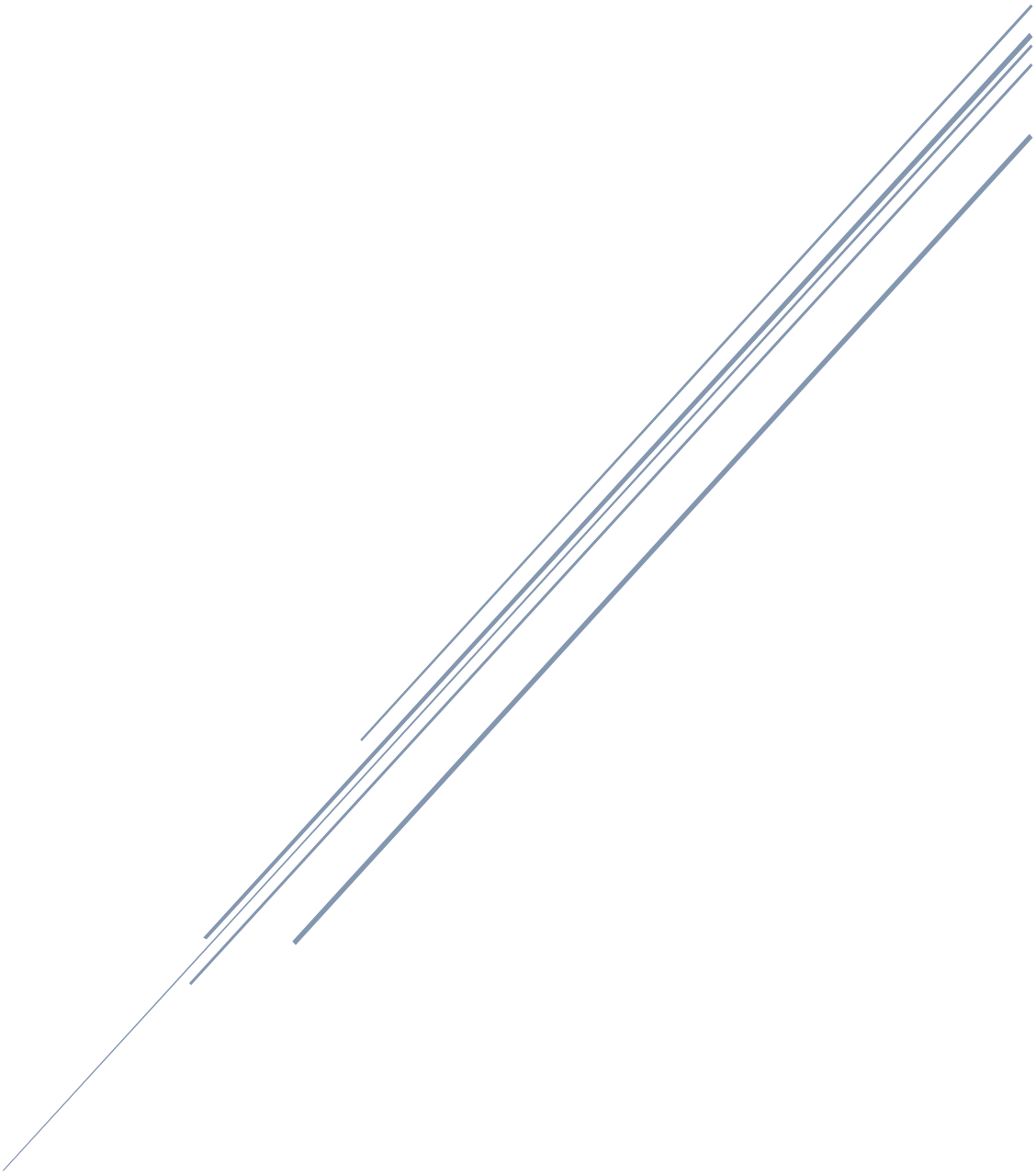
MACHINE LEARNING

Practical work 04 Genetic algorithm



HEIG-VD
Burgener François

1 Introduction

Le problème du voyageur de commerce est assez connu. Il demande de trouver un parcourt entre chaque ville le plus court possible en revenant au point d'origine. Le parcourt ne dois jamais passer plus d'une fois sur une ville mise a par la ville de départ à la fin du parcourt. Ce problème est un problème difficile. Ce problème est un NP-complet. Il nous est donc pas toujours possible de trouver avec certitude la solution optimale. En théorie il serait possible de trouver la solution optimale car nous pouvons avoir tous les chemins possibles mais cela nous prendra beaucoup trop de temps pour certain problème plus complet.

Dans notre cas nous utilisons se problème sur 14 villes en Burma via des coordonné latitude, longitude. Pour le calcule de la distance nous avons utiliser la formule de vincenty. Cette formule nous permet de calculer la distance entre deux points à la surface d'une sphère. Ces formules utilisent l'hypothèse que la terre est une sphère aplatie aux pôles ce qui permet d'obtenir des résultats plus précis qu'avec la formule de la distance du grand cercle.

2 Solution

Pour notre solution nous avons utilisé l'algorithme génétique afin d'éviter de devoir coder la solution du problème. L'algorithme nous permet de trouver la solution sans devoir coder le problème en lui-même. Avec cet algorithme, nous pouvons définir quel type de solution nous souhaitons obtenir afin qu'il trouve par lui-même la solution optimale. Le type de solution que nous avons définie est une liste de parcourt de chaque ville. L'algorithme va prendre les listes est prendre la meilleur, qui pour nous est le chemin le plus court. Nous utilisons donc une fonction d'évaluation qui va calculer la distance du parcourt en entier afin de pouvoir faire cette minimisation.

3 Provide the better route you found and the shortest path in kilometers. Is ist he optimal shortest path ?

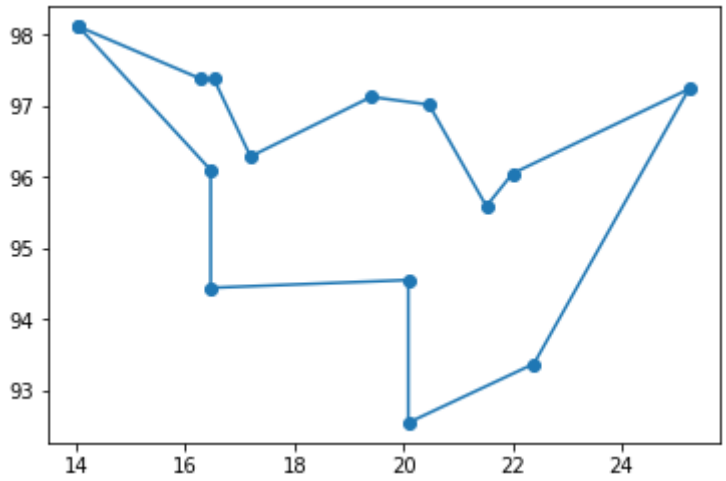
Meilleur parcourt : [9, 0, 1, 13, 2, 3, 4, 5, 11, 6, 12, 7, 10, 8].

Nous pourrons avoir d'autre parcourt similaire qui seront juste une rotation ou une inversion de notre solution. Le parcourt sera identique et donc la distance sera la même

Distance du meilleur parcourt : 3346.76 km

Notre parcourt est le plus optimale. Nous arrivons a trouvé le plus cours chemin en a peine 80 génération. J'ai fait des tests en allant jusqu'à 10'000 générations et les résultats étaient toujours les mêmes. Par contre il aurait été possible que notre solution ne soit pas optimal car dans certain cas il n'est pas possible de trouver une solution optimale (je parle dans un autre exemple que les 14 villes donné).

Résultat du chemin le plus court :



4 Fonction Fitness

Notre fonction fitness prend en paramètre un chromosome. Ce chromosome est une liste des villes parcourut. Ensuite nous calculons, en kilomètre, la distance du parcourt de toute ses villes en retournant a la ville d'origine. Ensuite nous retournant cette distance en tant que score. Ce score permettra a l'algorithme génétique de savoir si nous avons le meilleur parcourt ou non. Il gardera le meilleur parcourt en fonction de si on maximise ou minimum le résultat du score. Dans notre cas nous voulons minimiser la distance du parcourt.

5 Encoding solution

Pour coder la solution j'ai décidé de prendre une liste d'entier représentant le parcourt effectuer dans chaque ville. Cette liste a des chiffres entre 0 et 13.

Voici un exemple de chromosome possible : [9, 0, 1, 13, 2, 3, 4, 5, 11, 6, 12, 7, 10, 8]

Ces numéros nous permettront de récupérer les coordonnée (latitude, longitude) de chacune des villes car nous avons un tableau de coordonnée.

Tableau des coordonnées : [(16.47, 96.1), (16.47, 94.44), (20.09, 92.54), (22.39, 93.37), (25.23, 97.24), (22.0, 96.05), (20.47, 97.02), (17.2, 96.29), (16.3, 97.38), (14.05, 98.12), (16.53, 97.38), (21.52, 95.59), (19.41, 97.13), (20.09, 94.55)]

Comme ça dès que l'on récupère une valeur dans notre chromosome nous pouvons prendre ce numéro (qui est le numéro de la ville) et l'insérer en tant qu'index dans le tableau des coordonnés afin de récupéré ses coordonné. Ensuite il nous suffit d'utilises ses coordonnés dans notre fonction de calcule de distance pour connaitre la distance entre deux villes.

Pour initialiser notre chromosome, j'utilise la méthode proposer dans exemple TSP de pyevolve. Cette méthode nous créer une liste de la taille de notre génome allants de 0 à 13. Ensuite nous mélangeant cette liste afin d'avoir une liste aléatoire.

Pour ce qui est de la mutation j'utilise la méthode de swap qui va échanger aléatoirement des éléments d'un génome. Pour ce qui est du crossover je ne l'utilise pas même si je l'ai initialisé dans la solution. En ayant fait des tests avec j'ai pu remarquer que de ne pas l'utiliser nous donner de meilleur performance. Nous trouvons la solution optimale moin de générations qu'avec le crossoverrate

Pour cela j'ai utiliser le notebook **LaboGA.ipynb** .

6 Configuration of the GA

Pour la solution finale j'utilise comme paramètre :

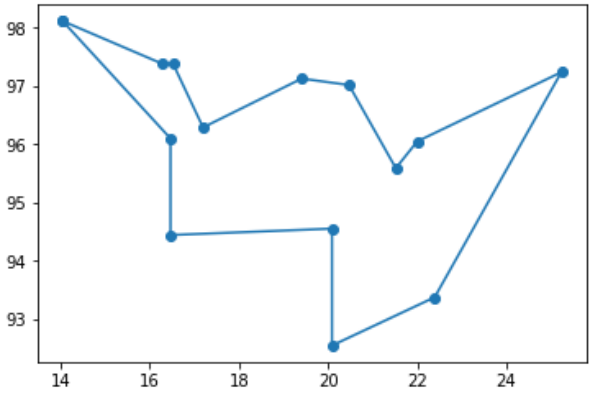
Generation : 500
Mutation : 0.1
CrossoverRate : 0.0
Population : 10
Selector : GrouletteWheel
Elitism : True

Comme je l'ai décrit plus haut, j'utilise la méthode swap pour les mutations. En se qui concerne le crossover je ne l'utilise pas et je le set à 0.0 se qui signifie qu'il est set a 0%.

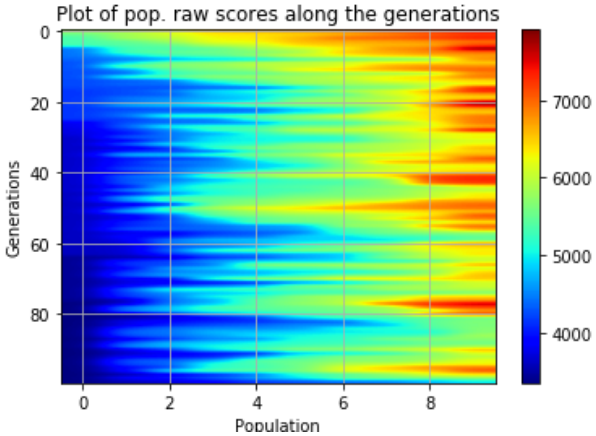
Au tout début de mes testes j'utilisais une population élevée afin de faire mes teste. Il me fallait donc un grand nombre de génération afin d'arriver a la solution optimale. J'ai donc préféré avoir une plus petite population afin de récupérer que les meilleurs résultats plus rapidement. Ensuite comme je n'utilisais pas le crossoverRate je n'avais que la mutation a changé. J'ai utilisé différente pourcentage de mutation afin d'arriver rapidement à la solution optimale. En ce qui concerne la génération j'ai préféré le laisser a 500 afin d'éviter de trouver la solution optimale en beaucoup de génération.

7 Provide relevant plots of your experiments and explanations

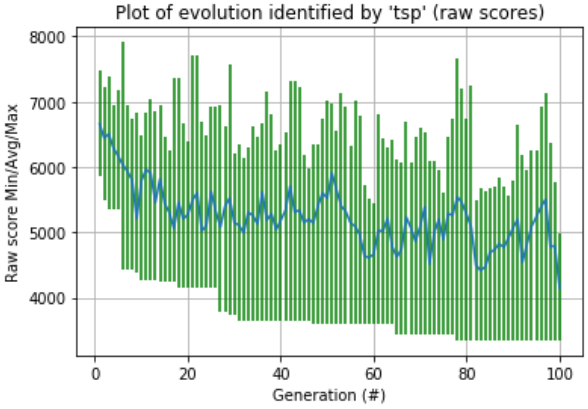
Ici nous pouvons voir le parcourt le plus court que nous avons obtenu. Chaque solution que nous trouvons aura le même résultat que le graphe si dessous. La seule différence et le sommet de départ et la direction dans le quel en parcourt les ville. Soit dans le sens des aiguilles d'une montre ou l'inverse. Cela nous donne donc 28 solutions de parcourt (14 dans un sens et 14 dans l'autre).



Ce graphe nous montre une carte de la distribution des scores brute de la population entre les générations. Ici nous pouvons voire vois qu'a partir de la génération 80 nous avons un grand nombre de la population qui est dans le bleu ce qui est pour nous une bonne chose car nous voulons que le score soit le plus bas possible c'est-à-dire dans le bleu.



Ce graphe nous permet de voir l'évolution du score entre la génération. Ici nous allons regarder le minimum est on peut voir qu'à partir de 80 le score ne change pas est qu'il reste au même stade. Cela confirme les résultats obtenus par l'algorithme génétique.



8 Conclusion

Pour conclure, nous pouvons remarque que le problèmes TSP, avec le set de donné qu'il ou était fournie, était un problème simple pour trouver la solution optimale. Avec la bonne configuration de notre algorithme générique, que se soit sur l'initialisation, les mutations ou alors le crossoverrate, nous pouvons trouver très rapidement ,et ça en peux de génération, la solution optimale. Au tout début j'avais de la peine a comprendre comment la GA pouvais trouver par lui-même une des solution optimale. Enfaite c'est grâce a notre fonction d'évaluation (fitness) qui permettais a l'algorithme génétique de dire si le chromosome choisie doit rester ou non dans la population. Si ce chromosome avait un meilleur score que les autres de la population, il était gardé et le plus mauvais chromosome était supprimer de la population. Cela permet de garder que les meilleurs individus dans la population.