Structure du code :

Le code python a été réalisé en plusieurs fichiers :

- main.py
- gui.py
- graph.py

Le fichier main.py permet de lancer une simple fonction main qui crée une instance de la classe GUI située dans le fichier gui.py. Cette classe GUI permet tout l'affichage graphique du code situé dans la classe GraphTheory présente dans le fichier graph.py. Le code fonctionne également sans affichage graphique mais pour cela il faut instancier la classe et appelé la méthode désirée dans le fichier graph.py. Toute la partie contenant les heuristiques est situé dans la classe GraphTheory.

graph.py

Les méthodes de la classe GraphTheory reprennent les heuristiques de graphe qui sont : le plus proche voisin, la plus proche insertion, Fletcher, Fletcher et Clarke, ainsi que les deux améliorations locales, extraction réinsertion et 2-OPT.

Les quatre heuristiques prennent en paramètre le sommet de départ qui doit être renseigné par l'utilisateur, elles retournent toutes le chemin ainsi que la distance de parcours de ce dernier. Quand aux 2 méthodes d'améliorations locales elles prennent en paramètres le chemin et la distance du résultat de l'heuristique précédemment réalisé. Il y a une gestion d'erreur dans chacune de ces méthodes, si l'utilisateur renseigne un sommet invalide le programme ne plante pas ou ne renvoie pas d'erreur.

GraphTheory utilise la bibliothèque python networkx qui permet de réaliser des graphes et d'utilisé des méthodes associées à la théorie des graphes tels que le degré des sommets ou les sommets adjacents. L'affichage d'un tracé de ce graphe se fait avec la bibliothèque matplotlib.

Pour initialiser cette classe l'utilisateur doit renseigner un fichier contenant une matrice de distance. La méthode constructeur __init__ crée également un graph orienté ou non orienté (grâce à la méthode Symmetric) selon la matrice que l'utilisateur a renseigné et qui a été chargé grâce à la méthode load_data. Le graphe est complété en y ajoutant les sommets et arrêtes avec leur poids respectifs contenu dans la matrice de distance grâce aux méthodes d'aide qui sont add_edges et add_nodes.

La classe GraphTheory possède également deux méthodes permettant un affichage :

-plot : cela permet de visualiser un graphe avec toutes arrêtes reliant les sommets (représente que peu d'intérêt donc il n'est pas possible de voir ce graphe en utilisant la représentation graphique de la classe GUI)

-plot_solution : permet l'affichage du chemin obtenu grâce à une méthode de résolution tel que le plus proche voisin. Le point de départ y est affiché en rouge contrairement aux autres sommets en bleu. Les graphes de matrices non symétriques sont affichés avec la direction du cycle. L'utilisateur à besoin de renseigner le chemin obtenu précédemment et le point de départ.

Quelques précisions sur les méthodes reprenant les heuristiques :

Les résultats des méthodes sont toujours path et path_lenght, path étant une liste de sommet et path_lenght la distance du trajet. Cette distance est toujours calculée de la même méthode, à partir du chemin obtenu on recherche le poids de l'arrête dans le graphe constitué du sommet à l'index x et du sommet à l'index x+1 pour tous les x allant de 0 à la taille du chemin moins 1.

-NN (plus proche voisin)

Il s'agit d'une boucle while qui permet de rajouter le sommet trouvé comme étant le plus proche voisin du sommet actuel dans le chemin à la condition que ce sommet ne se trouve pas déjà dans le chemin, sinon c'est la valeur suivante. Pour cela on effectue une boucle for qui parcours la liste des plus proches voisins en fonction du sommet actuel.

-NI (plus proche insertion)

Le principe reste le même que la méthode plus proche voisin mais cette fois-ci en regardant le plus proche voisin de tout les sommets constituants le chemin. La variable nearest_node permet donc de conserver la distance du sommet de plus proche distance entre tous les sommets déjà visités et current_node permet de conserver le sommet qui est actuellement le plus proche. Une fois tous les sommets visités, l'insertion du sommet s'effectue à l'index du plus petit delta C.

-Fletcher / Fletcher et Clarke

Ces deux méthodes utilisent la bibliothèque networkx pour trouver le plus court chemin, en effet il s'agit juste de rajouter des arrêtes selon un ordre de saving ou de coût et ensuite grâce à la bibliothèque networkx on peut vérifier s'il n'existe pas de fourche (sommet de degré trois) ou de cycle parasite (présence d'un cycle alors que le chemin n'est pas complet).

-ER / 2-OPT

Ces améliorations locales sont utilisées avec le chemin et la distance d'un résultat précédent. Les deux méthodes sont similaires, on utilise une liste identique au chemin pour faire des modifications dessus et si le coût total est réduit alors on garde l'amélioration. On n'utilise pas le calcul de delta C ici car pour des matrices non symétriques cela entraînera des erreurs sur la méthode 2-OPT.

Résultats :

Un fichier résultat au format text est fourni dans le dossier graph. Celui-ci comporte tous les résultats de toutes les méthodes des trois fichiers de matrices fourni dans le dossier. Il présente les chemins de plus courte distance avec le sommet de départ pour lequel la plus courte distance à était trouvé (il est cependant possible de trouver ce plus court chemin sur des sommet de départ plus grand mais pas l'inverse). On remarque pour les matrices br17 et bays29 que plus il y a d'améliorations plus on trouve un chemin identique que l'on peut présumer être le plus court possible. Ce chemin serait de coût 39 et atteignable des le sommet de départ 1 pour la matrice br17 et de 2020 aussi atteignable dès le sommet 1 pour bays29. Pour la matrice asymétrique ft53 et également de plus grande taille les résultats sont beaucoup moins flagrants. Le meilleur résultat est 6905 et obtenu à partir du sommet 10.

Quant aux meilleures méthodes il semble au travers de ce document que sans améliorations locales il s'agit de la plus proche insertion qui est la meilleur méthode (comme il n'y a que 3 matrices sur lesquels les tests ont été effectué ce résultat n'est pas certain), et qu'au travers de toutes les améliorations possibles ce soit la méthode Fletcher et Clarke qui est largement meilleure avec derrière la plus proche insertion (6 fois meilleurs contre 3 fois).