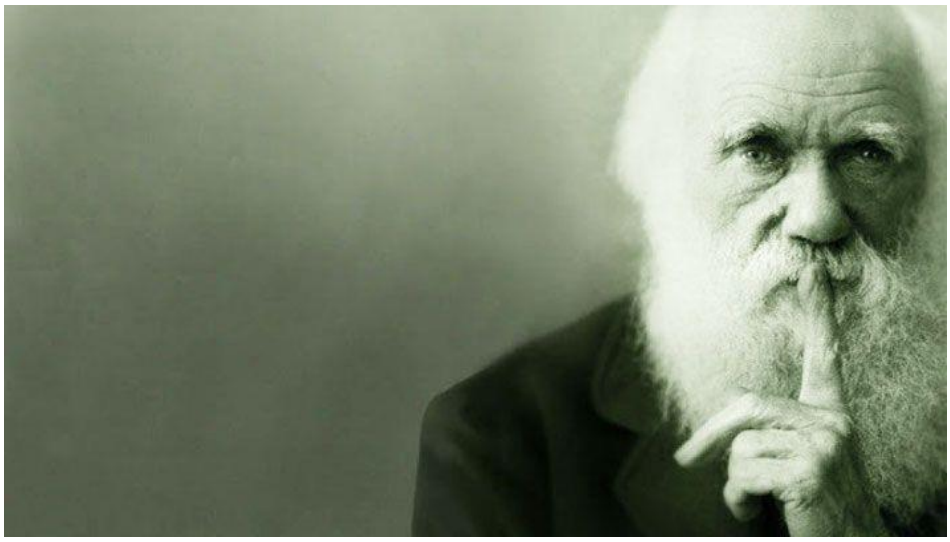
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

MACHINE LEARNING

PRACTICAL WORK TSP – GENETIC ALGORITHM

Lionel Burgbacher & David Jaquet

11.06.2020



6.1 INTRODUCTION

Dans ce laboratoire, nous avons redécouvert un célèbre problème. Il s'agit du « Traveling Salesman Problem » (TSP). C'est un problème d'optimisation. Le but de ce problème est de trouver le chemin le plus court passant par toutes les villes d'une liste une et une seule fois. De plus, il est important que nous ayons le plus grand cycle possible. Il faut donc que la première ville visitée soit la même que la dernière. La liste utilisée dans ce laboratoire contient 14 villes Birmanes. Cette dernière est utilisée pour représenter les gênes.

Le principe maintenant est d'utiliser l'algorithme génétique pour minimiser la distance totale. Pour calculer les distances entre les villes, sachant que la terre est sphérique, nous avons utilisé les géodésiques (distance la plus courte entre deux points sur une sphère) de la librairie `geopy`.

Nous allons donc vérifier s'il est possible de trouver la meilleure solution ou, du moins une solution acceptable, à ce problème avec l'aide d'un algorithme génétique sachant que c'est un problème difficile à résoudre.

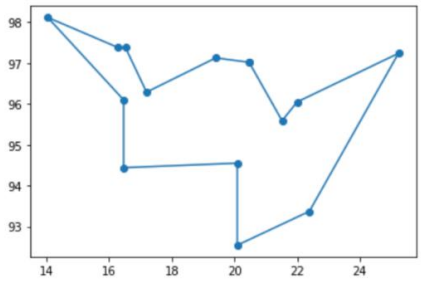
6.2 RÉSULTATS

Le meilleur résultat trouvé est le chemin suivant :

```
[12, 7, 10, 8, 9, 0, 1, 13, 2, 3, 4, 5, 11, 6]
```

Il est important de noter que nous devons avons avoir un cycle d'un graphe non orienté, la première valeur pourrait être n'importe laquelle, mais la suivante devrait toujours être celle qui la suit dans la liste, dans un sens ou dans l'autre. Un autre chemin possible serait, par exemple, `[12, 6, 11, 5, 4, 3, 2, 13, 1, 0, 9, 8, 10, 7]`.

Le parcours que nous avons trouvé a donc une distance d'environ de `3346 km`. Afin d'avoir un meilleur aperçu du cycle obtenu, nous avons décidé de l'afficher sur un plan orthonormé. Bien que cela ne se voit pas sur la liste ci-dessus, la ville de départ ainsi que celle d'arrivée est le numéro 12. Voici donc le résultat obtenu avec un graphique :



Il s'agit là de la meilleure solution trouvée, mais il ne s'agit pas obligatoirement la solution optimale à ce problème. Nous avons trouvé cette solution après plusieurs essais et différents paramètres.

Le professeur ayant donné l'indice `33xx kms`, sur teams, concernant la distance du meilleur chemin connu, on pense donc que notre algorithme est correct. Il existe pour ce problème $\frac{1}{2}(n-1)!$ solutions avec n le nombre de villes, ce qui est un nombre gigantesque. Avec $n = 14$, il existe 3'113'510'440 solutions différentes. Voilà pourquoi il est difficile d'être sûr que notre solution est bel et bien la solution optimale.

6.3 FONCTION FITNESS

Notre fonction fitness est toute simple, elle prend en paramètre une liste de position des villes. Ensuite, elle va additionner les distances entre chaque ville pour calculer la distance totale. On ajoute à la fin la distance entre la dernière ville et la première. Finalement elle retourne la somme obtenue. La distance entre les villes est calculée à l'aide des géodésiques de la librairie `geopy`. Il est important de noter que nous sommes partis du principe que la terre était une sphère parfaite afin de pouvoir utiliser les géodésiques.

6.4 SOLUTION

Notre solution est basée sur l'exemple `GA_evo-string` que nous avons adapté pour résoudre notre problème. Nous avons aussi trouvé des informations sur cette page, exemple 12 : http://pyevolve.sourceforge.net/0_6rc1/examples.html#example-12-the-travelling-salesman-problem-tsp

Les paramètres passés sont les indices de villes dans une liste.

Un exemple de chromosome est donc n'importe quelle combinaison de la liste des chemins.

Exemple : `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]`

6.5 CONFIGURATION DE L'ALGORITHME GÉNÉTIQUE

`Mutator` : nous avons utilisé le mutator qui va simplement intervertir les données, il n'y aura pas d'autre changement.

`Crossover` : dans la documentation, nous avons trouvé que le `G1DListCrossoverEdge` était le crossover parfait pour le TSP, il évite les doublons.

`Initializator` : pour éviter les doublons, nous avons utilisé celui dans l'exemple de `pyevolve`.

`Selector` : nous avons choisi le `Tournament Selector` pour que le gagnant de chaque tournois soit utilisé pour le crossover.

Pour minimiser la solution, nous avons utilisé la méthode `setMinimax` avec `minimize` comme type.

`Generations` : nous avons obtenu de bons résultats avec 200 générations, mais pour conserver de la marge avec l'aléatoire, nous avons opté pour une solution effectuant 300 générations.

`Crossover Rate` : nous avons trouvé un taux de 100% après plusieurs essais. La solution est stable.

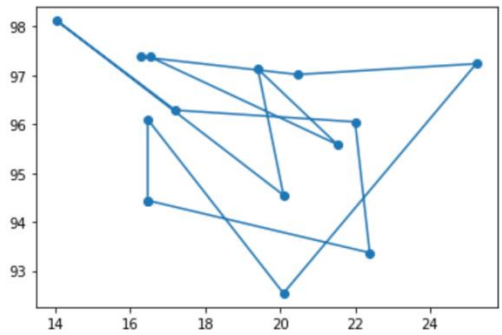
`Mutation Rate` : Concernant le `Mutation Rate`, un taux de 1% nous a permis d'avoir de bons résultats, nous avons donc décidé de le laisser au plus bas.

`Population Size` : nous avons commencé par une population de 500 avant de terminer à 100 ou les résultats semblent stables.

6.6 GRAPHES ET EXPÉRIENCES

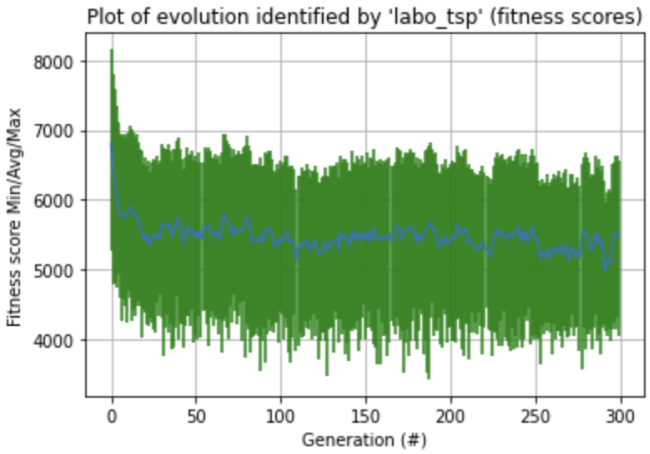
Nous avons tout d'abord constaté un problème dans notre algorithme, nous avions une valeur plus basse que la solution du professeur. Nous avons voulu voir le chemin parcouru et nous avons constaté que nous avions effectivement un problème.

```
[1, 3, 5, 7, 9, 13, 12, 11, 10, 8, 6, 4, 2, 0]
2875.4216129880633
```



Dans le graphique ci-dessus, il est évident que le chemin effectué n'est pas le plus court. Le problème venait de notre calcul dans la fonction de fitness. Une fois ce dernier corrigé, nous avions un graphique plus logique et une solution qui correspondait aux indications du professeur. Le schéma peut être trouvé dans le chapitre **6.2 Résultats**.

Nous avons aussi affiché l'évolution du score de la fonction fitness en fonction des générations. Pour cela nous avons utilisé le fichier pyevolve_graph.py trouvé sur le Github de pyevolve avec Python 3. D'autres graphs sont disponibles sur le fichier labo_tsp.ipynb (ou directement sur le Github). On constate sur le graphique que la meilleure solution est trouvée entre les générations 150 et 200.



6.7 CONCLUSION

Cet exemple d'utilisation d'un algorithme génétique nous a démontré leur puissance. Avec des logiciels plus classiques, le temps pour trouver une solution acceptable est bien plus long. De plus, bien que nous utilisions des libraires, le code est très compréhensible et très court, ce qui permet de bien comprendre chaque partie. Une aide non négligeable nous a été apportée avec les laboratoires 6 et 7 ainsi que les nombreux exemples trouvés. Cela nous a permis de gagner beaucoup de temps.

6.8 LIENS IMPORTANTS

Notre laboratoire est disponible sur Github à l'adresse suivante :
https://github.com/djaquet5/MLG_TravelingSalesmanProblem/blob/master/files/labo_tsp.ipynb

En plus des laboratoires précédents, nous avons utilisé la documentation de pyevolve ainsi que ses exemples. Nous avons principalement utilisé l'exemple numéro 12 qui est une résolution de l'algorithme. L'exemple est disponible à l'adresse suivante :
http://pyevolve.sourceforge.net/examples.html#example-12-the-travelling-salesman-problem-tsp