The Burman Traveling Salesman Problem

LABO 8

Forestier Quentin & Herzig Melvyn

MLG – 11.06.2022

Expliquez brièvement le problème et votre solution

Ce laboratoire est axé autour du problème du voyageur de commerce. Dans ce problème d'optimisation, nous tentons de déterminer, parmi une liste de villes, quel est le circuit le plus court qui passe par toutes les villes une et une seule fois. À l'heure actuelle, il n'y a pas d'algorithme qui permet de résoudre ce problème en temps polynomial. Il est qualifié comme un problème NP-complet.

Dans le cadre de cette expérience, nous avons 14 villes dont les latitudes et les longitudes sont les suivantes :

Ville	Lat	Lon	Ville	Lat	Lon
0	16.47	96.10	**7**	17.20	96.29
1	16.47	94.44	**8**	16.30	97.38
2	20.09	92.54	**9**	14.05	98.12
3	22.3*	93.37	**10**	16.53	97.38
4	25.23	97.24	**11**	21.52	95.59
5	22.00	96.05	**12**	19.41	97.13
6	20.47	97.02	**13**	20.09	94.55

Les distances séparant les villes n'ont pas été considérées dans un plan Euclidien mais comme étant sur un sphéroïde aplati.

Dans ce laboratoire, nous sommes intéressés par l'obtention de la meilleure solution possible en utilisant un algorithme génétique (GA). Comme nous ne connaissons pas la solution optimale, il faudra exécuter l'algorithme plusieurs fois pour vérifier s'il ne trouve pas une meilleure solution.

Fournissez la meilleure route que vous avez trouvé et le chemin le plus court en termes de kilomètres. Est-ce que c'est le chemin optimal ? Expliquez.

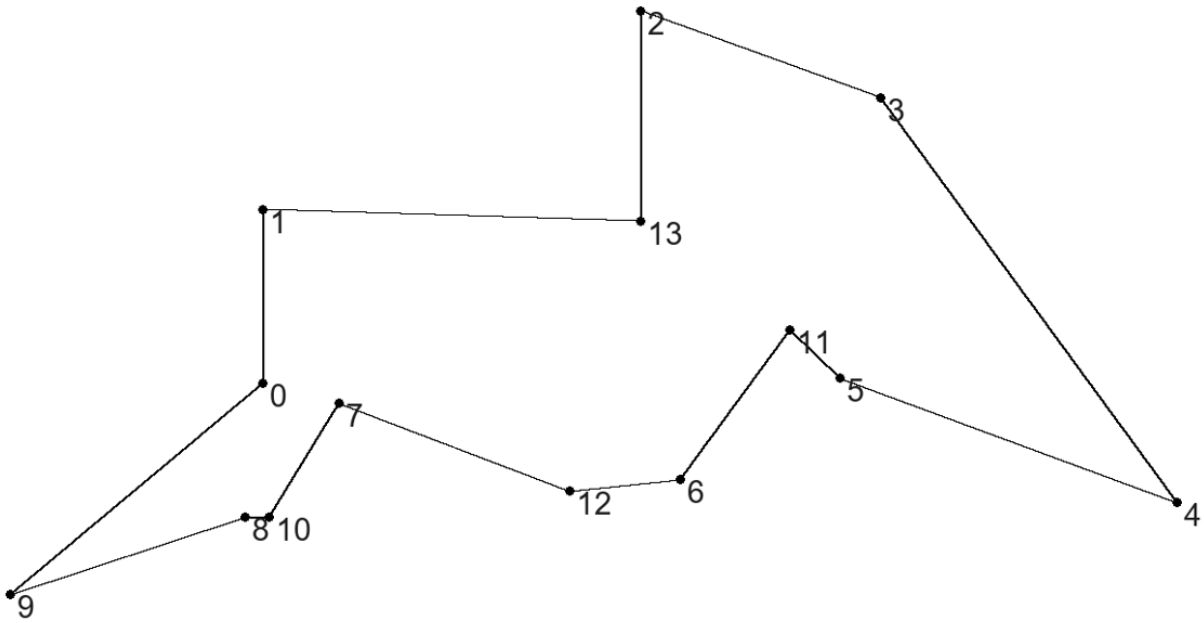

Figure 1 Meilleure route trouvée

La figure 1 présente la meilleure route que nous avons trouvé avec une distance de 3346.76 kilomètres. Nous pensons que ce chemin tend vers l'optimalité pour deux raisons :

- La première est purement visuelle. À l'œil, le chemin semble être intuitivement le plus court.
- La seconde est exploratoire. Après avoir effectué de nombreux tests avec des configurations différentes du GA, la distance 3346.76 a systématiquement été égalée mais jamais battue.

Bien que nous pensons que notre solution tend vers la solution optimale, nous affirmons qu'elle n'est pas foncièrement mauvaise puisqu'aucun segment du chemin n'en croise un autre, ce qui est une caractéristique incontestable d'une solution non optimale.

Décrivez votre fonction de « fitness »

Notre fonction de fitness est montrée en figure 2. Son fonctionnement est simple. Pour calculer le fitness d'un chromosome, nous calculons la distance totale du tour qu'il représente. Plus le tour est court, plus son fitness est petit. Évidement l'algorithme génétique va tenter de minimiser le fitness. Pour chaque ville à l'index « i » du chromosome, nous calculons la distance qui la sépare de la prochaine ville qui se trouve à l'index « i + 1 » du chromosome. Ainsi pour l'index 0 nous obtenons la distance entre la ville à l'index 0 et la ville à l'index 1. De la même manière, pour le dernier index, nous obtenons la distance jusqu'à la ville à l'index 0. Pour éviter de devoir recalculer les distances entre les villes à chaque exécution, la matrice *distances_matrix* est globale avec les distances en kilomètre entre chaque ville comme si elles étaient positionnées sur le globe terrestre.

```python
def fitness_tour_length(chromosome_tour):
    """Compute the fitness of a tour/chromosome"""
    global distances_matrix
    tour_length = 0.0
    nb_cities = len(chromosome_tour)
    for i in range(nb_cities):
        j = (i + 1) % nb_cities
        tour_length += distances_matrix[chromosome_tour[i], chromosome_tour[j]]
    return tour_length
```

Figure 2 Fonction de fitness

Expliquez de quelle manière vous avez encodé la solution, donnez un exemple de chromosome.

Les chromosomes sont encodés de manière à représenter un tour. Ils sont faits de 14 gènes qui sont des chiffres de 0 à 13 qui représentent un numéro de ville. La position du gène dans le chromosome représente la position de la ville qui lui est associé dans le tour. Un chromosome ne peut pas contenir deux gènes identiques, ce qui rendrait le tour invalide du point de vue du problème du voyageur de commerce.

0	4	3	1	2	8	10	11	13	9	6	7	12	5

Figure 3 Exemple de chromosome

Le chromosome ci-dessus donne le circuit :
0 -> 4 -> 3 -> 1 -> 2 -> 8 -> 10 -> 11 -> 13 -> 9 -> 6 -> 7 -> 12 -> 5 -> 0

Fournissez la configuration finale du GA que vous avez utilisé pour trouver vos meilleurs résultats : taux de mutation, taux de crossover, taille de la population, type de sélection, nombre de générations. Décrivez la méthodologie employée pour obtenir de meilleurs résultats.

Configuration finale :

Crossover rate	0.9
Mutation rate	0.01
Generations	150
Population size	100

Crossover	G1DListCrossoverOX
Selector	GTournamentSelector
Minimax	Minimize

Dans un premier temps nous avons fixé la méthode minimax pour minimiser le fitness. C'est un invariant car nous souhaitons avoir le circuit le plus court et non le plus long.

Ensuite, de manière naïve nous sommes inspirés de l'exemple du TSP disponible sur PyEvolve et nous avons défini notre crossover en *G1DListCrossoverOX* qui permet notamment de ne pas avoir des gènes en double. Puis arbitrairement, nous avons choisi de travailler avec une sélection[1] par tournoi plutôt que par roulette

À la suite de cela, nous avons défini des valeurs « aléatoires » pour le taux de crossover, le taux de mutation, le nombre de générations et la taille de la population.

À partir de ce point, nous avions un état initial duquel nous avons commencé une phase d'exploration.

Nous avons très aléatoirement tenté de modifier le taux de crossover, le taux de mutation, le nombre de générations et la taille de la population. À chaque modification d'un paramètre, nous effectuions quelques exécutions pour voir comment évoluait les résultats (individuellement et relativement à la configuration précédente). Puis, au fur et à mesure des itérations, nous avons obtenu une première série de paramètres qui produisaient des résultats satisfaisants.

Finalement, il restait à valider la méthode de crossover et la méthode de sélection. Comme en témoigne le résultat final, nous ne les avons pas modifiés par rapport à la configuration initiale. Nous avons tenté d'autres méthodes de crossover qui donnaient des résultats similaires ou moins bon. De ce fait, nous avons conservé la méthode initiale qui nous a permis d'arriver à des résultats satisfaisants. Concernant le sélecteur, nous avons tenté de passer sur une sélection par roulette mais encore une fois, les résultats obtenus ne valaient plus rien. Alors, nous sommes revenus sur la sélection par tournoi.

[1] La sélection par « ranking » a été interdite par le professeur.

Fournissez des graphiques pertinents de vos expériences et des explications.

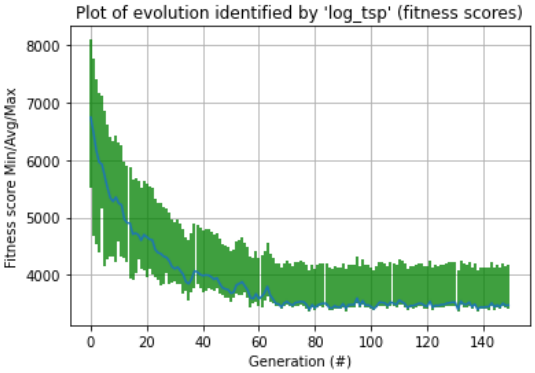

Figure 4 Meilleure configuration fitness vs générations

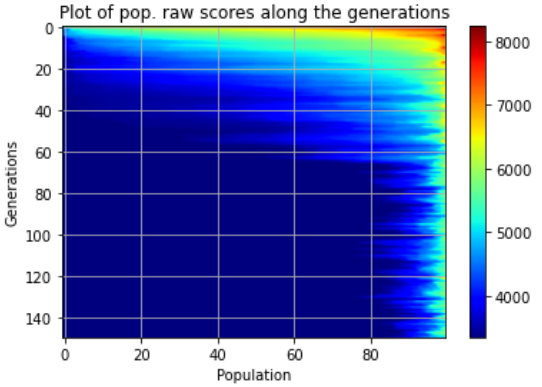

Figure 5 Meilleure configuration fitness heatmap

Les figures 4 et 5 montrent l'évolution du GA avec notre configuration finale. Nous pouvons voir que notre minimum est trouvé après 70 générations. Après ce point, comme le montre la heatmap, environ 80% de la population tourne autours du minimum. Le reste, environ 20%, qui doit être le fruit de mutations, tente d'explorer de nouveaux horizons mais en vain. Dans notre cas, l'évolution est assez stable. Les chromosomes qui mutent ne donnent pas de meilleurs résultats et reconvergent assez rapidement vers notre meilleur résultat connu.

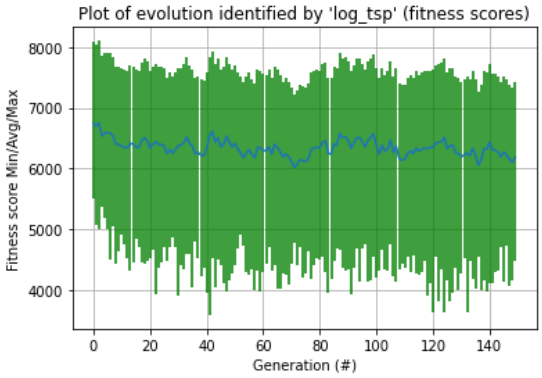

Figure 6 Configuration finale avec sélection par roulette, fitness vs générations

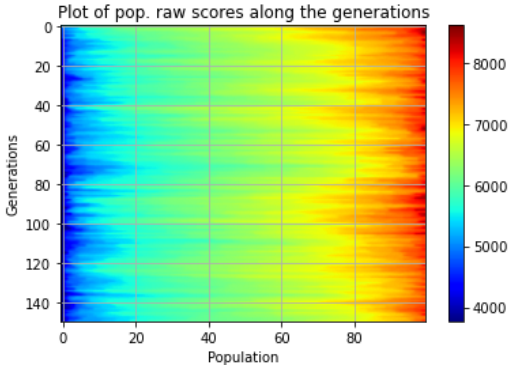

Figure 7Configuration finale avec sélection par roulette, fitness heatmap

Comme décrit dans le chapitre précédent, nous avons tenté de changer notre algorithme de sélection par tournoi par une sélection par roulette. Comme le montrent les figures 6 et 7, l'évolution est catastrophique. Il n'y a aucune amélioration après 150 générations. Nous en déduisons que la sélection par roulette n'est pas adaptée pour notre configuration. Cela ne veut pas forcément dire que cette sélection n'est pas adaptée pour le problème mais qu'elle nécessiterait une configuration différente pour fonctionner.

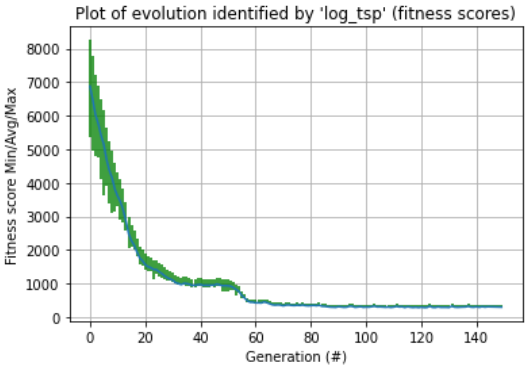

Figure 8 Fitness vs génération avec génomes non uniques.

Pour la science, nous avons changé la méthode de crossover pour laisser place à la méthode par défaut de G1DList qui autorise d'avoir deux génomes identiques. En figure 8, comme attendu, cela converge vers 0. Toutefois, dans cette exécution, cela stagne à 1000 entre la génération 30 et 50. Dans d'autres exécutions, cela converge directement à 0. Tout ça pour démontrer le caractère non déterministique des GA et que l'évolution prend du temps.

Conclusion

En vue des résultats obtenus, nous pouvons dire que les algorithmes génétiques sont une méthode prometteuse pour résoudre le problème du voyageur de commerce. Nous n'avons aucune certitude que notre solution soit la solution optimale. Toutefois, elle semble s'en approcher. Comme expliqué, aucun chemin ne se croise, ce qui est encourageant, sinon notre circuit serait automatiquement invalidé.

Bien que nous ne puissions pas dire si la solution obtenue est optimale avec certitude, nous pouvons néanmoins dire que nous l'approximons. Le problème est par nature très complexe. Toutefois, avec cette méthodologie nous arrivons très rapidement à un résultat prometteur.

Finalement, ce qui prend le plus de temps, c'est calibrer l'algorithme génétique jusqu'à atteindre un résultat qui semble satisfaisant. Ce qui aurait été intéressant, c'est de mesurer notre algorithme à d'autres configurations du problème du voyageur de commerce, dans l'idéal en ayant un résultat optimal comme référence.

De manière générale, il y a une longe phase d'exploration pour tenter de comprendre comment réagit l'algorithme avec ses différents paramètres en fonction du problème que nous souhaitons résoudre. Il faut tenter de comprendre comment évolue l'algorithme lorsque tel ou tel paramètre change. De plus, il ne faut pas négliger le hasard qui peut parfois donner des résultats surprenants comme nous l'avons vu avec la figure 8.