Volley-ball pools optimization

1 - Introduction

Étant joueur de volley dans une poule de N3, je me suis intéressé à la question de l'optimisation de la répartition des villes dans des poules sportives, lors d'un énième déplacement de plusieurs heures à l'extérieur. Afin de mieux comprendre comment optimiser ce genre de problèmes, j'ai tenté de créer un algorithme génétique qui pourrait trouver une répartition satisfaisante des 72 équipes de ma catégorie en France en 6 poules, pour comparer cette répartition avec celle réalisée dans la vie réelle.

2 - Obtention et pré-traitement des données

Afin de pouvoir proposer une solution à ce problème, il a tout d'abord fallu obtenir une base de données contenant les 72 villes en question, ainsi que leurs coordonnées GPS afin de pouvoir calculer des distances entre ces dernières. Après une tentative de scrapping des données infructueuse sur le site de la ffvb, j'ai finalement décidé de créer la base de lon jeu de données manuellement en entrant les noms des villes de chaque poule dans des listes python. Ces villes sont ensuite ajoutées dans un dataFrame pandas. J'utilise ensuite le package Nominatim qui permet de faire des requêtes pour obtenir les coordonnées GPS de chaque ville.

On peut ensuite afficher ces données sur une carte grâce au package Folium. Cela permet notamment de vérifier visuellement certains problèmes de coordonnées (un nom de ville que j'avais rentré me donnait notamment une position au Canada) et avoir une meilleure vue d'ensemble du problème. Cela est également important pour l'affichage final des résultats

Grâce à l'affichage des villes sur la carte, il a été assez facile de comprendre pourquoi les déplacements de cette année sont assez longs. En effet, malgré la taille de la région qu'est la nouvelle Aquitaine, le club de Talence se retrouve à être le plus isolé à l'échelle de toute la France pour cette catégorie.

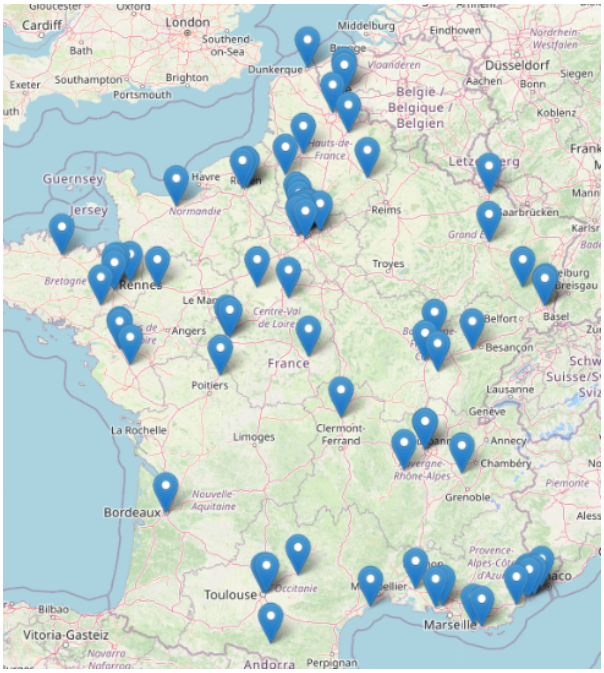

Figure 1 : Position de toutes les clubs de N3 en France

3 - Construction des premières fonctions de base

Après une première phase de travail sur les données. Certaines fonctions élémentaires ont été construites pour pouvoir utiliser un algorithme génétique plus tard.

3.1 - Initialisation de la population

On initialise une population de 100 chromosomes. Chaque chromosome contient 72 tuples de forme suivante : (ville : poule). Cela nous permettra par la suite de faire des opérations pour changer les villes de poule.

Une manière plus simple de créer les chromosomes aurait sûrement été de renseigner uniquement les numéros de poules, et considérer que l'index 0 est la ville 'Saint Avertin', la ville à l'index 1 'Tarascon' etc. Cela aurait sûrement permis d'améliorer la vitesse de l'algorithme.

```
Examples of chromosomes :
[('Saint avertin', 0), ('Tarascon', 4), ('Nantes', 0), ('Puygouzon', 2), ('Toulouse', 4), ('Talence', 5), ('Balma ', 3), ('Reze', 4), ('Chatellerault', 2),
[('Saint avertin', 5), ('Tarascon', 1), ('Nantes', 2), ('Puygouzon', 1), ('Toulouse', 5), ('Talence', 1), ('Balma ', 3), ('Reze', 4), ('Chatellerault', 1),
[('Saint avertin', 0), ('Tarascon', 0), ('Nantes', 3), ('Puygouzon', 4), ('Toulouse', 5), ('Talence', 1), ('Balma ', 1), ('Reze', 4), ('Chatellerault', 1),
[('Saint avertin', 1), ('Tarascon', 4), ('Nantes', 0), ('Puygouzon', 5), ('Toulouse', 2), ('Talence', 3), ('Balma ', 2), ('Reze', 1), ('Chatellerault', 0),
[('Saint avertin', 5), ('Tarascon', 1), ('Nantes', 1), ('Puygouzon', 0), ('Toulouse', 1), ('Talence', 5), ('Balma ', 5), ('Reze', 4), ('Chatellerault', 0),
```

Figure 2 : Exemple de 5 débuts de chromosomes

3.2 - Calculs de distance

Plusieurs fonctions ont été créées pour calculer les distances. Parmi elles, on peut retrouver une fonction qui calcule la distance entre deux points grâce à leur latitude et leur longitude.
Ce choix de calcul de distance à vol d'oiseau a été retenu pour simplifier le problème, mais il aurait été possible d'obtenir la distance réelle à parcourir entre deux clubs en voiture.

Ensuite, une fonction qui permet de calculer la distance parcourue au sein d'une même poule a été faite. Elle prend en argument la chromosome et calcule les distances entre chaque ville d'une même poule.

3.3 - Fonction de fitness

Afin d'évaluer les chromosomes de notre population, on utilise une fonction de fitness qui nous donne la somme des distances parcourues à l'intérieur de chaque poule. Cela nous permet donc d'évaluer assez grossièrement et simplement de savoir à quel point les villes sont bien réparties dans différentes poules.

Cependant, plusieurs améliorations sont à faire. Ces dernières seront détaillées dans les axes d'amélioration.

Application de la fonction de fitness à la poule réelle

Les fonctions de calcul de distance par poule et de fitness ont été appliquées à la configuration de poule réelle, afin de pouvoir comparer nos résultats à ces derniers. Voici les valeurs obtenues :

```
Distance dans chaque poule :
 {0: 383.262, 1: 180.948, 2: 274.220, 3: 165.408, 4: 200.758, 5: 377.0546}

Distance totale : 1581.653
```

On peut observer que la répartition des distances entre les différentes poules est assez inégale dans la vraie vie. En effet, pour les poules 0 (qui contient Talence) et 5, la distance totale parcourue est d'environ 380, et d'environ 270 pour la poule 2. Pour les trois autres poules, elle est de 200 ou moins.

On obtient une distance totale de 1580 avec la fonction de fitness choisie. On va donc tenter de s'en approcher ou bien de l'améliorer.

4 - Erreurs dans la première version de l'algorithme

4.1 - Résultats

En faisant tourner le premier algorithme, les résultats ont été absolument catastrophiques en termes de performances et de temps. Pour environ 3h d'exécution, seulement quelques dizaines de populations ont pu évoluer et les résultats ne s'amélioraient pas. La fitness de ces dernières orbitait autour de la fitness de la première population aléatoire, soit environ 3500.

4.2 - Causes

1 - Calcul des distances

Une première erreur assez simple à éviter au niveau du temps est la suivante : je calculais la distance entre chaque ville de même poule dans tous les chromosomes, et dans toutes les populations. Cela entraîne des temps liés aux calculs de distance extrêmement longs à chaque itération de l'algorithme. Dans la deuxième version, cela a été corrigé en créant en amont une matrice de distance entre chaque ville.

2 - Fusion des chromosomes

Pour faire évoluer les populations, j'ai premièrement utilisé une fonction de crossover. Cette dernière formait un chromosome enfant à partir de deux chromosomes parents. Pour ce faire, elle initialise aléatoirement un point de crossover entre les deux listes de parents. Elle prend donc la première partie d'un chromosome et lui ajoute la seconde partie de l'autre chromosome.

Cela posait plusieurs problèmes. Premièrement, il fallait effectuer de nouveaux calculs sur ce chromosome enfant, car rien ne garantissait (et c'est même plutôt l'inverse) que le nombre de villes par poules allait être conservé. Il faut donc appliquer des opérations coûteuses en temps pour re-placer les villes afin d'obtenir 12 villes par poule.

Le second problème est que le numéro des poules ne signifie pas la même chose d'un chromosome à l'autre. Par exemple, la poule 0 peut contenir des villes plutôt situées au Sud de la France pour un chromosome, mais aussi des villes plutôt au Nord pour un autre, ou bien même des villes distribuées totalement aléatoirement géographiquement parlant. Fusionner les villes d'une poule A d'un chromosome avec les villes d'une poule A d'un autre n'a donc aucun sens, et peut donc mener à des résultats quasiment aléatoires. Il a donc été décidé dans la

seconde version de l'algorithme de se débarrasser de cette fonction, pour la remplacer par une fonction de mutation

5 - Deuxième version de l'algorithme génétique

5.1 - Calcul en amont des distances entre chaques villes

On crée en amont une numpy array de taille (72,72) qui stocke toutes les distances entre chaque ville. Il est utilisé pour calculer la fitness de chaque chromosome.

5.2 - Rangement des chromosomes en fonction de leur fitness

Après avoir calculé les fitness des 100 chromosomes d'une population, on les ordonne afin de pouvoir réaliser deux actions importantes pour la création d'une nouvelle population :

1 - Garder les 20 meilleurs

A chaque itération de l'algorithme, on garde les 20 meilleurs chromosomes sur 100 pour les ajouter d'office à la nouvelle population. Cela permettra de ne pas perdre les meilleurs résultats obtenus, et on peut supposer que l'on a plus de chance d'obtenir de bons nouveaux chromosomes en mutant les chromosomes qui étaient déjà performants. On y ajoutera 80 enfants pour créer une nouvelle population de 100 chromosomes. Voyons comment sélectionner les parents qui les créeront ci-dessous.

2 - Obtenir de 'bons parents'

Grâce à la liste de fitness ordonnée, on peut attribuer un rang à chaque chromosome, allant de 0 à 99. Grâce à ce rang, on calcule une probabilité de devenir un parent, avec une probabilité augmentant en fonction du rang. On obtient donc un vecteur de probabilité, et avec un np.random.choice combiné à ce dernier, on obtient une liste de 40 parents.

On a donc des chromosomes qui sont à la fois dans les 20 meilleurs et dans les parents en même temps.

5.3 - Mutation des parents

Pour obtenir 80 enfants, on crée deux enfants par parent. Dans notre cas, la mutation est une permutation entre les poules de n*2 villes. Le paramètre ‘n’ est choisi entre 1 et 5, afin d’appliquer uniquement une légère modification au chromosome, qui avait été sélectionné car il a des grandes chances d’être ‘bon’.

Finalement, on crée facilement une nouvelle population en concaténant les 20 meilleures chromosomes avec les 80 chromosomes enfants d’une population.

5.4 - Résultats

```
Distance per pool :
{0: 232.726, 1: 170.748, 2: 391.538, 3: 262.947, 4: 53.243, 5: 180.948}

Total distance: 1292.154

Execution time : 3384 seconds ~ 56 minutes
```

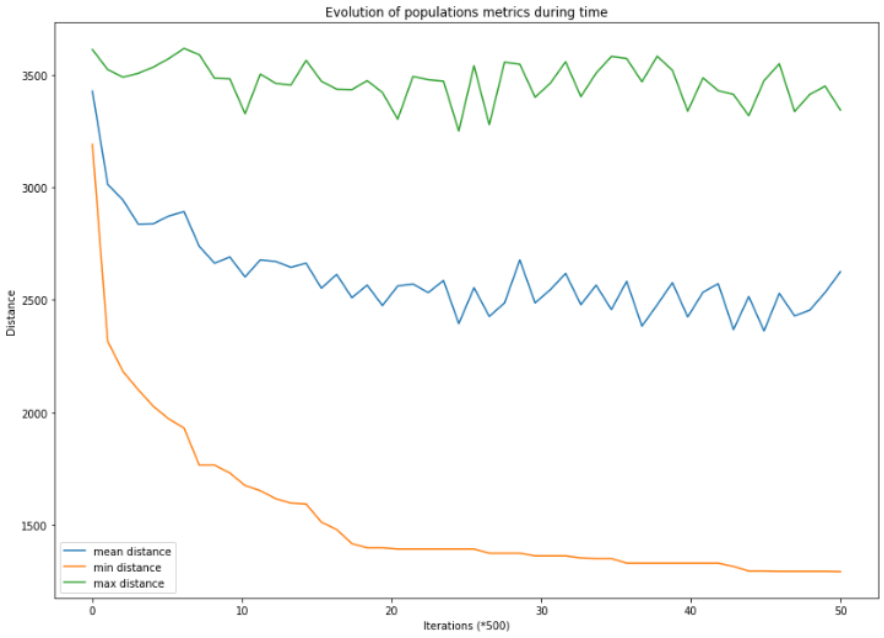

Figure 3 : Evolution des métrique de fitness par itération

On peut remarquer que la fitness du meilleur chromosome trouvé est plutôt bonne par rapport à celle trouvée dans la configuration réelle. En effet, on passe de 1580 à 1292. Cependant, la répartition des distances dans les poules reste très inégalitaire, avec une poule à environ 400 et une autre à environ 50.

On peut de plus voir que la fitness minimale descend assez vite sous la barre des 1500 en relativement peu de temps, mais peine à s'améliorer après.

Figure 4 : Affichage du résultat final

On peut assez facilement observer l'inégalité de distance entre les poules sur cette image. En effet, les équipes de la poule parisienne en rouge ne se déplacent que très peu, à l'inverse des équipes de la poule violette (qui contient d'ailleurs Talence…).

Cependant, à part quelques exceptions, la répartition des villes dans les poules semble cohérente par rapport à la fonction de fitness indiquée, et répond à peu près au besoin initial.

6 - Axes d’amélioration

6.1 - Modification de la fonction de fitness

L’algorithme génétique permet de trouver une bonne solution par rapport à la fonction de fitness choisie. Cependant, il ne permet pas d’obtenir une configuration de poule qui paraît réellement envisageable. En effet, l’hétérogénéité entre les poules est beaucoup trop forte, et certaines une des poules contient des villes bien trop dispersées.

Une solution pourrait être d’adapter la fitness pour trouver une configuration qui permette de rendre les déplacements de chaque ville plus homogène, et ce pour chaque déplacement (même si la moyenne des déplacements est acceptable, on ne peut pas faire de trajets trop longs sur un week-end pour un match). Utiliser la notion d’écart type pourrait être une solution afin de répartir les déplacements équitablement autour d”une certaine moyenne.

6.2 - Amélioration du calcul de la fitness

Un des problèmes majeurs dans l’algorithme actuel est que la fitness pour chaque chromosome est recalculée à chaque nouvelle itération pour la nouvelle population. Or, on connaît déjà la fitness pour les 20 premiers chromosomes car elle a été calculée au tour précédent. De plus, même pour les 80 enfants, on ne permute qu’au maximum les poules de 10 villes. On pourrait donc uniquement calculer la modification de la fonction de fitness engendrée par la permutation de plusieurs villes de poules, plutôt que de la recalculer entièrement.

6.3 - Amélioration du système de mutations

Quand on arrive à un certain nombre d’itérations, le nombre de mutations que l’on fait peut devenir un problème. En effet, l’amélioration de la fitness est très longue à partir d’un certain seuil, car la meilleure configuration de poule trouvée est déjà ‘bonne’. Or le nombre de mutations faites sur les chromosomes parents varie entre 1 et 5 (ce qui correspond à 2 à 10 permutations de poules), et il est très peu probable que l’on trouve une nouvelle configuration de poule qui soit meilleure avec 10 permutations quand on est déjà dans une ‘bonne’ configuration..

Il semble donc plus efficace de faire varier le nombre de permutations en fonction du numéro d’itération auquel on se trouve, en prenant un nombre de permutations plus élevé au début, afin de trouver de ‘bonnes’ et de le faire diminuer au fil du temps pour arriver uniquement à des permutations entre 2 poules et affiner nos résultats plus efficacement.