TP2 : Traveler Salesman Problem (TSP)

1. Changements apportés au template initial

a. La classe point2D

La première décision d'implémentation concerne la représentation des villes. Plutôt que de les représenter sous la forme d'une matrice de distance, nous allons les représenter par leurs points dans un repère cartésien. Ces points vont être représentés à l'aide de la classe point2D.

```
class point2D:
    x : float
    y : float
    id : int

    def __init__(self, id : int, x : float, y : float):
        self.id = id
        self.x = x
        self.y = y

    def dist(self, point : object):
        if isinstance(point, point2D):
            return sqrt((point.x - self.x)**2 + (point.y - self.y)**2)
```

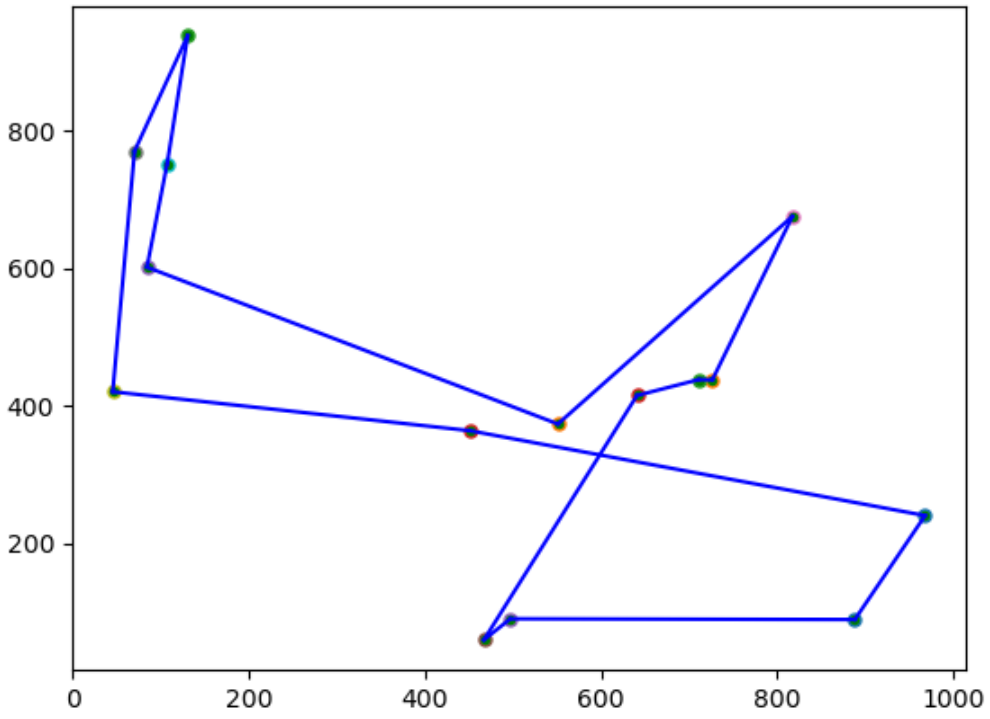
Comme on peut le voir sur la capture ci-dessus, cette classe prend 3 paramètres, sa position en x, sa position en y, tous les deux sous forme d'un nombre flottant, ainsi qu'un entier qui servira à les identifier. De plus cette classe implémente aussi la fonction dist permettant à un point de retourner sa distance par rapport à un autre point en utilisant la formule de la distance entre deux points.

b. Les fonctions cal_distance et voisinage

Les fonctions cal_distance et voisinages ont été modifiées, la première par rapport à la nouvelle implémentation et aussi pour utiliser au mieux la nouvelle fonction dist de point2D. La fonction voisinage quant à elle a été modifiée pour utiliser moins de paramètres.

c. L'affichage

Afin de mieux se représenter la solution finale, l'affichage à été modifié afin de pouvoir représenter graphiquement la solution à l'aide de matplotlib.pyplot, L'affichage des villes se fait donc avec un point de 5 de diamètre et on représente les solution comme ceci :



Où chaque point représente une ville et chaque segment que la distance entre les deux villes aux extrémités de ce segment a été parcourue par le voyageur de commerce.

2. Recuit simulé

Le recuit simulé n'a pas tant changé par rapport au template de base. ce qui a été modifié est une condition d'arrêt par limite de température à la place de changer par un nombre d'itération. Ce qu'il fait qu'on est à peu près sûr d'obtenir une convergence qui se fait vers les 5 de température. Le facteur de baisse de la température aide aussi à une convergence rapide.

3. Algorithme génétique

Choix d'implémentation :
Opérateur de mutation 2-opt plus optimisé pour ce genre de problème. (Il s'agit d'un opérateur qui inverse toutes les villes entre 2 villes données.
Opérateur de sélection par roulette.
Opérateur de croisement simple par pivot aléatoire.

4. Optimisation par colonie de fourmie

Choix d'implémentation :

Taux d'évaporation à 0.95 pour ne pas perdre trop vite les phéromones déposés.
Au niveau des calculs de probabilités de sélectionner une ville avec une fourmis, on ajoute une constante dans celui-ci pour encourager les fourmis à explorer leur environnement (fixé à 1 dans le code).
On fixe alpha et beta à 1 afin de considérer tout autant les distances entre les villes et les phéromones présents sur la route.

5. Comparaison

Après générations de 4 instances de TSP à 5, 10, 15 et 20 villes nous pouvons comparer le temps de résolution de ces différentes heuristiques en les comparant sur différentes valeurs (Fitness du meilleur individu, temps de résolution …)

	Recuit simulé		Génétique		ACO	
	fitness	temps	fitness	temps	fitness	temps
0.csv (5 villes)	2978	0.6s	2978	2.3s	2978	1s
1.csv (10 villes)	3038	0.9s	3038	3.4s	3327	3.8s
2.csv (15 villes)	2839	1.2s	3337	4.5s	4417	10.3s
3.csv (20 villes)	3927	1.3s	5334	5.4s	7117	23.1s

Comme nous pouvons le voir, le recuit simulé permet de trouver des meilleures solutions plus rapidement. L'ACO quant à lui est plutôt lent et ne permet pas de trouver de bonnes valeurs.
On peut penser que l'algorithme génétique nécessite un meilleur paramétrage (ajout d'individus) et pourquoi pas un 2e algorithme de croisement et de mutation pour diversifier encore plus la population. Du côté de l'ACO son temps d'exécution peut s'expliquer dans le fait que l'heuristique utilisée place 1 fourmi par ville et donc augmente la population en fonction du nombre de villes. Cependant le fait que la fitness soit mauvaise doit s'expliquer par les paramètres. Un meilleur choix de paramètres permettrait sûrement d'obtenir un meilleur résultat.

Conclusion

Ce TP nous à permis de prendre en main plusieurs heuristiques et de les appliquer au TSP. Les résultats ne semblent pas satisfaisants mais celà peut s'expliquer simplement avec le peu d'expérience sur l'utilisation de ces heuristiques (Notamment l'ACO) qui rend plus complexe le choix des paramètres.