Rapport ITD - problème du voyageur de commerce

par le groupe 11 constitué de Thomas Bazaille, Brendan Becaert et Louis Carrié

I Introduction

Nous rappelons brièvement le but du projet. Chaque groupe d'élèves avait pour mission de trouver une heuristique ou métaheuristique permettant de résoudre le problème du voyageur de commerce en moins de 60 secondes.

Le présent rapport décrit 7 des 9 algorithmes sur lesquels nous avons travaillés au cours du projet. Le premier algorithme que nous ne résumons pas dans la deuxième partie est l'algorithme qui consiste à insérer le sommet i en position i dans la tournée. Le second est celui qui, étant donnée une ville A, consiste à trouver la ville B la plus proche et construire ainsi un chemin emprunté par le voyageur de commerce.
La présentation de chaque algorithme suit le déroulé suivant. Premièrement, une description brève du concept de l'algorithme avec les difficultés rencontrées le cas échéant. Deuxièmement, une bibliographie.

Le tableau comparatif des performances des différents algorithmes (temps de calcul et proximité à la solution optimale dans le cas où le temps de calcul est inférieur à 60 secondes) se trouve dans la partie **Annexes**

II Algorithmes

1) LocalSearchSwap et 2-Opt

Ces algorithmes se fondent sur le principe du Local Search. L'idée du Local Search est la suivante. On crée un premier chemin emprunté par le voyageur de commerce. Ce chemin peut-être arbitraire ou aléatoire et est codé sous forme d'un tableau. Ensuite, on remplace le chemin courant (i.e emprunté actuellement par le commis voyageur) par la meilleure solution choisie dans le voisinage de la solution courante. Tant que l'on trouve une meilleure solution s' dans le voisinage de la solution courante, on répète le processus sur le nouveau chemin s'.

Dans le cas de LocalSearchSwap, le voisinage d'une solution s est définie comme l'ensemble des solutions s' obtenues par inversion de couples d'indices de s. La solution initiale est celle renvoyée par l'algorithme qui insère le sommet i en position i dans la tournée.

Dans le cas du 2-Opt, pour obtenir un voisin d'un chemin s codé sous la forme d'un tableau, il faut créer une copie de s dans laquelle on inverse un sous tableau. Un voisin de [0, 2, 1, 4, 5, 3, 6, 0] au sens du 2-Opt est par exemple [0, 2, 5, 4, 1, 3, 6, 0], le sous tableau [1, 4, 5] a été inversé pour donner [5, 4, 1]. La solution initiale est aléatoire.

Bibliographie : 1) Gilles Simonin, *Introduction aux techniques décisionnelles, Cours 2 - Introduction aux problèmes d'ordonnancement*
2) Jacques Teghem, *Recherche Opérationnelle, Tome 1 : Méthodes d'optimisation, Chapitre 15. Les heuristiques, page 380*, Ellipses
3) Gerhard Reinelt, *The Traveling Salesman, Computational Solutions for TSP. Applications, volume 840 of Lecture Notes in Computer Science (LNCS). Chapter 7. Improving Solutions* Springer-Verlag, Berlin Heidelberg, Heidelberg, 1994. Reachable through the link The Traveling Salesman Gerhard Reinelt
4) Christian Nilsson, *Heuristics for the Traveling Salesman Problem*, article reachable through the link Heuristics for the Traveling Salesman Problem

2) Algorithme Génétique

Les algorithmes génétiques sont inspirés de l'évolution darwinienne, de la reproduction sexuée et du hasard qui en découle.
Dans la suite de cette partie, on appelle individu un chemin emprunté par le voyageur de commerce. Un tel individu est codé sous la forme d'un tableau.
Tout d'abord, plusieurs individus sont générés aléatoirement pour former une population initiale de chemins. Ensuite, on sélectionne des paires de chemins parmi tous les chemins de la population initiale pour former la population sélectionnée. La sélection choisie est une sélection par tournoi, i.e deux chemins sont tirés au sort et celui qui sera sélectionné sera celui ayant le coût le moins important (dans notre cas, le coût est égal à la distance parcourue). Puis, pour chaque paire (parent1, parent 2) d'individus de la population sélectionnée, on génère une paire d'individus enfants (enfant1, enfant2) à l'aide d'un croisement. enfant1 est généré en recopiant l'ordre des villes apparaissant dans parent1 jusqu'à un certain indice c choisi aléatoirement. Pour tous les indices après c, on insère la première ville rencontrée de parent2 n'apparaissant pas déjà dans enfant1. On génère enfant2 en échangeant les rôles des deux parents. Après, la population enfant obtenue, nous effectuons une mutation sur chaque enfant avec une probabilité de 20%. La mutation retenue est un 2-Opt. Pour obtenir la mutation d'un enfant, il faut créer une copie dans laquelle on inverse un sous tableau de enfant. Une mutation de [0, 2, 1, 4, 5, 3, 6, 0] au sens du 2-Opt est par exemple [0, 2, 5, 4, 1, 3, 6, 0], le sous tableau [1, 4, 5] a été inversé pour donner [5, 4, 1]. Enfin, on modifie la population initiale en une population hybride constituée de la population enfant mutée et de certains chemins de la population initiale. Les m enfants mutés remplacent les m parents ayant le coût le plus élevé. On répète le processus précédent un certain nombre de fois et on retourne l'individu avec le plus faible coût.

Le choix de la méthode de sélection, du croisement et de la mutation se fondent sur les documents 1), 2) et 4) de la bibliographie.
Les choix de la taille de la population initiale et de la taille de la population sélectionnée ont été effectués par une étude empirique.

Le principe d'un algorithme génétique est résumé par la figure 2 de la partie **Annexes**

Nous avons tenté d'améliorer les performances en modifiant la phase de croisement. Nous nous sommes appuyés sur le document 2) de la bibliographie. La méthode de croisement est inspirée de la théorie des chocs élastiques entre deux particules et lorsqu'on se place à une dimension (équations rappelées en figure 3 de la partie **Annexes**). Malheureusement, cette tentative d'amélioration s'est révélée infructueuse. Nous avons néanmoins laissé le bout de code.

Bibliographie : 1) Jacques Teghem, *Recherche Opérationnelle, Tome 1 : Méthodes d'optimisation, Chapitre 17. Les métaheuristiques évolutionnaires, pages 400-412*, Ellipses
2) Ahmad B. A. Hassanat and Esra'a Alkafaween, *On Enhancing Genetic Algorithms Using New Crossovers*, published on arXiv, article reachable through the link On Enhancing Genetic Algorithms
3) Gerhard Reinelt, *The Traveling Salesman, Computational Solutions for TSP Applications, volume 840 of Lecture Notes in Computer Science (LNCS). Chapter 9. Further Heuristic Approaches* Springer-Verlag, Berlin Heidelberg, Heidelberg, 1994. Reachable through the link The Traveling Salesman Gerhard Reinelt
4) Christian Nilsson, *Heuristics for the Traveling Salesman Problem*, article reachable through the link Heuristics for the Traveling Salesman Problem
5) Yves Coueque, Julien Ohler et Sabrina Tollari, *Algorithmes génétiques pour résoudre le problème du commis voyageur*, Travail de recherche mis sur Internet (indisponible au moment de la rédaction du rapport) au lien Algorithmes génétiques pour résoudre le problème du commis voyageur

3) k-Opt

Au vu des bons résultats de l'algorithme 2-Opt et de la littérature existante pour le TSP, nous avons voulu coder le k-Opt.
On commence par coder les algorithmes 2-opt, 3-opt et 4-opt. Cela consiste à retirer respectivement 2, 3 ou 4 arêtes du chemin considéré et à les réinsérer de manière à créer un nouveau chemin dont le poids total est diminué. Dans notre implémentation, on a considéré uniquement les cas où la réinsertion des 2, 3 ou 4 arêtes est complètement nouvelle, c'est-à-dire les cas où les 2, 3 ou 4 arêtes insérées sont chacune nouvelles.
On améliore une solution existante en lui appliquant en cascade les algorithmes 2-opt, 3-opt et 4-opt. Tant qu'il y a des améliorations, on fait tourner 2-opt puis 3-opt. Quand il n'y a plus d'amélioration on fait tourner 4-opt et on recommence l'étape précédente si 4-opt a amélioré la solution.

Bibliographie : 1) Jacques Teghem, *Recherche Opérationnelle, Tome 1 : Méthodes d'optimisation, Chapitre 15. Les heuristiques, pages 380-381*, Ellipses
2) Christian Nilsson, *Heuristics for the Traveling Salesman Problem*, article reachable through the link Heuristics for the Traveling Salesman Problem
3) Andrius Blazinskas and Alfonsas Misevicius, *COMBINING 2-OPT, 3-OPT AND 4-OPT WITH K-SWAP-KICK PERTURBATIONS FOR THE TRAVELING SALESMAN PROBLEM. Article reachable through the link* k-Opt algorithm

4) Algorithme de Christofides

L'algorithme est le suivant :
Etant donné un graphe G = (V, E) dont les poids respectent l'inégalité triangulaire.

1. Calculer un arbre couvrant de poids minimal T de G
2. Soit I l'ensemble des sommets de degré impair dans T, calculer un couplage parfait M de poids minimum dans le sous-graphe induit par les sommets de I
3. On définit un multigraphe H à partir des arêtes issues de M et T
4. Trouver un cycle eulérien dans H (H est eulérien car il est connexe et tous les sommets sont de degrés pairs)
5. Transformer le cycle eulérien en un cycle hamiltonien en supprimant les éventuels passages en double sur certains sommets

Nous avons effectué le 1. à l'aide de l'algorithme de Prim. Ensuite le 2. se fait grâce à l'algorithme hongrois. Nous avons ensuite codé l'algorithme de Fleury pour le 4.
Nous avons eu beaucoup de difficultés avec le 2. On est censé trouver un couplage parfait M de poids minimum dans le sous-graphe induit par les sommets de I, que l'on appellera S. Le problème est que S est un graphe complet alors que l'algorithme hongrois donne un couplage parfait pour un graphe biparti. Nous avons donc décidé de séparer S de manière à obtenir un graphe biparti S', même si alors on perd de l'information et donc un couplage parfait sur S' n'est pas forcément un couplage parfait sur S. Nous avons beaucoup cherché et impossible de trouver un algorithme qui trouve un couplage parfait dans un graphe quelconque. Mis à part cela, l'implémentation de l'algorithme hongrois fonctionne bien pour certaines instances. Nous avons effectué l'algorithme à la main sur une instance dont notre implémentation ne trouvait pas de solution, et avons bien vérifié étape par étape ce que renvoyait le code. Il en résulte sur cet exemple que notre implémentation est conforme avec l'algorithme décrit par Kuhn-Munkres (les créateurs de l'algorithme hongrois). Nous avons alors pensé qu'il n'y avait peut-être pas de solution à notre exemple. Nous avons donc décidé de coder une fonction récursive qui teste toutes les possibilités et nous renvoie la solution si elle existe. Résultat, il existait bien une solution optimale. C'est pourquoi nous nous posons des questions quant à la terminaison de l'algorithme hongrois. Nous n'avons malheureusement pas réussi à mettre la main sur la preuve de la terminaison de l'algorithme et tout cela reste pour nous encore un grand mystère que nous aimerions bien élucider un jour.

Bibliographie : 1) Christian Nilsson, *Heuristics for the Traveling Salesman Problem*, article reachable through the link Heuristics for the Traveling Salesman Problem
2) Philippe Roux, *Chemins Eulériens et Algorithme de Fleury*, page web : Philippe Roux, Chemins Eulériens et Algorithme de Fleury
3) Wikipedia, *Christofides algorithm*, link : Christofides algorithm

5) Algorithme de la colonie de fourmis

L'algorithme des colonies de fourmis repose sur un principe similaire aux fourmilières dans lesquelles les fourmis cherchent à trouver de la nourriture et libèrent de la phéromone sur les chemins empruntés. La quantité de phéromone d'un chemin est d'autant plus importante que le chemin a du potentiel d'être court. Dans l'algorithme, les quantités de phéromone de chaque chemin sont initialement égales. Sur leur chemin de retour, les fourmis déposent de la phéromone sur les différents chemins empruntés. La nouvelle valeur de la quantité de phéromone est inversement proportionnelle à la longueur du chemin.

Le processus est réitéré, et les fourmis repartent à nouveau de la source. Une fourmi donnée va choisir entre les chemins selon une probabilité proportionnelle aux quantités de phéromone des chemins. Ainsi les chemins avec le plus de potentiel verront la probabilité d'être choisis augmenter grâce à une mise à jour des quantités de phéromone à chaque tour de boucle.

La principale difficulté que nous avons rencontrée lors de la modélisation de cet algorithme est la multitude des paramètres à prendre en compte : en effet nous avons choisi de modéliser chaque fourmi par une matrice, chaque élément de cette matrice étant un triplet modélisant la quantité de phéromone présente sur l'arrête entre les villes i et j, la distance séparant nos deux villes et enfin un booléen valant true si l'arrête avait été empruntée, false sinon.
Nous nous retrouvions ainsi avec une liste de fourmis, chacune modélisée par une matrice de triplet, une modélisation sans doute très lourde pour le problème auquel nous étions confrontés, ceci engendrant des parcours et des appels à des coefficients compliqués.
Nous n'obtenons pas de résultats concluants avec ce modèle, en effet notre programme retourne toujours le même modèle de solution, dans l'ordre croissant du numéro des villes.

Bibliographie : 1) Jacques Teghem, *Recherche Opérationnelle, Tome 1 : Méthodes d'optimisation, Chapitre 17. Les métaheuristiques évolutionnaires, pages 412-416*, Ellipses
2) Walid Tfaili, *Conception d'un algorithme de colonie de fourmis*, lien : Conception d'un algorithme de colonie de fourmis

6) Arbre couvrant

Dans cet algorithme, on construit tout d'abord un arbre couvrant de poids minimal. Un arbre couvrant de poids minimal est un arbre qui connecte tous les sommets d'un graphe ensemble et dont la somme des poids des arêtes est minimale. On effectue ensuite simplement un parcours en profondeur dans cet arbre afin d'obtenir un chemin hamiltonien.

Bibliographie : 1) Robert Sedgewick and Kevin Wayne, Minimum Spanning Tree, Article reachable through the link Minimum Spanning Tree

III Gestion du GitHub

Ce projet a été l'occasion pour chacun de nous de travailler pour la première fois sur GitHub. Nous avons mis un peu de temps avant de s'organiser pour ne pas avoir de problème de merging (notamment pour l'utilisation de TSPSolver). Cependant, une fois GitHub pris en main, nous avons apprécié le fait de pouvoir avancer en parallèle sur nos algorithmes respectifs. C'était donc une expérience enrichissante.

Annexes

Figure 1 - tableau comparatif des différents algorithmes

lecture : pour chaque algorithme, la première ligne correspond à la valeur retournée par l'algorithme, la deuxième ligne au temps d'exécution et la troisième ligne à l'écart relatif par rapport à la valeur optimale

Tableau 1

	d198	d657	eil10	eil51	eil101	kroA100	kroA150	kroA200	lin318	pcb442	rat575
Solution optimale	15 780	48 912	non communiqué	426	629	21 282	26 524	29 368	42 029	50 778	6 773
ArbreCouvrant	19 129	65 896	214	581	864	27 210	35 119	40 028	59 158	69 623	9 497
	55ms	197ms	2ms	10ms	28ms	23ms	37ms	106ms	96ms	124ms	316ms
	21,2 %	34,7 %		36,4 %	37,4 %	27,9 %	32,4 %	36,3 %	40,8 %	37,1 %	40,2 %
Chistofides	24 006		203	585	778	29 757	52 997	43 558	61 782		
	221ms		7ms	22ms	97ms	66ms	229ms	196ms	1s 459ms		
	52,1 %			37,3 %	23,7 %	39,8 %	99,8 %	48,3 %	47 %		
PlusProcheVoisin	17 620	60 174	177	482	746	24 698	31 479	34 543	49 201	58 950	7 993
	163ms	2s 278ms	3ms	23ms	51ms	65ms	116ms	179ms	387ms	938ms	1s 653ms
	11,7 %	23 %		13,1 %	18,6 %	16 %	18,7 %	17,6 %	17,1 %	16,1 %	18 %
AlgorithmeGenetique (moyenne sur 4 tests)	19 182		167	477	749	25 263	30 965	36 157	51 678		
	24s 806ms		1s 200ms	5s 200ms	10s	9s 774ms	14s 886ms	27s	53s 614ms		
	21,6 %			12 %	19,1 %	18,7 %	16,7 %	23,1 %	23 %		
LocalSearchSwap	20 789		208	620	1018	42 326	53 514	71 036	77 643	97 750	
	1s 160ms		3ms	41ms	307ms	368ms	2s 17ms	7s 742ms	14s 996ms	56s 591ms	
	31,7 %			45,5 %	61,8 %	98,9 %	101,8 %	141,9 %	84,7 %	92,5 %	
2-Opt	18 769	57 818	165	491	725	26 290	30 652	34 995	50 432	59 155	8 404
	2s 790ms	36s 238ms	39ms	218ms	674ms	698ms	1s 479ms	3s 437ms	11s 210ms	19s 972ms	26s 141ms
	18,9 %	18,2 %		15,3 %	15,9 %	23,5 %	15,6 %	19,1 %	20 %	16,5 %	24 %
k-Opt	16 166	51 043	159	438	637	21 343	27 650	30 537	44 366	52 573	7 212
	2s 80ms	16s 996ms	17ms	383ms	1s 667ms	7s 70ms	16s 212ms	5s 601ms	2s 48ms	37s 439ms	5s 162ms
	2,45 %	4,36 %		2,82 %	1,27 %	0,29 %	4,25 %	3,98 %	5,57 %	3,53 %	6,48 %
ColonieDeFourmis											

l'algorithme fonctionne mais ne renvoie pas de solution avant 60 secondes

l'algorithme ne fonctionne pas ou ne fonctionne pas comme il devrait

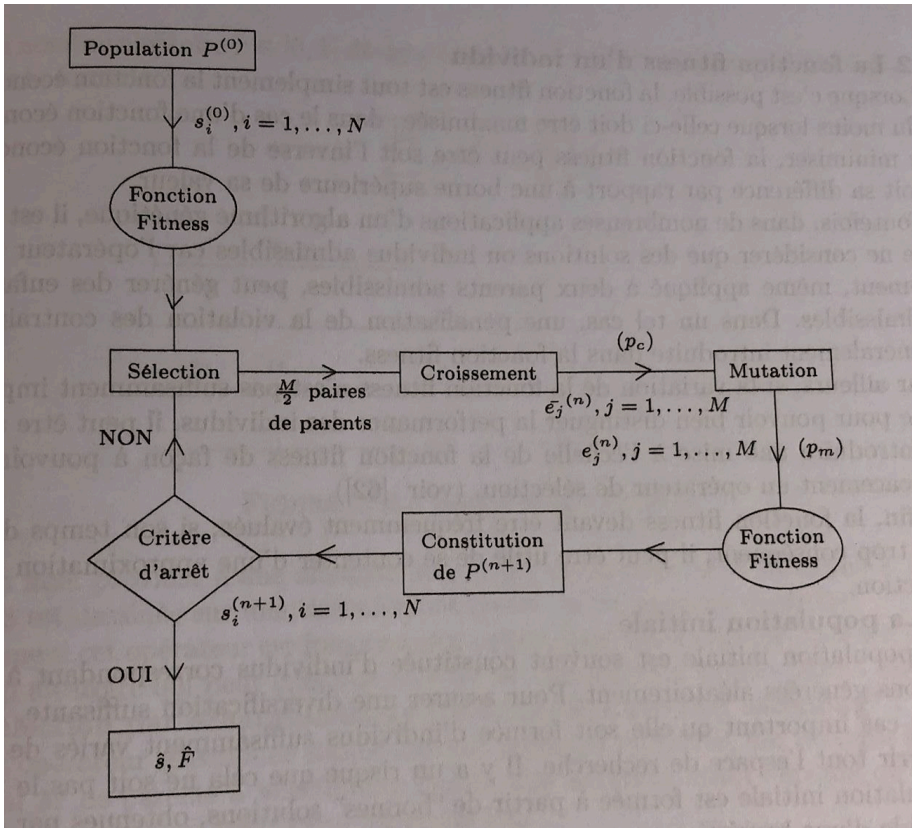

Figure 2 - organigramme d'un algorithme génétique
source : Jacques Teghem, *Recherche Opérationnelle, Tome 1 : Méthodes d'optimisation, page 405*

$$v'_1 = \frac{m_1 - m_2}{m_1 + m_2} v_1 + \frac{2m_2}{m_1 + m_2} v_2 \qquad (4)$$

$$v'_2 = \frac{2m_1}{m_1 + m_2} v_1 - \frac{m_1 - m_2}{m_1 + m_2} v_2 \qquad (5)$$

Figure 3 - équations des chocs élastiques entre deux particules
source : Ahmad B. A. Hassanat and Esra'a Alkafaween, *On Enhancing Genetic Algorithms Using New Crossovers*, published on arXiv, article reachable through the link
On Enhancing Genetic Algorithms