Rapport Méta-heuristique

Le problème du voyageur de commerce résolu avec des algorithmes génétiques

Réalisé par :

Carlos SANIN & Yann BISCH

Année 2014/2015

Table des matières

Introduction

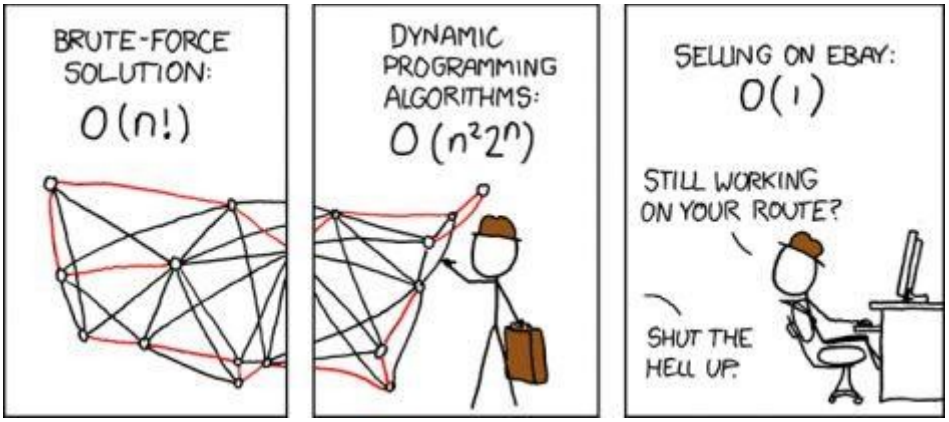
Comme nous avons déjà vu on peut fréquemment se trouver avec des problèmes très compliqués qui ont beaucoup de variables et leur analyse peut devenir un vrai casse-tête, comme les problèmes NP-Complets et NP-Difficiles.

Pour présenter le contexte, rien de mieux que des exemples ! Nous allons utiliser le problème très connu du voyageur de commerce, où un marchand veut visiter un ensemble de villes en finissant son voyage dans la même ville que celle dont il est parti, mais sous la condition de ne visiter qu'une et une seule fois chaque ville (pas de retour possible à part pour la ville de départ qui peut, selon la description du problème ou le choix d'implémentation, être considérée comme visité une seule fois du départ (ou de l'arrivée), ou bien être visité une fois au départ et une fois à l'arrivée (départ = arrivée, cycle Hamiltonien)). Dans notre cas, le chemin doit débuter par la ville de départ et finir par cette même ville

Ce problème, qui peut sembler facile au premier abord, est l'un des problèmes les plus difficiles connu si le nombre de villes à visiter est très grand et dépend aussi des connexions entre les villes.

Plus formellement on peut définir le problème de la façon suivante:

Soit un graphe complet G = (V,A,w) avec V un ensemble de sommets, A un ensemble d'arêtes et w une fonction de coût sur les arcs. Le problème est de trouver le plus court cycle Hamiltonien dans le graphe.



« Si un voyageur part du point A et que les distances entre toutes les villes sont connues, quel est le plus court chemin pour visiter tous les points et revenir au point A ? »

Il existe plusieurs techniques pour résoudre ce problème comme notamment des algorithmes d'approximation, des Heuristiques générales, des Heuristiques gloutonnes, la programmation dynamique, des algorithmes randomisés, la programmation linéaire, des algorithmes évolutifs, etc.

Dans le cadre de notre projet, nous essayerons de résoudre ce problème en utilisant les Algorithmes Évolutifs. Pour cela nous définirons ces quelques points:

- Contexte
- Les techniques utilisées pour résoudre le problème du voyageur de commerce
- Algorithmes et Pré-tests
- Les Tests

Contexte

Qu'est-ce que c'est un algorithme génétique, un peu d'histoire

Pour commencer, les algorithmes génétiques sont des méthodes d'optimisation basées sur une simulation partielle des mécanismes de l'évolution naturelle. Ils sont basés sur la théorie de l'évolution de Charles Darwin(1959) qui avait publié son livre intitulé « *L'origine des espèces ou moyen de la sélection naturelle ou la lutte pour l'existence dans la nature* ». Dans son livre Darwin rejette l'existence de systèmes naturels figés, déjà adaptés pour toujours et pour toutes les conditions extérieures. Selon lui les êtres vivants se sont graduellement adaptés à leur milieu naturel au travers de processus de reproductions[1].

Darwin avait proposé une théorie basé en quatre lois

- Croissance et reproduction
- Hérédité (qu'implique quasiment la loi de reproduction)
- Viabilité (résultat des conditions d'existence)
- Multiplication des espèces (qui amène la lutte pour l'existence et qui a pour conséquence la sélection naturelle).

Les Algorithmes génétiques ont étés créés dans les années 60 pour Jhon Holland(1975) comme un modèle pour l'étude du phénomène d'adaptation naturelle et pour le développement de mécanismes qui permettrons d'utiliser ces phénomènes en informatique. Il expliqua comment ajouter de l'intelligence dans un programme informatique avec les croisements (échangeant le matériel génétique) et les mutations (source de la diversité génétique). Ce modèle servira de base aux recherches ultérieures et sera plus particulièrement repris par Goldberg qui publiera en 1989 un ouvrage de vulgarisation des algorithmes génétiques, et ajoutera à la théorie des algorithmes génétiques les idées suivantes :

- Un individu est lié à un environnement par son code d'ADN.
- Une solution est liée à un problème par son indice de qualité.[2]

---

1 http://sis.univ-tln.fr/~tollari/TER/AlgoGen1/node5.html
2 http://souqueta.free.fr/Project/files/TE_AG.pdf

Les Algorithmes génétiques ont trouvé une grande popularité grâce à la publication du livre de Goldberg(1989) qui donnait de bonnes bases pour des applications sur des problèmes pratiques comme pour l'intelligence Artificielle, des problèmes d'optimisation, etc.

Les techniques pour mettre en route un algorithme génétique.

Pour mettre en route un algorithme génétique il faut avoir une population, la faire évoluer et observer le résultat.

Nous utiliserons la théorie de Darwin pour simuler ce comportement dans nos algorithmes.

Tout d'abord nous allons aborder les parties qui composent un algorithme génétique. Dans la nature le processus d'évolution biologique se fait de façon naturelle mais pour appliquer un algorithme génétique à la résolution de problèmes, il faudra suivre une séquence de pas.

Une prémisse est de prendre une population de taille suffisamment grande afin d'assurer la diversité des solutions. Il est souhaitable que la population soit générée de façon aléatoire pour obtenir cette diversité. Dans le cas où la population ne serait pas générée de façon aléatoire il faut garantir une certaine diversité dans la population générée.

Les pas basiques d'un algorithme génétique sont:

- Évaluer la « fitness » de chacun des chromosomes générées.
- Laisser les chromosomes les plus adaptés avoir plus de chance ou probabilité de se reproduire.
- Avec une certaine probabilité de mutation, muter un gène du nouvel individu généré.
- Organiser la nouvelle population.

Ces pas devrons se répéter jusqu'à ce qu'une condition d'arrêt soit vérifiée ou bien établir un nombre maximum d'itérations avant de terminer l'exécution de l'algorithme génétique ou encore, arrêter l'exécution lorsqu'il n'y a plus de changement dans la population (convergence de l'algorithme).

Avantages

Le premier avantage, et le plus important, est que les algorithmes génétiques sont intrinsèquement parallèles. La plupart des autres algorithmes sont en série et peuvent seulement explorer l'espace de solutions vers une solution dans un même temps, et si la solution trouvée est sous-optimale, il faut abandonner et recommencer. Néanmoins, comme les algorithmes génétiques ont une descendance multiple, ils peuvent exploiter un espace de solutions dans différentes directions en même temps. Si un chemin n'est pas conforme, les algorithmes génétiques peuvent l'éliminer sans problème et continuer dans les directions plus séduisantes, en donnant ainsi une plus grande probabilité à chaque exécution de trouver une solution.

Dû au parallélisme qui permet d'évaluer plusieurs schémas en même temps, les algorithmes génétiques fonctionnent très bien en résolvant des problèmes où l'espace de solutions potentiel est vraiment grand, c'est-à-dire assez grand pour faire une recherche exhaustive dans un temps raisonnable. La plupart des problèmes qui sont dans cette catégorie sont connus comme « des problèmes non linéaires ». Dans un problème linéaire, l'aptitude de chaque composant est indépendante, ce qui signifie qu'une amélioration dans n'importe qu'elle sous problème donnera une amélioration dans le système complet. Il n'est pas nécessaire de préciser que dans la réalité, il est difficile de trouver des problèmes de ce type (linéaires). Les problèmes non linéaires sont la norme, où changer un composant peut avoir des effets sur tout le système, et où de multiples changements qui individuellement pourraient être mauvais, en combinaison, seraient amener à une amélioration dans l'aptitude.

Exemple

Un algorithme génétique développé par des ingénieurs de « General Electric » et de « Rensselaer Polytechnic » permit la conception de la turbine d'un moteur à réaction de haute prestation qui était 3 fois meilleur que la configuration conçu par des humains, et 50% meilleur que la conception réalisée par un système expert qui avait parcouru avec succès un espace de solutions qui contenait plus de 10 387 possibilités. Les Méthodes conventionnels pour concevoir des turbines sont une partie fondamentale des projets d'ingénierie qui peuvent avoir une durée de 5 ans et avoir un coût de plus de 2000 millions de dollars. L'algorithme génétique découvrit cette solution en deux jours, dans une station de travail en ingénierie (Holland 1992[36], p. 72)

Inconvénients

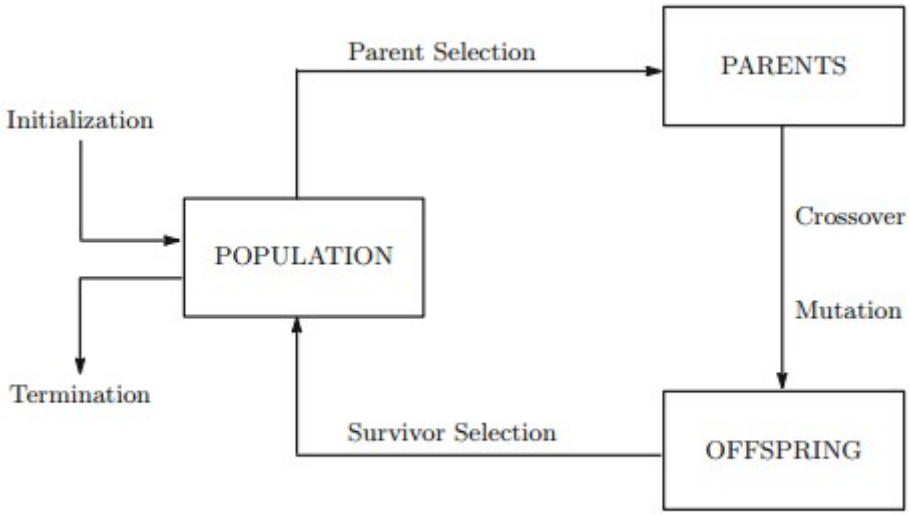
Même si les algorithmes génétiques ont montré leur efficacité et leur puissance comme stratégie pour la résolution de problèmes, ils ne sont pas la solution à tous les problèmes. Les algorithmes génétiques ont certaines limites.

La première et la plus importante considération quand nous allons créer un algorithme génétique est de définir une représentation du problème. Le langage pour spécifier les solutions candidates doit être robuste; c'est-à-dire, qu'il doit être capable de supporter des changements aléatoires, qu'il ne produise pas d'erreurs fatales ni de résultats sans aucun sens.

Les techniques utilisées pour résoudre le problème du voyageur de commerce.

Généralités

Nous allons commencer en nous rappelant avec un petit schéma comment fonctionne un algorithme génétique :



Basic scheme of an evolutionary algorithm

- **Initialisation:** Définir une fonction de fitness et générer une population de manière aléatoire.
- **Crossover et Mutation:** Le but du « crossover » (croisement) est de produire un ou plusieurs fils qui auront de grandes probabilités d'avoir de meilleures propriétés que leurs parents. Les mutations sont faites avec une certaine probabilité de manière périodique pour donner de la variété (et pouvoir sortir de maximaux/minimaux locaux).
- **Sélection des Meilleures**: Dans cette partie la fonction de fitness a un rôle très important car c'est elle qui détermine qui peut passer à la génération suivante.

Notre sujet :

Dans le cas du problème du voyageur de commerce, la fonction de fitness correspond à la distance d'un chemin passant par chaque ville une et une seule fois. Un individu est donc représenté par une liste de villes, dont l'ordre dans la liste (tableau) correspond à l'ordre des villes dans le chemin que va parcourir le voyageur.

Dans notre étude des algorithmes génétiques pour la réalisation de ce problème, nous avons essayé diverses méthodes de croisements, de mutations, et de sélection des survivants, dont voici les listes :

Croisements (Crossover)

- **« Single-point Crossover »** : section en un point du chemin et recopie d'une partie dans un fils et de l'autre partie dans un autre fils.
- **« Uniform Crossover »** : Probabilité uniforme d'échanger chaque élément d'un chemin vers un autre (un enfant hérite avec une probabilité p fixée soit de l'élément du premier parent, soit du deuxième parent)
- **« Order Crossover »** : section d'un chemin père en deux points, recopie de la partie du chemin entre les deux points vers le fils, puis restructuration de l'ordre des autres éléments (pour plus de détails, voir la partie « « Order Crossover » : Le meilleur des meilleurs »)

Mutations

- **« Swap Mutation »** : intervertir deux éléments d'un même chemin fils
- **« Flip Mutation »** : intervertir 2 éléments consécutifs avec 2 autres éléments consécutifs d'un même chemin fils avec l'ordre inversé
- **« Inversion Mutation »** : choisir 2 indices dans le chemin fils, intervertir le premier avec le dernier, le second avec l'avant dernier etc…

Algorithmes et Pré-Tests

Tout d'abord, il faut savoir que nous avons essayé de mettre en place une présélection utilisant la méthode de l'eugénisme (plus connue sous le nom d'élitisme) qui consiste à ne garder que les meilleurs et à les faire se reproduire entre eux (les moins bon n'ont aucune chance). Mais les résultats se trouvant désastreux avec nos implémentations, nous ne l'avons donc pas utilisé ici. Il est possible que cette méthode fonctionne pour d'autres méthodes de croisement/mutation/sélection des survivants, ou des configurations différentes des méthodes présentées ici.

Ensuite, il faut noter que nous n'avons utilisé que la monogamie dans nos méthodes. C'est à dire que ce sont seulement deux parents qui se reproduisent entre eux et donnent naissance à uniquement deux fils.

Enfin, dans le but d'éviter que ce soit toujours les individus de la même famille qui se reproduisent entre eux, nous mélangeons l'ensemble de la population. Les couples de parents varient donc à chaque génération (un individu peut avoir plusieurs partenaires au cours de sa vie, mais un seul par génération du fait de la monogamie).

En ce qui concerne les mutations, nous avons testé différentes manières notamment, avec une probabilité de muter ou encore en ne faisant muter qu'une partie de la population et il s'avère qu'avoir une mutation à chaque génération pour chaque enfant s'avère le plus efficace.

Dans ce rapport, nous ne montrerons qu'une partie des résultats avec un nombre d'exécutions de 10 sur un ensemble de 127 villes et de 657 villes.

Dans tout ce qui suit, la taille de la population utilisée est 48.

« SinglePoint Search »

Nous avons choisi de réaliser deux versions du « SinglePoint Search ». La première version remplace à chaque génération le parent par son fils si celui-ci est meilleur, sinon le fils n'est pas ajouté à la population. Cela présente un avantage qui est de garder seulement les individus de la génération suivante qui sont meilleurs que leurs parents. Le désavantage réside dans le fait qu'il se pourrait qu'on écrase des parents meilleurs que d'autres parents de la même génération (cela laisse plus de chance aux moins bons parents, mais peu ralentir la recherche du meilleur chemin).

Voici les résultats des différents couples croisement/mutation effectués avec un temps de 5 secondes par exécutions :

	Swap Mutation	Flip Mutation	Inversion Mutation
SinglePoint Search	74510.95280346369	59976.231494817985	48230.33933117198

« SinglePoint Search 2 »

La seconde version diffère dans la sélection des survivants ainsi que dans l'application de la mutation. En effet, cette fois-ci nous ne remplaçons pas les parents par leurs enfants s'ils sont meilleurs, mais nous ne gardons que les meilleurs individus parmi les parents et les enfants. De plus, nous ne mutons qu'un pourcentage des enfants (après croisement de deux parents bien sûr), et ici nous avons choisi 10% (en prenant exemple sur les explications de notre enseignant).

Voici les résultats des différents couples croisement/mutation effectués avec un temps de 5 secondes par exécutions :

	Swap Mutation	Flip Mutation	Inversion Mutation
SinglePoint Search 2	132748.2673019877	92392.26908610907	48230.33933117199

« Uniform Corssover »

L'avant dernier algorithme que nous avons mis en place est le « Uniform Search ». Son principe est similaire à celui de « SinglePoint Search ».

Voici les résultats des différents couples croisement/mutation effectués avec un temps de 5 secondes par exécutions :

	Swap Mutation	Flip Mutation	Inversion Mutation
Uniform Search	456154.01885801693	459467.8040773695	460517.7162260753

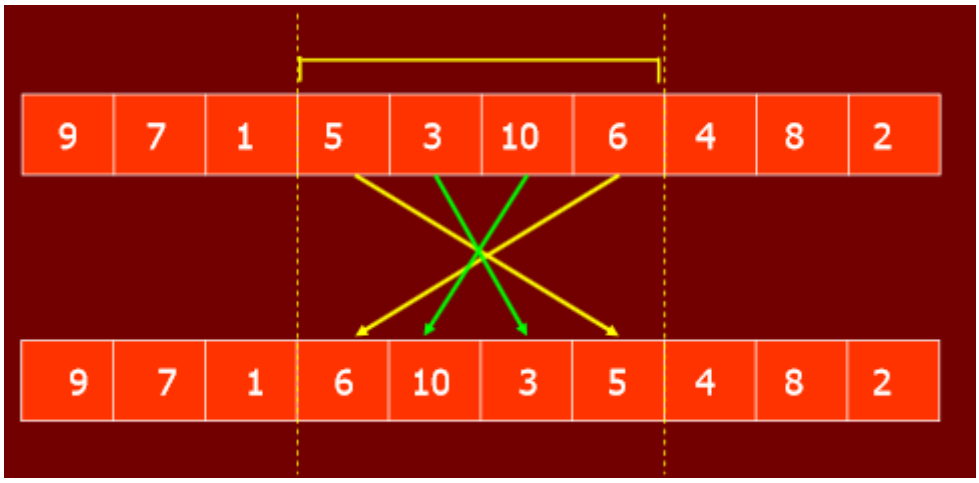
« Order Search » : Le meilleur des meilleurs

Pour résoudre ce problème, dans la littérature les spécialistes disent qu'il est recommandé d'utiliser comme méthode « Order Search » qui marche de la façon suivante:

PAS	ACTION	EXEMPLE
0	Sélectionner P1 et P2 de la population	P1 = 1-2-**5-4-3**-7-6 P2 = 5-4-**2-6-3**-1-7
1	Sélectionner aléatoirement deux points c_1 et c_2 avec $c_1 < c_1$	random = 0.44; alors c_1 = int(7x0.44)+1 = 3 random = 0.71; alors c_2 = int(7x0.71)+1 = 5
2	Changer les positions (c_1, c_1+1,...c_2) dans P1 et P2 pour former partiellement C1 et C2.	C1 = ?-?-**2-6-3**-?-? C2 = ?-?-**5-4-3**-?-?
3	Créer une liste L1(L2) en réaccommodant les éléments de P1(P2) dans le sens des aiguillés de l'horloge $c_2 + 1$, $c_2 + 2$,... 1, 2,...c_2	L1 = (7, 6, 1, 2, 5, 4, 3) L2 = (1, 7, 5, 4, 2, 6, 3)
4	De L1(L2), créez L1' (L2') en éliminant les noedes déjà attribues à C1(C2) dans le pas 2 et au même temps nous conservons l'ordre dans L1 et L2	L1' = L1 - (2, 6, 3) = (7, 1, **5, 4**) L2' = L2 - (5, 4, 3) = (1, 7, **2, 6**)
5	Attribuer les éléments de L1'(L2') aux éléments qui manquent dans C1(C2) avec l'ordre suivant $c_2 + 1$, $c_2 + 2$,... 1, 2,...$c_1 - 1$	C1 = **5-4**-2-6-3-7-1 C2 = **2-6**-5-4-3-1-7

Une fois le croisement réalisé, pour la mutation nous avons choisi Inversion Mutation qui est une variation de Swap Mutation et qui marche de la façon suivant:

On choisit 2 points de manière aléatoire et on change l'ordre de parcours entre les points choisis, comme dans la figure ci-dessous.



Pour cette configuration, nous avons choisi de remplacer les 2 moins bons éléments de la population (ce peut-être des parents ou bien des enfants qui étaient meilleurs que les moins bons parents et les ont donc remplacé mais sont restés les moins bon dans la population ainsi renouvelé).

Voici les résultats des différents couples croisement/mutation effectués avec un temps de 5 secondes par exécutions :

	Swap Mutation	Flip Mutation	Inversion Mutation
Order Search	86650.5620228743	64307.11910822934	48230.33933117196

Nous pouvons noter après ces phases de pré-tests que la mutation de type inversion semble la plus efficace. C'est pour cela que nous avons choisi de n'utiliser que celle-ci pour l'ensemble des tests effectués dont vous pouvez voir les résultats ci-après.

Les Tests

Ensemble de 127 villes

Les résultats avec « Inversion Mutation » sur l'ensemble de 127 villes :

Method/Temps (sec)	*Random*	*Uniform*	*SinglePoint*	*SinglePoint2*	*Order*
5	443128.5462631358	458319.4626647226	48230.339331171197	48230.339331171985	48230.339331171985
10	/	/	/	/	/
20	/	/	/	/	/
30	/	/	/	/	/
40	433974.6247166071	444965.89252337086	48230.339331171196	48230.339331172	48230.339331171985
50	430956.1272890781	445917.37540813535	48230.339331171956	48230.339331172	48230.339331177
60	428921.9225009605	440864.67319248884	48230.339331171197	48230.339331171985	48230.339331171985
Ordre meilleur résultat	*3*	*4*	*1*	*1*	*1*

On observe ici que les trois derniers algorithmes trouvent la meilleure solution en 5 secondes, ce qui ne permet pas de bien différencier leurs performances. Nous avons donc réalisé les tests sur un ensemble de villes en contenant 657. Nous pouvons néanmoins observer que « Uniform » est pire que « Random » sur cet ensemble.

Vous remarquerez qu'il y a une légère différence au niveau des décimales, mais elles sont négligeables et probablement dû à des arrondissements.

Ensemble de 657 villes

Voici les résultats avec « Inversion Mutation » sur l'ensemble de 657 villes :

Méthode/Temps (sec)	*Random*	*Uniform*	*SinglePoint*	*SinglePoint2*	*Order*
5	779110.8707450775	790956.8326603968	417887.52118054463	306239.71620896074	155043.34411279968
10	771183.2986312967	788682.0984295909	304280.52130819164	214796.32785466142	79687.39870581796
20	767468.1596029422	786280.1132624426	199989.0027397327	141043.08239364586	33863.03262403997
30	781897.974549531	769226.281619413	133790.25985474425	100605.42748667585	20498.06392137557
40	767497.150980971	783268.8000398105	91655.06784031524	81133.969474449811	16444.701577444313
50	765417.0125363476	781517.3813956252	59837.10116455167	69315.6412921874	14352.174622014601
60	764801.5750782737	777587.5647485028	41118.00476755642	59113.250094786636	13154.420587499833
Ordre meilleur résultat	*4*	*5*	*2*	*3*	*1*

Nous observons ici que « Order » est de loin celui qui a les meilleurs résultats. C'est pour cela que nous l'avons choisi pour participer au championnat.

Néanmoins, nous pouvons également constater que SinglePoint2 qui semblait meilleur au début, c'est avéré moins bon sur la durée par rapport à SinglePoint (problème d'endurance ?).

Nous pouvons également constater que « Uniform » donne bel et bien de moins bons résultats que « Random », même s'il reste du même ordre de grandeur (comparé aux autres).

Conclusion

Comme nous avons pu le constater, utiliser les algorithmes génétiques (évolutionnaire) pour résoudre un problème NP-Complet tel que le voyageur de commerce est une bonne alternative à condition de bien configurer ses différents paramètres (pré-sélection, croisement, mutation, sélection survivant, et évidemment fonction de fitness).

En effet, en observant les tests que nous avons effectués, nous pouvons affirmer que le croisement « Order Crossover » allié avec des mutations de type inversion (« Inversion Mutation »), est le meilleur parmi ceux testés, ce qui va dans le sens de ce que l'on peut trouver dans la littérature.

Nous pouvons néanmoins faire remarquer que les configurations utilisées ici sont spécifiques (nous n'avons pas tout testé), ainsi pour le croisement « Uniform », il est possible que la configuration utilisée n'ait pas été appropriée. Nous l'avons tout de même présenté ici afin de montrer les risques d'une mauvaise configuration. En effet, nous avons pu constater que ses résultats sont moins bons que ceux d'une recherche aléatoire.

Enfin, après quelques modifications, nous avons pu constater des améliorations du résultat mais que nous n'avons pas pu autant tester que les algorithmes présents ici par manque de temps. Vous trouverez tout de même la description des modifications apportées et quelques-uns de leurs résultats en annexe.

Bibliographie

- **Altannar Chinchuluun - Panos M. Pardalos - Rentsen Enkhbat - Ider Tseveendorj.** OPTIMIZATION AND OPTIMAL CONTROL Theory and Applications, Chapitre: A Hybrid Evolutionary Algorithm for Global Optimization.
- **Hamdy A. Taha.** Investigacion de Operaciones 9 edición. Chapitre 11: Problema del agente viajero(Problème de TSP).
- **Heinrich Braun.** On solving Travelling Salesman Problems by Genetic Algoritms, Institut fiir Logik, Komplexithät und Deduktionssysteme, Universiät Karlsruhe.
- **Kusum Deep - Hadush Mebrahtu.** New Variations of Order Crossover for Travelling Salesman Problem, Department of Mathematics, Indian Institute of Technology, Roorkee, India.

Annexe

« SinglePoint Search 3 »

Il s'agit d'une variation de « SinglePoint 2 ». En effet, au lieu de ne muter que 10% des enfants générés par croisements, nous les mutons tous systématiquement une fois. Nous gardons toujours uniquement les meilleurs parmi les parents et les enfants.

« Order Search 2 »

Il s'agit là encore d'une variation mais cette fois-ci de « Order Search ». Dans ce cas nous réutilisons le principe de sélection de l'offspring (sélection des survivants), ou nous ne gardons que les meilleurs (au lieu de remplacer les deux moins bons). Nous n'avons pas changé le fait que nous mutons tous les enfants systématiquement.

Résultats

Voici les meilleurs résultats constatés en 10 exécutions de 60 secondes, en utilisant toujours « Inversion Mutation » :

Méthodes/Temps (sec)	SinglePoint 3	Order 2
60	12861.650095815854	12824.25828922671

Comme nous pouvons le constater, les résultats sont très proches. Et sur quelques exécutions que nous avons réalisées pour tester le fonctionnement de ces algorithmes, nous avons pu constater que le meilleur résultat varie (l'un ne semble pas vraiment meilleur que l'autre).

Nous pouvons noter qu'en comparaison avec les résultats du meilleur présenté ci-dessus (Order avec pour résultat 13 154 sur 60 secondes), ils semblent être un petit peu plus efficaces (à confirmer avec plus de tests).