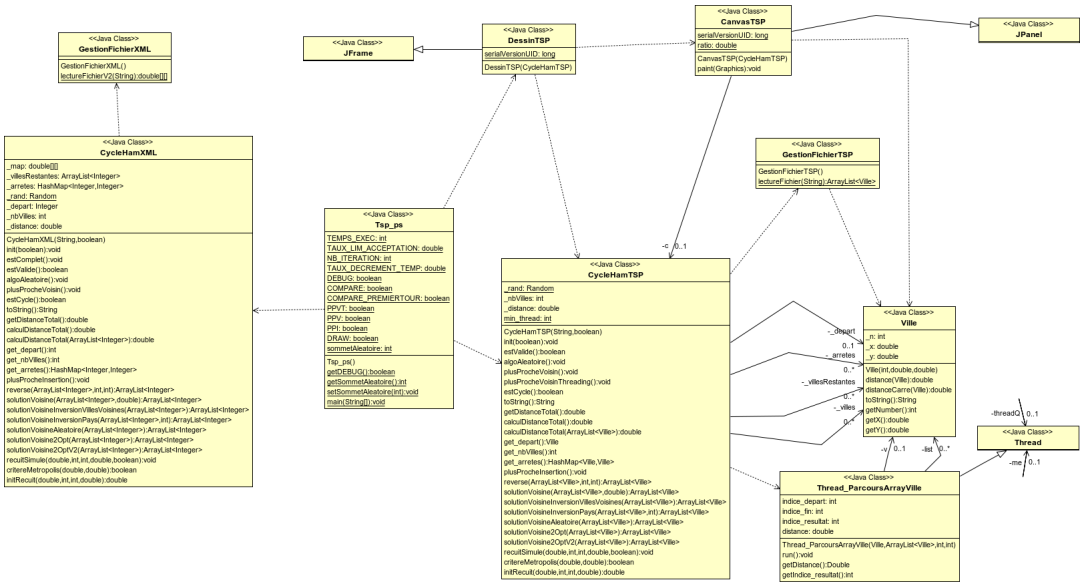

FIGURE 1 – Diagramme UML des classes de notre programme

2 Présentation des différentes classes

2.1 GestionFichierTSP

Cette classe permet de lire un fichier au format ".tsp". Elle contient uniquement une méthode statique qui, à partir du nom de fichier donné en paramètre, renvoie une arraylist de type Ville. S'il y a une erreur, l'arraylist renvoyée est vide. Les fichiers lus doivent être formatés selon le format de données de la TSPLIB et le jeu de donnée doit être au format EUC_2D ou CEIL_2D.

2.2 GestionFichierXML

Cette classe permet de lire un fichier au format ".xml". Elle contient uniquement une méthode statique qui, à partir du nom de fichier donné en paramètre, renvoie une matrice d'Integer. Nous avons choisi d'utiliser une matrice pour stocker nos données car nous avons affaire à des graphes qui sont tous complets

(exception faîtes du graphe "br17.xml"). Si il y a une erreur la matrice renvoyée est vide. Les fichiers lus doivent être formatés selon le format de données de la TSPLIB.

2.3 Ville

Cette classe permet de stocker une ville pour les fichiers ".tsp". Elle comprend les méthodes d'accès à ses différents éléments et une fonction qui calcule la distance euclidienne avec une ville voisine.

2.4 CycleHamTSP

Cette classe permet de construire et d'optimiser un cycle Hamiltonien. Elle appelle la méthode statique fournie par GestionFichierTSP afin de lire le document voulu et initialise le cycle hamiltonien. Elle contient trois méthodes pour former un cycle hamiltonien de base :

- Le Plus Proche Voisin (PPV) : *Cette méthode créer un cycle hamiltonien en reliant le point actuellement étudié à son plus proche voisin.*
- La Plus Proche Insertion (PPI) : *Cette méthode part d'un cycle hamiltonien constitué du point défini comme départ et son plus proche voisin. Elle ajoute au fur et à mesure tous les points ne faisant pas partie du cycle à celui-ci, en trouvant le point le plus proche appartenant au cycle.*
- Le Plus Proche Voisin Threadé (PPVT) : *Cette méthode est exactement la même que celle du Plus Proche Voisin, à la différence près que pour trouver le plus proche voisin du point étudié nous utilisons deux threads. Cela permet de gagner du temps à l'exécution sur les grosses instances, telle que celle de Mona Lisa. C'est cette méthode que nous utilisons par défaut.*

Cette classe permet aussi d'optimiser un cycle hamiltonien via l'algorithme du recuit simulé. Le calcul de la température est automatique. Nous utilisons un algorithme proche de celui du 2-opt pour trouver une solution voisine.

2.5 CycleHamXML

Cette classe, tout comme la classe CycleHamTSP, permet de construire et d'optimiser un cycle Hamiltonien. Elle appelle la méthode statique fournie par GestionFichierXML pour lire le document voulu et initialise le cycle hamiltonien. À la différence de CycleHamTSP, elle ne contient que deux méthodes pour former un cycle hamiltonien de base :

- Le Plus Proche Voisin (PPV) *Cette méthode créer un cycle hamiltonien en reliant le point actuellement étudié à son plus proche voisin. Nous utilisons cette méthode par défaut.*
- La Plus Proche Insertion (PPI) *Cette méthode part d'un cycle hamiltonien constitué du point défini comme départ et son plus proche voisin. Elle ajoute au fur et à mesure tous les points ne faisant pas partie du cycle à celui-ci, en trouvant le point le plus proche appartenant au cycle.*

Cette classe permet aussi d'optimiser un cycle hamiltonien via l'algorithme du recuit simulé. Le calcul de la température est automatique. Nous utilisons un algorithme proche de celui du 2-opt pour trouver une solution voisine.

2.6 Thread_ParcoursArrayVille

Cette classe a pour simple objectif la parallélisation de la recherche du plus proche voisin d'un point. Elle sert uniquement à l'algorithme du Plus Proche Voisin Threadé de la classe CycleHamTSP. Elle hérite de la classe Thread.

2.7 DessinTSP

Une fois le cycle hamiltonien construit, il est agréable de voir le dessin obtenue par nos algorithmes. Cette classe a pour objectif de dessiner le cycle hamiltonien obtenue pour les fichiers ".tsp". Elle hérite de la classe JFrame puis appelle la classe CanvasTSP pour dessiner le chemin du voyageur de commerce.

2.8 CanvasTSP

Cette classe dessine le cycle hamiltonien donné en paramètre. Elle hérite de la classe JPanel.

2.9 Tsp_ps

Cette classe contient la fonction principale *main*. Elle gère les options que nous avons implémentées et qui sont décrites dans le document utilisateur. Elle permet, entre autre, de spécifier des options pour tester l'influence des paramètres sur le recuit simulé, de comparer des fichiers entre eux et de tester si les fichiers peuvent être lus.