

# Machine Learning

## Practical work TSP – Genetic Algorithm

Lionel Burgbacher & David Jaquet

11.06.2020

Une image contenant homme, personne, intérieur, cravate

Description générée automatiquement

## 6.1 Introduction

Dans ce laboratoire, nous avons redécouvert un célèbre problème. Il s'agit du Traveling Salesman Problem. C'est un problème d'optimisation. Nous avons une liste de villes et le but de déterminer le chemin le plus court pour parcourir chaque ville et terminer dans la ville de départ sans repasser par une ville déjà visitée.

Nous avons représenté les gênes par une liste content les indices des villes. 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 :

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

À noter qu'il faut le voir comme un anneau, la première valeur pourrait être n'importe laquelle mais la suivante devrait toujours être celle qui la suit dans la liste. Un autre chemin possible serait par exemple [12, 6, 11, 5, 4, 3, 2, 13, 1, 0, 9, 8, 10, 7].

Il a une distance totale de, à peu près, 3346 km.

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

Le professeur ayant donné l'indice 33xx, sur teams, concernant la distance du meilleur chemin connu, on pense donc que notre algorithme est correct. Il existe pour ce problème solutions avec n le nombre de ville, ce qui est un nombre gigantesque avec n = 14, voilà pourquoi il est difficile d'être sûr que notre solution est optimale.

## 6.3 Fonction fitness

Notre fonction fitness est toute simple, elle prend en paramètre une liste de position des villes. Elle va ensuite additionner les chemins entre chaque ville pour y calculer le totale. On ajoute à la fin la distance entre la dernière ville et la première. Finalement elle retourne la distance obtenue. La distance entre les villes est calculée à l'aide des géodésiques de la librairie geopy.

## 6.4 Solution

Notre solution est basée sur l'exemple GA\_evo-string que nous avons adapté pour résoudre notre problème.

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 swaper 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 Tourament 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 obtenus de bons résultats avec 150 générations mais pour conserver de la marge avec l'aléatoire, nous avons opté pour une solution à 200 générations.

Crossover Rate : Nous avons trouvé un taux de 70% après plusieurs essaies. La solution est stable.

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

PopulationSize : Nous avons commencé par une population de 500 avant de terminer à 90 ou les résultats semblaient 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.

Une image contenant texte, carte

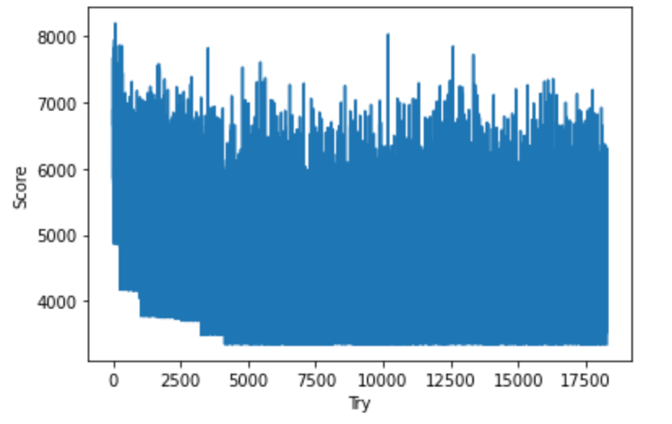
Description générée automatiquement

Après avoir résolu notre problème dans la fonction fitness nous avions un graphique plus logique et une solution qui correspondait.

Une image contenant texte, carte

Description générée automatiquement

Nous avons aussi affiché l'évolution de la fonction fitness pour voir s'il y avait une convergence vers la solution ou si chaque essaie était indépendant. On voit bien que plus l'algorithme avance, plus la solution est bonne.



## 6.7 Conclusion

Cet exemple d'utilisation d'un l'algorithme génétique nous a démontré leur puissance. Avec des logiciels plus classiques, le temps pour trouver une solution acceptable est bien plus longue. 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 apporté avec les laboratoires 6 et 7, ce qui nous a permis de gagner beaucoup de temps.