Projet de SD-Graphe:

Rapport final

AFFES Roua LO Amy

Plan :

1. Introduction :

Le problème de la tournée de véhicules avec capacité (CVRP) concerne la planification des livraisons pour minimiser les coûts de transport tout en respectant les contraintes de capacité des véhicules. Dans notre cas, chaque véhicule part du même dépôt, livre plusieurs clients, puis retourne au dépôt une fois sa tournée terminée. L'objectif principal est de minimiser la distance totale parcourue par l'ensemble des véhicules.

2. Précisions du cahier des charges

Description du problème de Tournée de Véhicule :

Le problème de tournée de véhicule concerne la planification efficace des tournées pour livrer des marchandises à un ensemble de clients tout en respectant les contraintes de capacité de chargement des véhicules.
Pour préciser les exigences et contraintes du cahier des charges, on va prendre en compte les points suivants :

- **Nombre de clients :** Le problème concerne la livraison d'un ensemble de clients, dont le nombre peut varier d'une instance à une autre.
- **Capacité des véhicules :** Les véhicules ont une capacité de chargement maximale fixe , exprimée en nombre d'unités de marchandises. Cette capacité doit être respectée lors de la planification des tournées de livraison.
- **Distances entre les clients :** Les distances entre chaque paire de clients et entre le dépôt et chaque client sont données.
- **Demandes de livraison :** Chaque client a une demande spécifique en termes d'unités de marchandises à livrer. Ces demandes doivent être prises en compte lors de la planification des tournées de véhicules.
- **Dépôt central :** Toutes les tournées de livraison commencent et se terminent au dépôt.
- **Objectifs de minimisation des coûts :** L'objectif principal est de minimiser le coût de la distribution en parcourant une distance totale minimale .

3. Modélisation et Méthode de résolution

Graphe h :

Définition :

- **Ensemble de sommets :** Les sommets de H représentent une étape de la distribution
- **Ensemble d'arcs :** Les arcs de H représentent les différentes tournées possibles des véhicules. Chaque arc relie. Chaque arc est pondéré par le coût (distance) associé à se déplacer d'un sommet à un autre et les clients visités dans la tournée.
- **Pondération des arcs :** La pondération des arcs représente le coût(distance) associé à à la tournée en tenant compte des contraintes de capacité des véhicules.

Caractéristiques :

Le graphe H est un graphe orienté, où les arcs ont une direction spécifique indiquant le sens de la tournée. Il est également un graphe pondéré, où chaque arc a un poids associé représentant le coût de déplacement et les clients inclus dans la tournée.

La méthode de résolution qu'on va utiliser dans le projet c'est la méthode route-first/cluster second :

- **Explication de la Création du tour géant:**

  Cet algorithme construit une tournée en partant d'un client initial et en ajoutant progressivement le client le plus proche non encore visité. Il commence par initialiser les distances minimales et marque les colonnes du dépôt et du client initial comme traitées. Ensuite, pour chaque étape, il cherche la distance minimale non encore traitée dans la ligne courante et met à jour l'indice du client le plus proche, qu'il ajoute à la tournée. Une fois qu'un client est ajouté à la tournée, la colonne correspondante est marquée comme traitée pour éviter de revisiter ce client. Ce processus se répète jusqu'à ce que tous les clients soient visités, en mettant à jour l'indice de la nouvelle ligne à traiter et en recherchant la prochaine valeur non marquée pour continuer la construction de la tournée.

- **Explication de l'algorithme du plus court chemin :**

  Algorithme de Bellman-Ford

  Pour trouver le plus court chemin entre un sommet source et tous les autres sommets d'un graphe pondéré, on utilise l'algorithme de Bellman-Ford car le graphe H n'a pas de poids négatif et les sommets sont déjà triés par niveau.

  Donc, la complexité de l'algorithme de Bellman-Ford est de O(n+m), où 'n' est le nombre de sommets et 'm' est le nombre d'arêtes dans le graphe car comme le graphe est triés par niveau, il n'a pas besoin de passer par toutes les itérations pour tous les sommets. Cela réduit la complexité de l'algorithme à O(V+E), ce qui est significativement plus rapide que O(V×E) qui est celle de l'algorithme de Dijkstra .

4. Choix et justification des différentes structures de données utilisées

4.1. Tableau comparatif de la représentation du graphe auxiliaire H

Voici un tableau comparatif des trois structures de données (SD) proposées pour représenter un graphe, avec leurs avantages et inconvénients. Ce tableau justifie également

le choix des listes chaînées et des tableaux (head, succ, pondération) en fonction des critères de flexibilité, efficacité, et complexité de manipulation.

V : nombre de sommets

E: nombre d'arcs

Critères	Matrice d'incidence + Tableau de pondération	Tableau head + Tableau succ + Tableau pondération	Tableau de pointeurs de listes chaînées
Complexité spatiale	O(V2) pour la matrice d'incidence, où V est le nombre de sommets.	O(V+E) pour les tableaux, où E est le nombre d'arêtes.	O(E) pour les listes chaînées, où E est le nombre d'arêtes.
Complexité temporelle	O(1) pour accéder aux arêtes et aux poids.	O(1) pour accéder aux arêtes et aux poids.	O(E) pour trouver un arc dans une liste.
Coût mémoire	Efficace pour les graphes denses, mais inefficace pour les graphes creux en raison de l'espace quadratique utilisé par la matrice.	Plus efficace que la matrice pour les graphes creux en raison de l'espace linéaire utilisé par les tableaux.	Efficace pour les graphes creux en raison de l'espace linéaire utilisé par les listes chaînées.
Opérations d'ajout/suppression	nécessite la réorganisation de la matrice.	nécessitent la réorganisation des tableaux.	ajout/suppression directe des maillons.
Facilité de manipulation	accès direct aux arcs et pondérations.	indexation simple des arcs et pondérations.	nécessite la gestion des pointeurs.

Accès aux successeurs	accès rapide via la matrice.	accès rapide via indexation des tableaux.	nécessite le parcours des listes.

Choix justifié des listes chaînées:

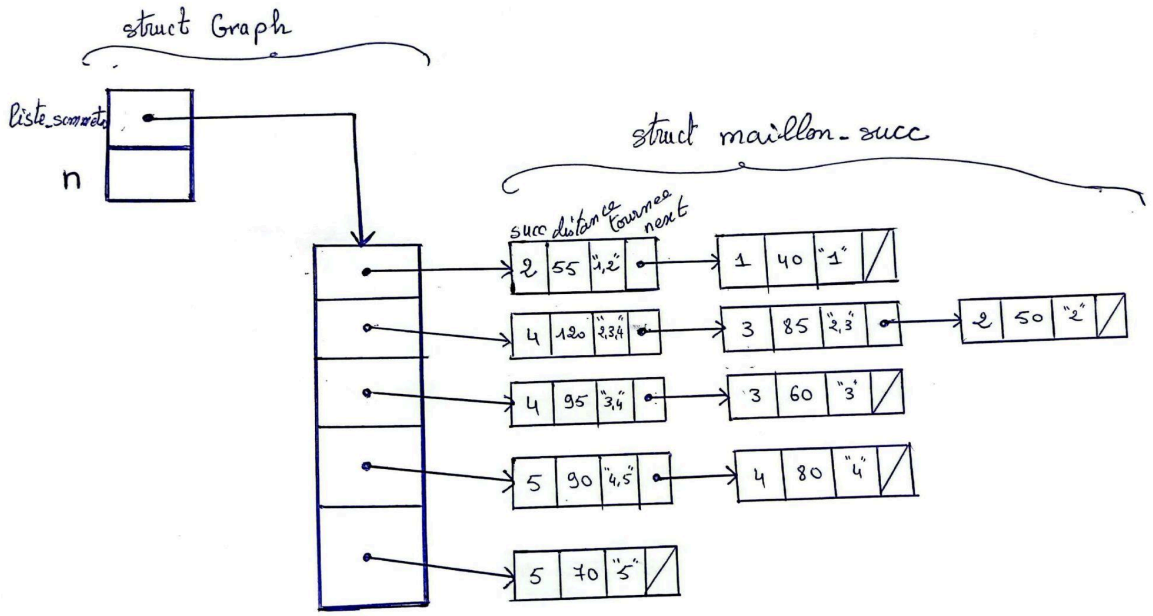
- **Flexibilité:** Les listes chaînées permettent une grande flexibilité pour modifier la structure du graphe, ajoutant ou supprimant des sommets et des arcs sans réorganisation majeure.
- **Espace mémoire:** Efficace pour les graphes , car seules les connexions existantes sont stockées, réduisant la consommation de mémoire.
- **Applications:** Particulièrement adaptées aux applications dynamiques où la structure du graphe change fréquemment.

Comme les listes chaînées sont efficaces pour accéder aux distances lorsque le degré des sommets est moins élevé (E << V²), alors c'est la structure de données idéale pour représenter le graphe H avec une complexité de O(E).

4.2 Schéma illustratif

Voici le schéma illustratif des structures de données retenues appliquées à l'exemple de base (fichier exemple.dat) :

Graphe auxiliaire H



Tour géant :

1	2	3	4	5

Matrice distance :

0	20	25	30	40	35
20	0	10	15	40	50
25	10	0	30	40	35
30	15	30	0	25	30
40	40	40	25	0	15
35	50	35	30	15	0

Tableau demande clients :

5	4	4	2	7

5. Algorithmes de résolution

5-1. Méthode pour créer le tour géant

Données :

- *MAT matrice d'entiers : matrice distance*
- *n : entier : nombre de clients*
- *ID :entier : numéro du client avec lequel le tour commence*

Résultat :

- *T : tableau d'entiers : Tableau du tour géant*

Locales :

- *min :entier : distance minimale dans une ligne donnée*

- *min_j : entier : indice de la colonne de la distance minimale dans la ligne courante*
- *i : entier : indice de la ligne de la distance courante*
- *j : entier : indice de la colonne de la distance courante*
- *tot : entier : dimensions de la matrice M (matrice carrée)*
- *k : entier : indice du choix courant à faire pour l'insérer dans T*
- *p : entier : indice pour parcourir les lignes de la matrice M*
- *test : entier : indice pour parcourir les lignes de la matrice M*
- *M : matrice d'entiers : matrice distance*

Algorithme :

```
procédure tourgeant(n, mat, t, id)
   // dupliquer la matrice mat pour ne pas la modifier durant l'algorithme
   déclarer m[n + 1][n + 1]
   pour i de 0 à n faire
      pour j de 0 à n faire
         m[i][j] <- mat[i][j]
      fin pour
   fin pour

   // initialiser les variables
   min <- m[0][id]
   min_j <- id
   i <- id
   tot <- n + 1
   t[0] <- id

   // marquer les colonnes correspondantes comme traitées
   pour p de 0 à tot - 1 faire
      m[p][0] <- -1  // marquer la colonne correspondante au dépôt comme traitée
      m[p][id] <- -1  // marquer la colonne correspondants au client id  comme traitée
   fin pour

   // mettre à jour les indices et la distance minimale
   i <- min_j
   min <- m[i][0]
   // trouver le min du prochaine ligne
   test <- 0
   tant que min == -1 ou min == 0 faire
      test <- test + 1
      min <- m[i][test]
      min_j <- test
   fin tant que

  // recherche de la distance minimale dans la ligne courante
   pour k de 1 à tot - 2 faire
      pour j de 0 à tot - 1 faire
         si i ≠ j et m[i][j] ≠ -1 et m[i][j] < min alors
            min <- m[i][j]
            min_j <- j
         fin si
```

```
        fin pour
        t[k] <- min_j  // insérer l'indice de colonne plus proche client dans le tableau du tour géant

        // marquer la colonne correspondante au plus proche client comme traitée
        pour p de 0 à tot - 1 faire
            m[p][min_j] <- -1
        fin pour

        // mettre à jour les indices et la distance minimale
        i <- min_j
       // trouver le nouveau min de la prochaine ligne
        min <- m[i][0]
        test <- 0
        tant que min == -1 ou min == 0 faire
            test <- test + 1
            min <- m[i][test]
            min_j <- test
        fin tant que
    fin pour

    // afficher le tour géant
    afficher "le tour géant est :"
    pour i de 0 à n - 1 faire
        afficher t[i], "|"
    fin pour
    afficher une nouvelle ligne
fin procédure
```

5-2. Procédure SPLIT

Données :

- *T:vecteur:d'entier: tour géant*
- *Q capacité des véhicules*
- *n nombre de clients*
- *M matrice distance*
- *q vecteur du nombre d'unités à livrer à chaque client*

Résultat :

- *pondération : tableau de structure pond : tableau qui contient la pondération des arcs*
- *head : vecteur d'entiers :tableau head de du graphe résultant*
- *succ : vecteur d'entiers : tableau succ du graph résultant*

Locales :

- *cost coût courant*
- *load chargement courant*
- *i, j indices utilisés pour parcourir les clients de T*
- *k indice pour remplir le tableau pondération*
- *h indice pour parcourir le tableau head*
- *code_ascii c'est le code ascii du caractère de l'indice du client*

```
procédure split(k, q, n, m, q, graph)
        // initialiser le tableau t pour contenir les sommets de la tournée géante
        déclarer t comme tableau de n+1 entier
        initialiser t[0] à 0
        pour i de 1 à n faire
        t[i] ← k[i-1]
        fin pour

        // déclarer les variables pour le coût et la charge
        déclarer cost comme réel
        déclarer load comme entier
        déclarer i, j, k comme entiers

        // parcourir chaque sommet i
        pour i de 1 à n faire
                j ← i
                load ← 0

        // boucle pour ajouter des sommets à la tournée tant que la charge est acceptable
                répéter
        // ajouter la demande du sommet t[j] à la charge totale
                        load ← load + q[t[j]]

                        // calculer le coût de la tournée
                        si i = j alors
                                // cas où la tournée commence et se termine au même sommet
                                cost ← 2 * m[0][t[i]]
                        sinon
                                // mise à jour du coût
                                cost ← cost - m[t[j-1]][0] + m[t[j-1]][t[j]] + m[t[j]][0]

                        fin si

                        // si la charge totale est inférieure ou égale à la capacité du véhicule
                        si load ≤ q alors
                                // construire la chaîne représentant la tournée
                                déclarer tournee comme chaîne
                                tournee ← ""  //  initialise la tournée à vide

                                // ajouter chaque sommet de la tournée à la chaîne
                                pour k de i à j faire
                                        déclarer str comme chaîne
                                        // convertir le numéro du sommet en chaîne
                                        str ← convertir_en_chaine(t[k])
                                        tournee ← concaténer(tournee, str)

                                        // ajouter une virgule entre les sommets
                                        si k < j alors
                                                tournee ← concaténer(tournee, ",")
                                        fin si
                                fin pour
```

```
                    // créer un nouvel arc (maillon) avec le coût et la tournée
                    déclarer new comme nouveau maillon de successeur
                    new ← créer_succ(j, cost, tournee)

                    // ajouter l'arc à la liste des successeurs du sommet i-1
                    new.next ← graph.liste_sommets[i-1]
                    graph.liste_sommets[i-1] ← new
                fin si

                // incrémenter j pour ajouter le prochain sommet à la tournée
                j ← j + 1
            jusqu'à (j > n) ou (load ≥ q) // arrêter si j dépasse n ou si la charge dépasse q
        fin pour
fin procédure
```

5-3. Plus court chemin

Données :

- **graph:** Le graphe sur lequel l'algorithme est exécuté.
- **r:** Le sommet source à partir duquel l'algorithme commence à calculer les distances les plus courtes.

Résultats :

- **pi:** Un tableau contenant les distances les plus courtes de la source r à chaque sommet du graphe.
- **père:** Un tableau contenant les prédécesseurs de chaque sommet sur le chemin le plus court de la source r.
- **Tournée**: Un tableau contenant les informations sur la tournée optimale pour chaque sommet.

Variables locales :

- **n:** Le nombre total de sommets dans le graphe.
- **arc:** Variable temporaire utilisée pour parcourir les listes d'adjacence et accéder aux arcs.
- **x, y**: Indices des sommets utilisés dans les boucles de l'algorithme.
- **i:** Variable de boucle utilisée pour itérer à travers les sommets du graphe.

```
Procédure BellmanFord(graph: Graphe, r: Entier)
    n =  Nombre de sommets dans le graphe
    Déclarer pi[n + 1] comme tableau de réels
    Déclarer pere[n + 1] comme tableau d'entiers
    Déclarer tournee[n + 1] comme tableau de chaînes de caractères

    // Initialiser pi à + infini et Père à 0
    Pour chaque sommet i de 0 à n faire
```

```
            pi[i]  <-  +infini
            pere[i]  <-  0
            tournee[i]  <- NULL
        Fin Pour

        // Définir pi[r] à 0 et Père[r] à r
        pi[r]  <-  0
        pere[r]  <-  r

        // Pour chaque successeur de r
        Pour chaque arc dans la liste d'adjacence de r faire
            succ<- arc<-succ
            pi[arc.succ] <-  arc<-distance
            pere[arc.succ] <- r
            tournee[arc.succ] <-  arc<-tournee
            arc<- arc<-next
        Fin Pour

        // Boucle principale pour l'algorithme de Bellman-Ford
        Pour i de 1 à n faire
            Pour chaque sommet x de 0 à n faire
                Pour chaque arc dans la liste d'adjacence de x faire
                    y <- arc.succ
                    Si pi[x] + arc.distance < pi[y] alors
                        pi[y]  <-  pi[x] + arc.distance
                        pere[y]  <-  x
                        tournee[y]  <-  arc.tournee
                    Fin Si
                Fin Pour
            Fin Pour
        Fin Pour
        affichage_bellman(n, tournee, pi, pere)
        affichage_tournee_optimale(n, r, tournee, pere)
Fin Procédure
```

6. Procédure globale d'enchaînement

```
Déclarer nb_clients comme entier
Déclarer capacite_vehicule comme entier

// Lire le nombre de clients
Si lire(nb_clients) échoue alors
   Afficher "Erreur lecture nb_clients"
Fin Si
```

```
// Lire la capacité du véhicule
Si lire(capacite_vehicule) échoue alors
   Afficher "Erreur lecture capacite_vehicule"
Fin Si

Déclarer demande_client comme tableau de nb_clients entiers
demande_client[0] <- 0

// Lire les demandes des clients
Pour i de 1 à nb_clients faire
   Si lire(demande_client[i]) échoue alors
      Afficher "Erreur lecture demande_client"
   Fin Si
Fin Pour

Déclarer mat_distance comme tableau de nb_clients + 1 par nb_clients + 1 réels

// Lire la matrice des distances
Pour i de 0 à nb_clients faire
   Pour j de 0 à nb_clients faire
      Si lire(mat_distance[i][j]) échoue alors
         Afficher "Erreur lecture mat_distance"
      Fin Si
   Fin Pour
Fin Pour

Déclarer Tour comme tableau de nb_clients entiers
Déclarer premier_client comme entier

//choisir le client avec lequel on commence le tour géant
premier_client <- 1

// Calculer le tour géant
tourGeant ( nb_clients, mat_distance, Tour, premier_client)

// // Construire le graphe auxiliaire
Déclarer graph comme Graph
graph <- créerGraphe(nb_clients)
procedureSplit (Tour, capacite_vehicule, nb_clients, mat_distance, demande_client,
  graph)

// Appliquer l'algorithme de Bellman-Ford
bellmanFord (graph, 0)

// Libérer la mémoire
liberation_memoire (nb_clients, graph, Tour)
```

7. Mode d’emploi

1- Compiler le projet :commande : make
2- Exécuter le projet : commande : make run FILE=nom_du_fichier.dat

exemple :
make
make run FILE=cvrp_100_1_c.dat

Dans le code , l’algorithme commence par défaut par le client 1 , si l’utilisateur veut commencer par un autre client il suffit de changer la valeur affectée à la variable **premier_client** dans la procédure globale d'enchaînement puis compiler et exécuter le projet

8. Description des exemples traités et résultats obtenus

Tous les exemples des fichiers tests donnés ont été traités avec succès :

- exemple.dat
- cvrp_100_1_r.dat
- cvrp_100_1_det.dat
- cvrp_100_1_c.dat

A titre d’exemple pour le fichier **exemple.dat :**

L'algorithme commence par calculer un tour géant, qui est une tournée passant par tous les clients une seule fois de manière à minimiser les distances parcourues. Le résultat est le suivant :

le tour géant est : 1|2|3|4|5|

Ce tour géant représente l'ordre dans lequel les clients sont visités pour former une tournée initiale avant toute optimisation.

- **Listes d'adjacence du graphe H**

Ensuite, l'algorithme construit le graphe auxiliaire H en utilisant les données fournies. Ce graphe est représenté par des listes d'adjacence, où chaque sommet (représentant un client ou le dépôt) est relié à ses successeurs possibles selon la capacité du véhicule et les distances minimales. Un exemple d'une des listes d'adjacence obtenues est :

```
Sommet 4 : -> (Succ: 5, Dist: 70.00, Tournee: 5)
```

Cela signifie que du sommet 4, on peut atteindre le sommet 5 avec une distance de 70.00, et cette transition représente la tournée du client 5.

- **Résultats de l'algorithme de Bellman-Ford**

L'algorithme de Bellman-Ford est exécuté sur le graphe H pour déterminer les distances minimales du sommet source (le dépôt) vers tous les autres sommets, ainsi que leurs prédécesseurs. Les résultats obtenus sont présentés sous forme de tableau :

Sommet	Distance depuis la source	Prédécesseur	Tournée
0	0.00	0	c'est la racine
1	40.00	0	1
2	55.00	0	1,2
3	115.00	2	3
4	150.00	2	3,4
5	205.00	3	4,5

Sommet : Le numéro du sommet .

Distance from Source : La distance minimale depuis le dépôt jusqu'à ce sommet.

Predecessor : Le prédécesseur de ce sommet dans le chemin le plus court trouvé par l'algorithme.

Tournée : La tournée associée à ce chemin.

- **Coût total**

```
le coût total est 205.000000
```

Ce coût représente la somme des distances minimales qu'effectuent les véhicules V1, V2 et V3 grâce à l'algorithme de Bellman.

- **Tournée optimale**

En utilisant les informations de prédécesseurs et de tournées obtenues, l'algorithme extrait et affiche la tournée optimale comme suit :

```
Tournée optimale : {(1,2)(3)(4,5)}
```

Cette tournée optimale représente les groupes de clients servis ensemble en minimisant la distance totale parcourue :

- (1,2) : Les clients 1 et 2 sont servis ensemble.

- (3) : Le client 3 est servi seul.
- (4,5) : Les clients 4 et 5 sont servis ensemble.

Les résultats montrent que l'algorithme de Bellman-Ford a été capable de trouver des distances minimales et les tournées optimales pour servir les clients en respectant les contraintes de capacité du véhicule. L'utilisation des listes chaînées pour représenter le graphe H permet une gestion efficace de l'espace mémoire et une flexibilité dans la manipulation des successeurs et des arcs.

9. Conclusion, Améliorations possibles

En conclusion, tout le code a été correctement implémenté et les tests ont été exécutés avec succès. Chaque composant de l'algorithme, y compris la lecture des données, la construction du tour géant, la création du graphe auxiliaire, et l'application de l'algorithme de Bellman-Ford, a fonctionné comme prévu, produisant les résultats attendus.

Cependant, des améliorations potentielles pourraient être apportées. Pour commencer, nous aurions pu utiliser un tableau de marquage dans la procédure de construction du tour géant au lieu de dupliquer la matrice des distances et de marquer les colonnes traitées avec des valeurs de -1. Cette modification rendrait le code plus clair et plus efficace en termes de mémoire.

De plus, il aurait été préférable de lire les fichiers de test directement plutôt que de les passer en entrée, ce qui aurait offert une gestion plus pratique des fichiers de test et facilité le changement de la valeur du premier client.

10. Bilan personnel

Amy LO
« Ce projet a constitué une opportunité majeure pour améliorer mes compétences en structures de données et en théorie des graphes, ainsi qu'en programmation en langage C. Cette expérience a été à la fois enrichissante et formatrice, démontrant une solide compréhension des concepts théoriques et des compétences pratiques acquises au cours de cette période. Travailler sur ce projet, que ce soit en autonomie ou en collaboration, a mis en évidence l'importance de la rigueur, de la communication et de l'entraide pour la réussite de nos objectifs.

Par ailleurs, ce projet m'a permis d'apprendre non seulement de mon binôme, mais aussi de moi-même. Il a été l'occasion d'appliquer mes compétences en algorithmique pour résoudre des problèmes complexes liés à l'optimisation des tournées de véhicules et à la gestion des graphes. Je suis convaincue que ce travail sera extrêmement précieux pour mon parcours professionnel, me permettant de mieux valoriser mes compétences dans ce domaine. »

Roua AFFES
Ce projet a constitué une expérience enrichissante qui m'a permis de mobiliser et d'approfondir mes connaissances acquises en structures de données, résolution de problèmes et théorie des graphes. J'ai eu l'opportunité de les mettre en pratique face à un défi concret et complexe : The Capacitated Vehicle Routing Problem (CVRP).

En concevant et en implémentant des solutions algorithmiques, j'ai pu illustrer de manière concrète comment les connaissances théoriques acquises en formation peuvent se traduire en applications pratiques ayant un impact tangible sur les coûts de transport et l'efficacité logistique.

Au-delà des compétences techniques, ce projet m'a permis de développer mes aptitudes relationnelles et ma capacité à collaborer efficacement. Le travail en binôme a été l'occasion de partager des idées, de formuler des arguments pour défendre mes choix et d'être attentif aux points de vue de mon partenaire. J'ai ainsi appris à composer avec des personnalités et des styles de travail différents, à trouver des solutions consensuelles et à privilégier l'écoute active et le respect mutuel. Cette expérience a renforcé ma capacité à travailler en équipe et à m'intégrer dans un environnement professionnel collaboratif.

En somme , ce projet a été une expérience enrichissante à la fois sur le plan technique et relationnel. Il m'a permis de mobiliser mes connaissances théoriques, d'en acquérir de nouvelles et de développer mon sens du travail en équipe. Ces acquis seront indéniablement précieux pour mon développement professionnel futur.

11. Auto-Évaluation

O1	O2	O3	O4	O5	note(sur 12)
0.5	0.5	1	1	1	10