Emilie LHOMME
Thomas OUSTRIC
Laura MABRU

Rapport – Voyageur de Commerce (TSP)

Sommaire

I. Présentation du projet

1) Objectifs

L'objectif du problème est de concevoir une méthode heuristique permettant de résoudre des problèmes de TSP et d'essayer de trouver une solution optimale.

Nous avons à notre disposition un frameworklogiciel en Java : un ensemble de librairies et classes permettant de lire une instance du TSP et d'en visualiser la solution. Il nous faut y ajouter les algorithmes de résolution. Nous comparons alors les différentes solutions trouvées grâce aux 10 instances test : 10 jeux de données représentant différentes versions du problème à résoudre.

La seule contrainte : la méthode ne doit pas dépasser la limite de temps (60secondes).

2) Choix de méthode de résolution

Il existe deux approches pour résoudre ce problème d'optimisation : exacte ou approchée. Le but derrière la création de cet algorithme est de pouvoir l'utiliser avec un fichier de plus de 100 données (exemple : trouver le plus court chemin entre toutes les villes de France). Rien qu'avec 100 villes, cela fait plus de *100!* possibilités de chemin sachant que nos ordinateurs ne dépassent pas une certains nombres d'opérations par seconde.

Nous avons choisi de commencer par l'algorithme **du plus proche voisin** en algorithme de construction que nous améliorons ensuite avec une **recherche locale** ou **2-opt méthode** sur la solution trouvée.

En vue du temps qu'il nous restait suite à l'implémentation de ces deux algorithmes, nous sommes partis sur **l'algorithme des colonies de fourmis**.

II. Implémentation d'algorithme de construction

1) Algorithme du plus proche voisin

Principe de l'algorithme :

- Fixer une ville de départ (qui sera aussi la ville d'arrivée)
- Trouver la ville la plus proche non visitée et y aller
- Continuer tant qu'il y a des villes non visitées
- Une fois toutes les villes parcourues, retourner à la ville de départ

Avantages : solution rapide et simple à coder

Inconvénients : solution peu performante.

Conclusion : la solution trouvée étant non optimale, on construira des algorithmes d'amélioration par la suite en utilisant cette solution pour pouvoir l'améliorer (algorithme de recherche locale sur la solution par exemple).

2) Algorithme des colonies de fourmis

En observant une colonie de fourmis à la recherche de nourriture dans les environs du nid, on remarque qu'elle résout des problèmes tels que celui de la recherche du plus court chemin. On simule ainsi leur comportement au travers d'algorithmes.

Pour mieux expliquer l'algorithme, il est bon de savoir que les fourmis déposent au sol des phéromones lorsqu'elles marchent du nid à la source de nourriture et vice-versa (le premier chemin étant aléatoire). Les phéromones permettent donc de créer une « piste chimique », sur laquelle les fourmis s'y retrouvent. En effet, d'autres fourmis peuvent détecter les phéromones grâce à des capteurs sur leurs antennes.

Les phéromones ont un rôle de marqueur de chemin : quand les fourmis choisissent leur chemin, elles ont tendance à choisir la piste qui porte la plus forte concentration de phéromones. Cela leur permet de retrouver le chemin vers leur nid lors du retour.

Utilisons alors ce comportement pour notre algorithme du TSP.

A tout instant t, chaque fourmi choisit une ville de destination selon un choix défini. Toutes les fourmis se placent à l'instant t+1 dans une ville de leur choix. On appelle une itération de l'algorithme *ant system*, l'ensemble de déplacements de l'ensemble de la colonie entre l'instant t et l'instant t+1. Ainsi après n itérations, l'ensemble de la colonie aura effectué un circuit hamiltonien sur le graphe.

Maintenant, on précise que chaque fourmi a une mémoire implémentée par une liste de villes déjà visitées. Cela permet de garantir qu'aucune fourmi ne visitera deux fois une même ville au cours de sa recherche. La mémoire des fourmis est vidée lorsqu'elles ont terminé leur cycle.

Ensuite, une fourmi k, placée sur la ville i à l'instant t, va choisir sa ville j de destination en fonction de la visibilité de cette ville et de la quantité de phéromones déposée sur l'arc reliant ces deux villes. Puis à la fin de chaque cycle, les variables des phéromones sont mises à jour.

Principe de l'algorithme :

1. Chaque fourmi calcule la longueur du chemin qu'elle a réalisé.

2. On calcule ensuite la quantité de phéromones a déposée par la fourmi k sur l'arc entre deux villes i et j.

3. On observe quelle fourmi a trouvé le chemin de longueur minimum. Si ce chemin est meilleur que le meilleur chemin jusqu'ici, on le mémorise.

4. Les mémoires des fourmis (liste des villes visitées) sont effacées.

5. A la fin de l'itération, on a la somme des phéromones qui ne se sont pas évaporées et de celles qui viennent d'être déposées :

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

6. Les fourmis recommencent à nouveau, toujours au départ de la ville sur laquelle elles avaient été placées au début de l'algorithme.

Avantages : c'est un algorithme parfait pour les problèmes basés sur des graphes et a une capacité d'adaptation très grande pour différents problèmes.

Inconvénients : un état bloquant peut arriver (les fourmis stagnent, lorsque toutes les fourmis font le même tour, on est dans une situation de stagnation où le programme arrête de chercher des alternatives) et le temps d'exécution peut être parfois long (nombre de cycle très grand pour trouver le chemin le plus court).

Conclusion : bien que le résultat trouvé derrière est bien meilleur que l'algorithme du plus proche voisin, le temps de calcul est énorme, et nous perdons donc en rapidité.

III. Implémentation d'algorithme d'amélioration

Recherche locale (permutations + itérations):

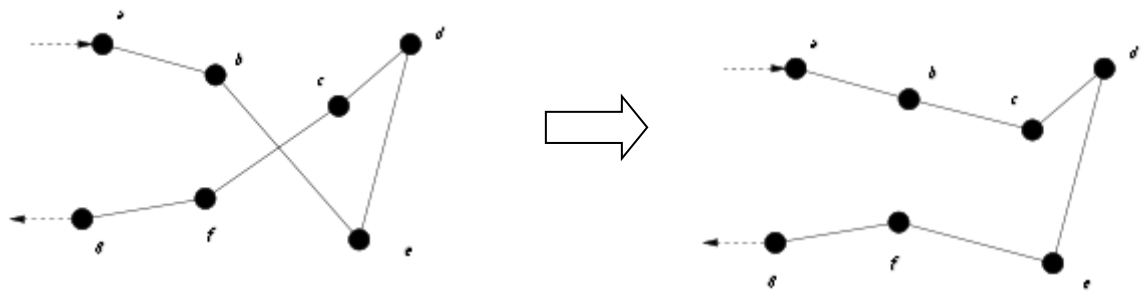
Principe de l'algorithme : on choisit une ville i et on l'échange de position avec toutes les autres villes (on fixe cependant toujours la première et la dernière ville). On garde ensuite le tableau des positions des villes ayant la distance la plus petite, puis on fait des itérations à partir de la solution optimale trouvée.

Création de la fonction swap dans la classe Solution, utilisée dans la classe TSPSolver, solve().

La recherche locale améliore donc la solution trouvée avec l'algorithme, on trouve en effet une meilleure solution mais avec un temps de calcul plus grand. Pour l'instant la contrainte de temps est respectée (cf. comparaison des solutions). La solution n'étant pas optimale, nous avons décidé de nous lancer dans l'algorithme des colonies de fourmis qui nous semblait être l'algorithme le plus adapté pour trouver une solution rapidement et optimale.

Méthode 2-opt :

Cet algorithme améliore la solution en réorganisant deux arêtes à la fois. C'est un algorithme itératif : à chaque étape, on supprime deux arêtes de la solution courante et on reconnecte les deux tours ainsi formés. Traduction algorithmique : Pour tout sommet xj de H, avec j différent de i-1, de i et de i+1 faire : si distance(xi, xi+1) + distance(xj, xj+1) > distance(xi, xj) + distance(xi+1, xj+1) alors remplacer les arêtes (xi, xi+1) et (xj, xj+1) par (xi, xj) et (xi+1, xj+1) dans H



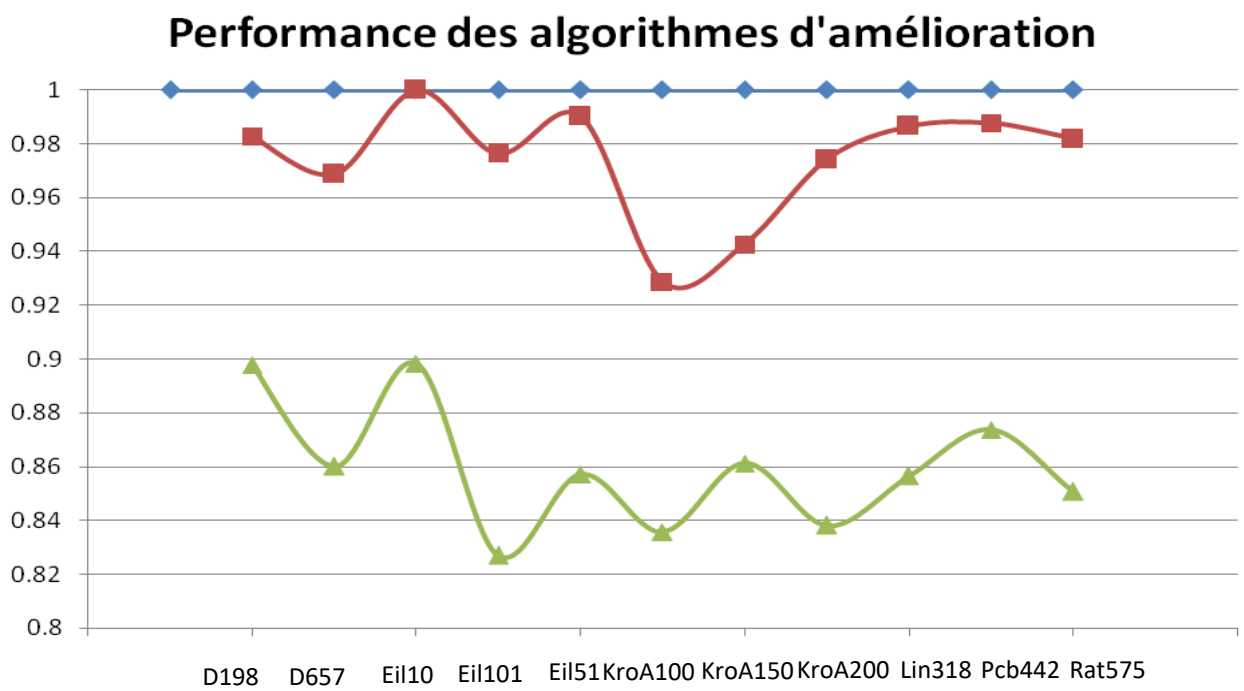
IV. Résultats

1) Comparaison des solutions

Comme nous pouvons voir d'après l'annexe 1, 2 et 3, parmi nos algorithmes mis en œuvre, la meilleure solution reste l'algorithme d'amélioration 2-opt appliqué initialement à la solution de l'algorithme du plus proche voisin qui présente la meilleure solution en terme de distances mais aussi en terme de temps d'exécution (très inférieur au temps d'exécution de la recherche locale par exemple). En temps réel, pour calculer une plus courte distance, il est donc plus intéressant d'appliquer l'algorithme 2-opt.

Cependant, nous avons tenté d'effectuer l'algorithme des colonies de fourmis qui aurait surement présenté une meilleure solution (distance), mais qui aurait demandé un temps d'exécution bien supérieur.

Comparaison des solutions trouvées pour la recherche local, 2-opt, (en pourcentage du PPV)



2) Conclusion

A défaut d'avoir pu réussir à obtenir des résultats satisfaisants de l'algorithme des colonies de fourmis (dont on aurait pu ensuite améliorer la solution avec un algorithme d'amélioration), on réussit avec un algorithme d'amélioration simple (2-opt) en partant du plus-proche-voisin, à avoir des vitesses d'exécution rapides et des résultats avec un taux moyen d'erreur à la solution optimale de 6%.

Nous présentons donc pour la compétition la combinaison de ces deux algorithmes.

V. Annexes

Annexe 1 : Solutions trouvées après l'algorithme du plus proche voisin

Données	Temps	Solution	%Optml	Données	Temps	Solution	%Optml
Brazil58		30774		**KroA100**	5	27807	76.53%
D198	18	18240	86.51%	**KroA150**	8	33633	78.86%
D657	34	61626	79.37%	**KroA200**	8	35859	81.90%
Eil10	1	177		**Lin318**	12	54019	77.80%
Eil101	4	803	78.33%	**Pcb442**	25	61979	81.93%
Eil51	2	511	83.37%	**Rat575**	27	8605	78.71%

Pour ces instances tests, notre taux d'erreur par rapport à la solution optimale est de 20% en moyenne

Annexe 2 : Solutions trouvées après recherche local sur le plus proche voisin

Données	Temps	Solution	%Optml	Données	Temps	Solution	%Optml
Brazil58				**KroA100**	44	25827	82.40%
D198	142	17926	88.03%	**KroA150**	113	31705	83.54%
D657	8148	59705	81.92%	**KroA200**	154	34936	84.06%
Eil10	1	177		**Lin318**	496	53293	78.86%
Eil101	26	784	80.23%	**Pcb442**	948	61220	82.94%
Eil51	5	506	84.19%	**Rat575**	2987	8449	80.16%

Pour ces instances tests, notre taux d'erreur par rapport à la solution optimale est de 17% en moyenne

Annexe 3 : Solutions trouvées après l'algorithme 2-opt sur le plus proche voisin

Données	Temps	Solution	%Optml	Données	Temps	Solution	%Optml
Brazil58				**KroA100**	29	23239	91.58%
D198	82	16376	96.36%	**KroA150**	58	28967	91.57%
D657	1375	53000	92.29%	**KroA200**	146	30053	97.72%
Eil10	1	159		**Lin318**	180	46265	90.84%
Eil101	40	664	94.73%	**Pcb442**	354	54154	93.77%
Eil51	11	438	97.26%	**Rat575**	721	7321	92.51%3

Pour ces instances tests, notre taux d'erreur par rapport à la solution optimale est de 6% en moyenne

Annexe 4 : bibliographie

http://www.i3s.unice.fr/~crescenz/publications/travaux_etude/colonies_fourmis-200605-rapport.pdf

https://en.wikipedia.org/wiki/2-opt

https://campusneo.mines-nantes.fr/campus/pluginfile.php/65475/mod_resource/content/1/Cours%201.pdf

Lien TSP GITHUB : https://github.com/Lightlo/TSP_Mabru_Lhomme_Oustric