Bilan du Projet S3

Arthur Correnson

January 10, 2020

Introduction

Lors du semestre 3 de licence, il nous a été demandé d'implémenter en langage C un logiciel de résolution du **problème du voyageur de commerce**. Ce problème d'optimisation combinatoire est insoluble en grande dimension, mais plusieurs méthodes permettent d'approcher la solution optimale à moindre coût. Nous proposons ici quatre méthodes de résolution :

- **Force Brute** Recherche exhaustive du plus court chemin.
- **Algorithme génétique** Approche évolutive basée sur le croisement de solutions intermédiaires.
- **Plus Proche voisin** Méthode heuristique calculant de proche en proche le plus court chemin.
- **Marche aléatoire** Tirage au sort d'un chemin.

Notons qu'une méthode d'optimisation nommée **2 optimisation** a aussi été développée. Elle permet d'améliorer les résultats obtenus par les méthodes précédentes.

Programme rendu

La réponse au sujet prend la forme d'un programme C utilisable en ligne de commande. Il attend en entrée un fichier au format standard TSP décrivant le problème à résoudre et produit une solution au problème sous la forme d'un fichier **CSV**. Les quatre méthodes demandées ont été implémentées et vérifiée.

Structure des sources

Le code source est organisé selon une arborescence stricte :

- **docs/** : La documentation
- **scripts/** : Scripts utiles python (affichage graphique des résultats)
- **data/** : Exemples de fichiers TSP et des résultats associés.
- **src/** : contient le code source en C
- **src/tsplib/** : module d'interaction avec les fichiers au format TSP.
- **src/methods/** : implémentation des différentes méthodes de résolution
- **include/** : contient tous les en-têtes
- **include/tsplib** : en-têtes pour le module tsplib
- **include/methods** : en-têtes pour le méthodes de résolution
- **tests/** : dossier contenant les modules de tests unitaires.

Notons que la séparation des sources et des en-têtes est un choix de design utilisé afin de faciliter l'usage de notre programme à la fois en tant que logicielle, mais également en tant que bibliothèque utilitaire. Le programmeur souhaitant re-exploiter tout ou partie de ces sources pourra aisément importer les en-têtes de son choix en passant l'option -I include/ au compilateur C.

Compilation

Le *pipeline* de compilation a été écris en CMake, une alternative aux *Makefiles*, par soucis de clarté et de simplicité. Un *Makefile* principale placé à la racine permet cependant de contrôler toutes les tâches liées au projet.

commande	**utilisation**
make build	compilation de l'ensemble des sources
make run	lance le programme sur un petit exemple
make graph	affichage graphique des résultats de tests
make doc	génère la documentation avec Doxygen
make clean	Nettoie le dossier de build et les artefacts de compilation

Utilisation du programme

option	méthode
-bf	Force Brute
-bfm	Force Brute (optimisation matricielle)
-ga	algorithme génétique
-rw	marche aléatoire (*Random Walk*)
-ppv	plus proche voisin (*Nearest Neighbour*)
-2opt	application de la **2-optimisation**

Points Forts du Projets

Au cours du développement du projet, l'accent à été mis sur plusieurs points. En particulier, une grande importance a été accordée à la **fiabilité**, la **généricité**, la **maintenabilité**, la **documentation** ainsi que la qualité des **entrées/sorties**.

Fiabilité

Certains algorithmes (ou portion d'algorithmes) de résolution du problème du voyageur de commerce sont assez délicats, de leur bonne implémentation dépend la qualité des résultats obtenus ou même la terminaison du programme. Aussi, les portions les plus complexes d'algorithmes ont été développée avec le soucis d'être correcte et vérifiée. Cette vérification passe par des tests unitaires, mais également par l'utilisation d'assertions. Par ailleurs, les algorithmes ont été travaillé manuellement avant d'être implémenté dans une version en langage C. Parmi les portions vérifiées du programme, on trouve **l'algorithme de calcul des permutations** utilisée pour la méthode Force Brute (la preuve a été faite manuellement), et l'algorithme de **croisement DPX** utilisé pour l'algorithme génétique. De même, la cohérence de l'ensemble des méthodes de résolutions a été testée sur des jeux d'exemples (notamment, la vérifications que les réponses obtenus sont bien des permutations).

Généricité

Une mauvaise utilisation du programme ou le nom respect des formats de données ne doit pas causer le plantage du programme. Aussi, la lecture des entrées a fait l'objet d'un traitement exhaustif. Un *parser* complet a été implémenté pour lire les paramètres de la ligne de commande ainsi que les

fichiers **TSP**. En cas de non respect des formats, un message d'erreur précis est affiché.

```
[arthur] ~/Documents/github/projet_S3 (master) $ ./build/Tsp data/trash.tsp
[cli - error] invalid option data/trash.tsp
[arthur] ~/Documents/github/projet_S3 (master) $ ./build/Tsp -f data/trash.tsp
[cli - error] no method provided
[arthur] ~/Documents/github/projet_S3 (master) $ ./build/Tsp -f data/trash.tsp -bf
[Error] violation of TSP format : field DISPLAY_DATA_SECTION or NODE_COORD_SECTION missing
[arthur] ~/Documents/github/projet_S3 (master) $
```

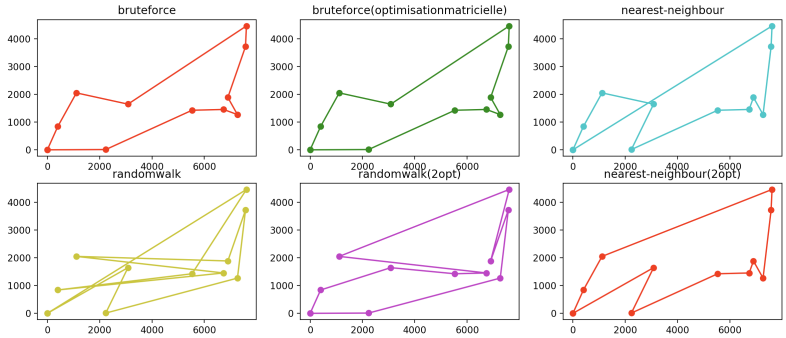
Maintenabilité

La maintenabilité est la facilité d'un projet à être administré, modifié, corrigé et partagé. Ici, le code source a été écris avec une attention particulière portée à la clarté et à la lisibilité du code. Ainsi, il est relativement aisé de se plonger dans la compréhension du code source et donc d'appliquer des éventuels correctifs ou modifications. De plus, un formateur de code a été utilisé pour s'assurer que le code satisfait à des standards strictes en terme de disposition et de lisibilité.

Entrées/sorties

La qualité des entrées sorties est une chose très importante. Le logiciel demandé s'utilise en ligne de commande, l'absence de retour graphique implique donc un soin d'autant plus grand des interactions avec l'utilisateur (puisque pas d'interface graphique). Aussi, les informations affichées par le programme sont formatées et misent en couleur. C'est un détails simple à mettre en place mais qui améliore grandement l'expérience utilisateur. De plus, un script d'affichage graphique des solutions est fournis avec le programme. Cela permet de visualiser les résultats produits par le logiciel.

```
Method Brute Force running...
Method Brute Force (with distance matrix) running...
Method Nearest Neighbour running...
Method Random Walk running...
Applying 2-optimization (nearest-neighbour)...
Initial solution :0 4 10 5 2 3 1 9 8 7 6 (24185.799306)
Optimized solution :0 4 10 6 7 8 9 1 3 2 5 (23778.051247)
1.69 % optimization
Applying 2-optimization (random-walk)...
Initial solution :0 6 3 4 1 10 9 7 8 2 5 (50401.740234)
Optimized solution :0 4 5 3 1 10 6 9 7 8 2 (34482.206210)
31.59 % optimization
```



Résultats

Les résultats obtenus sur les problèmes **att10**, **att20** et **att48** (trois problèmes de la bibliothèque TSPLIB) sont convaincant. La méthode génétique est de loin la meilleure, elle donne les plus petites longueurs en comparaison avec les approches **Plus Proche Voisin** et **Marche Aléatoire**.

problème	**plus proche voisin + optimisation**	**génétique**
att10	23778.05	21088.99
att20	26634.01	24113.94
att48	42084.34	39719.07

Conclusion