TELECOM Nancy

Projet PPII-2

Gehin Clément
Natanelic Romain
Renard Nicolas

Responsables du module :
Olivier Festor
Gerald Oster

Table des matières

1 Introduction au sujet

1.1 Contexte du projet

Ce projet à été réalisé dans le cadre du module PPII-2 et C/SD (Structure de Données) de la première année du cycle ingénieur sous statut apprenti de TELECOM Nancy.

L'objectif de ce projet est de permettre à un utilisateur un ensemble de fonctions qui l'aide dans le déploiement et le dimensionnement d'un réseau de recharge de véhicules adapté aux besoins.

1.2 Organisation du document

Dans le chapitre 2, nous ferons une présentation des différents algorithmes adaptés à la résolution du problème à travers un état de l'art.

Dans le chapitre 3, nous présenterons la conception et l'implémentation de notre solution. Nous détaillerons la conception de notre projet ainsi que sa structure

Dans le chapitre 4, nous présenterons les tests réalisés dans notre solution et les performances de cette dernière.

Enfin, dans le chapitre 5, nous présenterons les éléments et outils de gestion de projet que nous avons utilisés, puis un bilan de projet sera fait dans la la partie 6.

1.3 Principe général de notre solution

L'objectif de notre application est de permettre à un utilisateur possédant une voiture électrique de se rendre d'un point A à un point B. Notre travail est de penser un algorithme lui permettant de passer le moins de temps possible sur la route, mais également sur les bornes de recharges. Pour cela, plusieurs paramètres sont à prendre en compte :

- L'autonomie de la voiture, variant selon les modèles de 95 à 685 km.
- Le temps de charge du véhicule
- Les emplacements des bornes, leur nombre et leur puissance.

Sachant tous ces éléments, nous utilisons un algorithme capable de calculer le plus court chemin indiquant les bornes de recharge par lesquelles passer.

La deuxième phase du projet a pour but de créer une "simulation" permettant d'évaluer la charge du réseau de bornes électriques. Cette simulation permettra de gérer un ensemble d'usagers (leur nombre, parcours...) et va montrer le taux de charge des bornes en fonction du temps et voir s'il y'a des files d'attentes qui se créent.

Les tests pourront également être paramétrés selon un taux de charge sous lequel il ne faut pas descendre, un temps maximum pour recharger sur une borne...

2 État de l'art

2.1 Étude et choix de l'algorithme

Il nous était demandé de réalisé un état de l'art sur les différents algorithme de recherche de plus court chemin, qui nous a ensuite permis de choisir lequel implémenté dans notre solution. Nous en avons trouvé 5, tous plus ou moins pertinents :

- Dijkstra
  - Avantages : Garantit de trouver le plus court chemin entre un nœud source et tous les autres nœuds, peut être optimisé, et donc utiles pour l'analyse des performances et enfin avantage non négligeable, nous avons déjà implémenté cet algorithme lors de notre premier projet PPII.
- Bellman-Ford
  - Avantages : Trouve le plus court chemin entre un nœud source et tous les autres nœuds dans un graphe pondéré.
  - Inconvénients : Moins efficace que Dijkstra, donc dans ce cas autant reprendre Dijkstra directement.
- A*
  - Avantages : Utilise une heuristique pour guider la recherche vers la solution optimale, ce qui permet d'explorer plus efficacement l'espace des états. De plus il est plus efficace et rapide que Dijkstra.
  - Inconvénients : Son efficacité dépend de l'heuristique choisie, qui peut amener à des résultats sous-optimaux. Cet algorithme dépend grandement de l'heuristique, point sur lequel notre groupe n'est pas assez bon pour choisir cette solution.
- Floyd-Warshall
  - Avantages : Trouve les plus courts chemins entre toutes les paires de nœuds dans un graphe pondéré.
  - Inconvénients : Nécessite une matrice de distances pour toutes les paires de nœuds, donc utilisation d'un grand espace mémoire et pertes de performances possibles.
- Johnson
  - Avantages : Utilise l'algorithme de Bellman-Ford et l'algorithme de Dijkstra modifié.
  - Inconvénients : Moins efficace que Floyd-Warshall et requiert des opérations supplémentaires pour la reconstruction des chemins les plus courts, donc perte d'optimisation.

Après études de ces cinq solutions, nous avions décidé d'utiliser l'algorithme de Dijsktra, notamment, car nous l'avions déjà implémenté, et donc nous le connaissions bien. De plus, le changement de langage n'était pas difficile et nous aurait fait gagner du temps. Malheureusement après avoir testé cette solution, nous nous sommes rendus compte que l'algorithme de Dijkstra n'était pas assez performant étant donné notre nombre de points/stations. De plus, on a à faire à un graphe complet (chaque sommet est relié aux autres), et dans cette optique Dijsktra est encore moins performant.

C'est dans cette optique que nous nous sommes tournés vers une version alternative de l'algorithme A-star, nous permettant dans un premier temps de tracer autour de notre point de départ un rayon donné (équivalent à l'autonomie de la voiture). Puis dans la seconde étape, calculer la distance entre tous ces points relais et le point d'arrivée, afin de trouver le plus proche, et donc de le sélectionner.

2.2 Choix de la structure de données

En dehors de l'algorithmique, nous avons également étudié les possibilités pour la structure de données. En effet, nous avons voulu créé une structure de stations qui serait à la fois pertinente et simple, c'est-à-dire ne contenant pas énormément de données, afin de ne pas surcharger le projet.

Pour cela, nous avons étudié les 38 champs disponibles dans le CSV (ceux-ci sont répertoriés sur la figure 1). Nous avons finalement décidé de garder les champs :

1. Commune
2. Longitude
3. Latitude
4. Puissance nominale

```
champs[0] = nom_amenageur
champs[1] = siren_amenageur
champs[2] = contact_amenageur
champs[3] = nom_operateur
champs[4] = contact_operateur
champs[5] = telephone_operateur
champs[6] = nom_enseigne
champs[7] = id_station_itinerance
champs[8] = id_station_local
champs[9] = nom_station
champs[10] = implantation_station
champs[11] = adresse_station
champs[12] = code_insee_commune
champs[13] = coordonneesXY // Attention, tableau a 2 dimensions
champs[14] = nbre_pdc
champs[15] = id_pdc_itinerance
champs[16] = id_pdc_local
champs[17] = puissance_nominale
champs[18] = prise_type_ef
champs[19] = prise_type_2
champs[20] = prise_type_combo_ccs
champs[21] = prise_type_chademo
champs[22] = prise_type_autre
champs[23] = gratuit
champs[24] = paiement_acte
champs[25] = paiement_cb
champs[26] = paiement_autre
champs[27] = tarification
champs[28] = condition_acces
champs[29] = reservation
champs[30] = horaires
champs[31] = accessibilite_pmr
champs[32] = restriction_gabarit
champs[33] = station_deux_roues
champs[34] = raccordement
champs[35] = num_pdl
champs[36] = date_mise_en_service
champs[37] = observations
champs[38] = date_maj
```

FIGURE 1 – Schéma de la base de donnée

3 Conception et implémentation de l'application

3.1 Introduction

Pour la conception de notre solution, nous avons utilisé chacun notre propre environnement sur lequel nous étions le plus à l'aise. Certains ont utilisé Visual Studio Code et le compilateur Clang, tandis que certains ont préféré Inteellij IDEA et le compilateur gcc. Comme demandé dans le sujet, nous avons travaillé à partir du langage C, et avons pu traiter un jeu de données sous format CSV.

3.2 Jeux de données

3.2.1 Le fichier CSV pour les stations

Pour débuter ce projet, nous devions nous appuyer sur un jeu de données comportant l'ensemble des bornes de recharges électriques disponibles en France. Nous avons donc importer du site du gouvernement ces données qui se trouvaient sous un fichier CSV.

Comme vu durant l'état de l'art, ce fichier contenait plus de 35 champs, tous plus où moins utiles pour l'implémentation de notre solution. Étant donné que notre objectif principal est de récupérer les coordonnées de chaque station, ainsi que le temps qu'il faut pour recharger les véhicules, nous avons opté comme expliqué précédemment pour garder quatre champs, sur lesquels nous reviendrons par la suite.

Afin de procéder à la récupération des champs de toutes les stations, nous utilisons les fonctions nécessaires pour dans un premier temps ouvrir le fichier CSV. Une fois ce fichier ouvert, nous procédons à l'extraction des champs qui nous intéressent (champs 12, 13 et 17, comme nous l'indique plus haut la figure 1.) en les insérant dans notre structure de données. Notre CSV ne nous permet de retirer les champs que sous forme de chaînes de caractères. Il faut donc ramener ces chaînes sous les formes qui nous intéressent, c'est-à-dire des types dee données "double" pour la plupart.

3.2.2 Le fichier pour les types de véhicules

Dans un deuxième temps, nous avions accès à un second jeu de données qui nous donnait toutes les caractéristiques des voitures électriques à notre disposition. Lorsque nous arrivons sur l'interface de ce jeu de données (c.f. figure 2), nous avons toutes les informations nécessaires afin de créer nos simulations.

Nous avons alors accès aux performances de la voiture, celles de la batterie, l'autonomie des différents modèles (variant de 90 à 685 km)... Malheureusement nous n'avons pas eu le temps de pousser notre solution, mais ces données auraient permis d'offrir de grandes possibilités de simulations

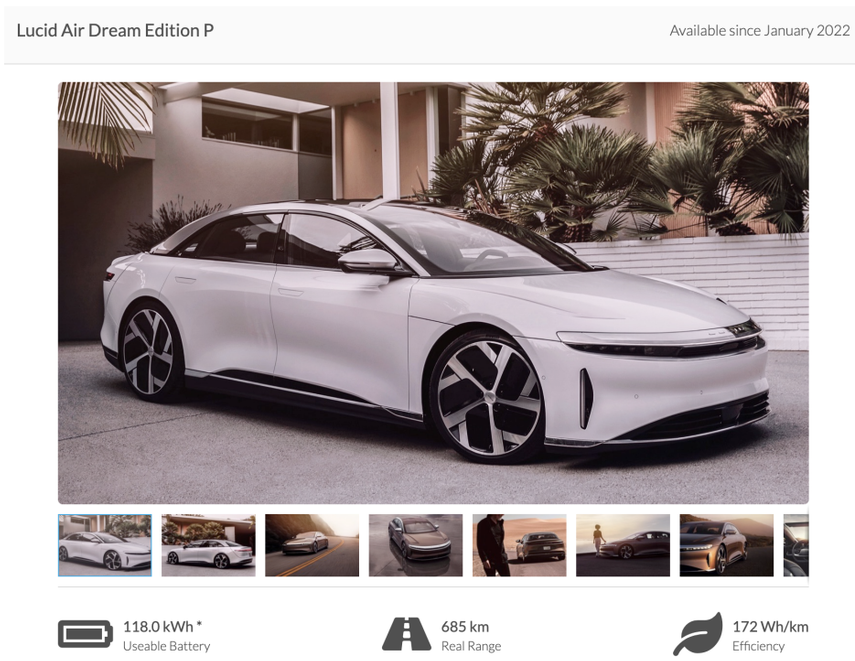

FIGURE 2 – Interface offrant les caractéristiques des différents véhicules

3.3 Structures de données

Les structures de données sont une composante essentielle de ce projet. Elles ont pour but de stocker de manière efficace une quantité importante d'informations (pratique puisque notre fichier CSV de stations contient plus de 55000 lignes qu'il faut stocker !).

3.3.1 La structure Station

Dans notre situation, comme évoqué plus tôt dans ce rapport, nous avons opté pour une structure telle que celle-ci :

```
typedef struct station_t
{
    bool visite;
    double lat;
    double lng;
    double puissanceNominale;
    char commune[50];
} Station;
```

L'avantage de l'utilisation d'une structure de données est de pouvoir stocker efficacement toutes les données nécessaires pour chaque station et de les regrouper par la suite dans une liste. Cette structure permet de conserver dans une liste de stations toutes les stations répertoriées par le fichier CSV. Cette liste une fois créée va nous permettre d'avancer vers la prochaine étape, qui est la simulation. Pour cela, nous utilisons une autre structure.
Pour comprendre un peu mieux l'utilisation de cette structure, voici e détails les champs qu'elle contient :

- visite : un booléen indiquant si la station a été visitée ou non.
- lat : la latitude géographique de la station.
- lng : la longitude géographique de la station.
- puissanceNominale : la puissance nominale de la station.
- commune : une chaîne de caractères représentant la commune où se trouve la station.

3.3.2 La structure Data

Cette structure à pour but d'initialiser la simulation, et est construite de la manière suivante :

```
typedef struct data_t
{
    int nbStation;
    Station** listeStation;
    int idStationSource;
    int idStationDest;
    int autonomie;
} data;
```

Cette structure permet donc de choisir un point de départ et un point d'arrivée parmi toutes les stations provenant du jeu de données, afin de réaliser l'algorithme permettant de trouver le parcours le plus optimisé. Voici en détails l'explication de chaque champ :

- nbStation : le nombre total de stations dans votre jeu de données.
- listeStation : un pointeur vers un tableau de pointeurs de structures Station, qui contient toutes les stations répertoriées.
- idStationSource : l'identifiant de la station de départ choisie pour l'algorithme de recherche du parcours optimal.
- idStationDest : l'identifiant de la station d'arrivée choisie pour l'algorithme de recherche du parcours optimal.
- autonomie : la valeur représentant l'autonomie (ou capacité) du véhicule utilisé dans la simulation.

Nous n'avons pas détaillé les structures suivantes, car nous ne les utilisons pas énormément et n'ont pas un aussi gros impact sur l'implémentation de la solution que les deux premières.

- **geographicDistance.h**, permettant de donner les coordonnées de deux stations.

— **ouvrir_csv.h**, qui nous permet de récupérer les chaîne de caractères dans le CSV.
— **fonction.h**, qui contient les fonctions permettant d'afficher les solutions aux simulations notamment.

Il était nécessaire de traiter de cette partie qui a fait l'objet d'un état de l'art et d'une grande réflexion de notre part, car cela permet d'avoir une solution plus efficace et performante. En effet, entre la manipulation, l'accès et le stockage des données, tout est rendu plus facile grâce à des structures de données optimales.

3.4 MakeKile

3.4.1 le fichier MakeFile

Pour expliquer le principe de MakeFile, c'est un fichier de configuration utilisé dans le but d'automatiser la compilation des programmes développés en langage C, afin qu'ils puissent par la suite être exécutés et utilisés.

En utilisant un Makefile, nous pouvons automatiser le processus de compilation, simplifier la gestion des projets et augmenter la rapidité globale du projet. En effet, au lieu de taper manuellement les commandes de compilation chaque fois que nous apportons des modifications à nos programmes, il est possible d'exécuter la commande "make" qui lira le Makefile et exécutera les règles nécessaires pour générer les cibles appropriées.

De plus l'utilisation de MakeFile permet l'utilisation de la commande "rm", afin de supprimer à chaque compilation les fichiers "fichiers.o", nécessaires uniquement lors de la compilation. Cela offre quelques avantages tels que :

- Libérer de l'espace disque inutile.
- Éviter les conflits de compilation.
- Reconstruction complète, évitant ainsi une quelconque erreur de compilation
- Simplification du processus de nettoyage, plutôt que de retirer les fichiers à la main

N.B. Notre fichier MakeFile est disponible en Annexe.

3.5 Algorithmique

Dans cette dernière section de la partie conception, nous avons choisi de parler de la partie algorithmique, "code pur" de notre programme.

Concernant la partie temps de traitement, optimisation... Nous y reviendrons dans le prochain chapitre qui est exclusivement consacré aux tests et performances.

3.5.1 Notre approche A*/A-Star

Comme nous l'avons évoqué auparavant dans l'état de l'art, nous nous étions penchés au départ sur l'implémentation de l'algorithme de Dijkstra. Mais état donné le nombre de stations, son implémentation était trop laborieuse et pas assez efficace.

Notre idée principale, qui s'est révélée par la suite être une implémentation de l'algorithme A-Star était simple :

1. Prendre une station de départ et une station d'arrivée
2. Autour de la station de départ, nous traçons un cercle ayant pour rayon l'autonomie du véhicule choisi. À l'intérieur de ce cercle se trouve (normalement) une ou plusieurs stations de recharge
3. Dans le cas où il y'aurait plusieurs stations, nous calculons la distance entre les stations de recharge et le point d'arrivée et cela pour chaque station
4. L'algorithme décide de retenir la station la plus proche du point d'arrivée.
5. On recommence jusqu'à pouvoir atteindre le point d'arrivée.

Cet algorithme de recherche de chemin nous permet de trouver le parcours optimal entre deux points. L'algorithme peut générer un itinéraire complet, comprenant les étapes de conduite et les arrêts de recharge, pour atteindre le point d'arrivée de manière efficace.

4 Tests et performances

4.1 Objectifs

Le but de cette partie est de nous faire réfléchir sur les performances de notre programme, et l'optimisation des fonctionnalités et algorithmes. Nous avons pu penser aux différentes manières d'améliorer notre solution, en temps et en complexité (les deux étant liés). Malheureusement, toutes nos idées n'ont pas eu la possibilité d'être intégrer, par manque de temps.

4.2 Tests

Afin d'évaluer et potentiellement améliorer les performances et la précision de notre programme, nous avons pu :

- Choisir l'algorithme de traitement le mieux optimisé face à notre besoin (c.f. État de l'art et conception)
- Améliorer le temps d'exécution en regroupant les données du fichier CSV en double.
- La "sensibilité" aux paramètres : C'est-à-dire étudier l'impact des différents paramètres tels que la taille du cercle d'autonomie du véhicule ou le choix de la station de recharge initiale, qui affectent vraisemblablement la qualité des résultats et le temps de calcul.

4.3 Complexité

4.3.1 Complexité temporelle

Afin d'améliorer notre algorithme, nous pouvons également nous appuyer sur la complexité de notre algorithme A-Star. Pour cela, nous avons pu mesurer la complexité temporelle de notre algorithme.

Nous utilisons un algorithme A-star classique, qui peut donc être exprimé en complexité O(N log N), notamment avec l'utilisation d'une structure de données avec une file de priorité (c'est-à-dire qu'on sélectionne la station la plus proche du point d'arrivée).

4.4 Performances

Les différences tests que nous avons pu réaliser, ainsi que le calcul de la complexité de notre algorithme nous donne une bonne indicationn sur les performances de notre solution.

D'un point de vue critique, nous pouvons admettre que l'utilisation du langage C nous permet d'avoir de grandes performances (lecture d'un fichier CSV de 55000 lignes en quelques secondes).

De plus, notre code est assez flexible, puisque n'importe quelle taille respectant le format (= les champs) que celui que nous avons utilisé, peut à son tour être utilisé, ce qui offre une bonne viabilité à notre projet.

5 Gestion de projet

5.1 Equipe de projet

Pour ce projet, notre équipe se compose de trois étudiants en filière par alternance :

— Clément Gehin
— Romain Natanelic
— Nicolas Renard

Notre groupe se réunissait généralement une fois par semaine en présentiel pour discuter de l'avancement du projet, de nouvelles idées la concernant ainsi que de la répartition du travail à réaliser. Durant tout le projet, nous discutions quotidiennement entre les cours ainsi que sur Discord, et travaillions parfois en groupe sur les salons vocaux du réseau social.

Toute la partie Gestion de Projet a été réalisé sur l'application Notion (planification, comptes-rendus de réunions...). Toute la programmation a été réalisée et partagée sur le Gitlab de l'école. La rédaction du rapport a, quant à elle, été réalisée sur Leaf.

5.2 Analyse du projet

5.2.1 Définition des objectifs

Chaque tâche a été attribué à chaque membre du groupe à l'aide de la méthode SMART :

	Critère	Indicateur
S	Spécifique	L'objectif est défini clairement. (c.f. Sujet du projet en Annnexe)
M	Mesurable	L'objectif est mesurable, par des indicateurs chiffrés ou livrables. (Ici, nous choisissons deux stations, une de départ et une d'arrivée, et le programme nous donne un parcours à réaliser)
A	Atteignable	L'objectif doit être motivant sans être décourageant et doit apporter un plus par rapport au lancement du projet. (Pour cela, nous avons simplement suivi le sujet qui nous était donné sous la forme AGILE, en nous lançant en premier lieu dans la phase 1, puis seulement dans la phase 2)
R	Réaliste	L'objectif doit être réaliste au regard des compétences et de l'investissement de l'équipe du projet. (Nous avons dû composer en fonction de nos emplois du temps, de nos compétences technique et de notre nombre)
T	Temporellement défini	L'objectif doit être inscrit dans le temps, avec une date de fin et des jalons. (Utilisation du Gantt)

5.3 Organisation du projet

Le projet s'est déroulé de mi-mars à fin mai 2023, nous laissant un peu plus de deux mois. Nous avons réparti le travail en plusieurs étapes pour faciliter l'organisation et la réalisation de ce projet. (c.f. figure 3). Nous avons dès le départ fixé des dates à ces échéances, sachant que, comme dans beaucoup de projets, ces dates n'allaient pas être respectées, notamment à cause des imprévus et du travail en parallèle que nous avons eu. C'est dans cette optique que nous avons volontairement créé des jalons longs et réalistes pour la plupart des tâches, afin de ne pas être pris au dépourvu arrivé à échéance du projet.

Aa Nom	∑ Durée	Date	∑
Reflexion autour du sujet	12	30 mars 2023 → 11 avril 2023	0
Proposition et choix des stru	7	12 avril 2023 → 19 avril 2023	0
Lecture fichier CSV	6	20 avril 2023 → 26 avril 2023	0
Saisie CSV dans structure	6	27 avril 2023 → 3 mai 2023	0
Proposition et choix de l'algo	7	12 avril 2023 → 19 avril 2023	0
Ecriture premiere fonction al	17	20 avril 2023 → 7 mai 2023	0
Choix station départ et arrivé	4	17 mai 2023 → 21 mai 2023	0
Affichage résultat	0	22 mai 2023	0
Assemblage différentes fonc	11	8 mai 2023 → 19 mai 2023	0
Test	1	23 mai 2023 → 24 mai 2023	1
Ecriture rapport	24	29 avril 2023 → 23 mai 2023	1

+ Nouvelle page

FIGURE 3 – Description des tâches du groupe et de leurs durées respectives

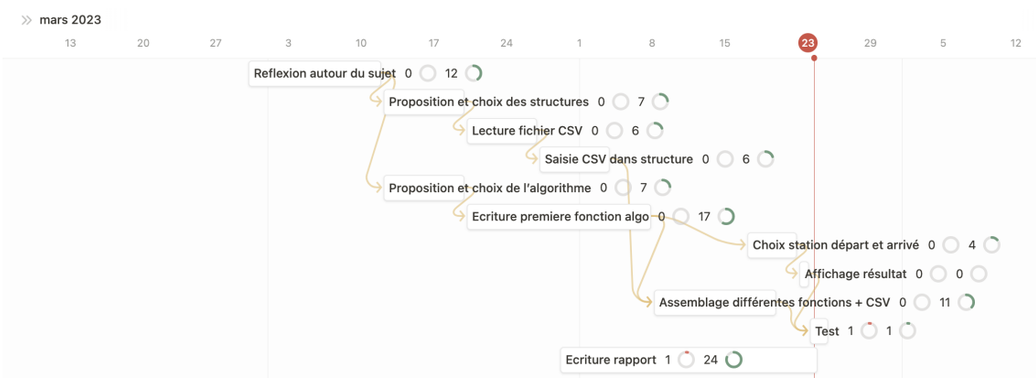

FIGURE 4 – Répartition de nos différentes tâches selon le calendrier

5.4 Outils de Travail

Afin de travailler tous ensemble sans se gêner, nous avons travaillé sur le Gitlab mis en place par l'école et avons séparé notre travail en branches, notamment la partie lecture du CSV. Une à deux fois par semaine, nous faisions des fusions sur la branche principale, qui opérait comme un "point de passage", afin de valider le travail déjà réalisé.

5.5 Comptes-rendu des réunions

5.5.1 24 mars 2023

Présent	Absent
Clément Nicolas Romain	

Objectifs

1. État initial du projet
2. Répartition des premières étapes
3. Planification

D'abord, pour réaliser le projet, on pense utiliser Dijkstra, qui est déjà bien clair dans nos esprits, et qui permettrait bien de parcourir les bornes comme les sommets d'un graphe. On va utiliser un fichier de données (CSV) et il va falloir le scraper, un pour les véhicules et un pour les bornes/villes. C'est à vol d'oiseau.

Problème d'autonomie

Planif : terminer mi-avril le projet de base, puis ajouter les fonctions au fur et à mesure (ex : interface graphique ?)

Prochaine réunion : Vendredi Prochain (31/03)

Objectifs d'ici là : trouver parser csv, algo djisktra en c et voir l'existant.

5.5.2 11 avril 2023

Présent	Absent
Clément Nicolas Romain	

Objectifs

1. Lecture CSV fonctionnelle - Clément
2. Structure d'une station - Nico
3. Algo de recherche - Romain
4. fonction de calcul de distance entre 2 coordonnées géo - Nico

Nous n'avons pas eu le temps de rééllement avancer depuis la dernière réunion. >De plus, c'est plus difficile pour nous de communiquer en période d'entreprises.

5.5.3 18 avril 2023

Présent	Absent
Clément Nicolas Romain	

Objectifs

1. MakeFile
2. Intialiser un tableau avec toute les structures dans le CSV
3. Continuer a réecrire l'algo dijkstra - Romain

Pour revenir sur le point principal : Intialiser un tableau avec toute les structures dans le CSV puis Initialisation (Tableau distance –> chaque ligne correspond au point i, les colonnes correspondent aux distances entre le point i et les autres points)

5.5.4 3 mai 2022

Présent	Absent
Clément Nicolas Romain	

Objectifs

1. MakeFile - Nicolas
2. Rendre propre le dépôt Git - Romainn
3. Prendre dans le csv les données importantes - Clément

Nous n'avons pas eu le temps de rééllement avancer depuis la dernière réunion du fait de l'entreprise

Nous avons rappeler les dates de rendu (25 Mai)

5.5.5 12 mai 2023

Présent	Absent
Clément Nicolas Romain	

Objectifs

1. Mettre la structure créée par Clément dans celle créée par Nicolas
2. Push Makefile
3. Get index utilisateur en entrée
4. Commencer le rapport

Retour sur les tâches défini dans la dernière réunio n qui ont été faites.

Il faut désormais avancer nous sommes dans la dernière ligne droite et tout arrive en même temps.

5.5.6 22 mai 2023

Présent	Absent
Clément Nicolas Romain	

Objectifs

1. Terminer le rapport
2. utiliser les structures Station venant du CSV
3. Terminer les dernières fonctions

Nous sommes dans le sprint final. Nous allons passer énormément de temps sur le projet, à nous d'être efficace. Clément s'occupe du rapport pendant que Romain et Nicolas finissent la partie programmation.

6 Bilan du projet

6.1 Bilan global du projet

Travail attendu	Le but du projet était de créer une simulation en langage C utilisant des structures de données visant à matérialiser la charge du réseau. Ce projet doit être présentable à des professionnels
Ce que nous avons fait	Difficultés à continuer le travail pendant la période d'examen et d'entreprise. Nous avons réussi la phase 1 mais n'avons pas réussi à aller bien plus loin par manque de temps. Cepeendant nous arrivons bien à lire le jeu de données composé des stations de recharges, et pouvns donc naviguer entre deux stations (départ et arrivée) en nous arrêtant aux meilleurs endroits, avec un parcours optimisé.

6.2 Bilan du projet par membre

6.2.1 Clément Gehin

Points positifs	J'ai pu consolidé mes (faibles) bases en structures de données et en C. Cela m'aide à comprendre mieux ce type de langage et son utilité
Points négatifs	Manque de temps et déçu de ne pas avoir mener à son terme le projet.
Expérience personnelle	Une bonne entente au sein du groupe, comme lors du premier semestre. Un projet comme celui-ci, unn peu dans l'urgence étant donnné les échéance m'a permis d'engranger de l'expérience. **Total** : 70h
Axes d'amélioration	Plus d'organisation et donc une meilleure gestio du temps.

6.2.2 Nicolas Renard

Points positifs	C'est toujours intéressant de realiser quelque chose à plusieurs, le groupe étant hétérogène en compétence, nous avons à mon sens réussis à trouver un équilibre dans le groupe. Les réunions et les points furent plus présent qu'au premier PPII ce qui est une bonne avancée.
Points négatifs	Déception sur le fait de ne pas être allé au bout de la phase 2.
Expérience personnelle	: N'utilisant que rarement des langages où on doit gérer précisément la mémoire, ce projet m'a permis de combler cette lacune . Le C étant un langage de niveau relativement complexe, cela m'a permis de voir comment realiser "From Scratch" des programmes relativement complexe, techniquement, c'est une expérience intéressante. **Total** : 70h
Axes d'amélioration	Améliorer notre découpage des tâches et mieux gérer les échéances dans des périodes de révision.

6.2.3 Romain Natanelic

Points positifs	Bonne implication de tout le groupe, nombreuses idées et reflexions de chacun Organisation assez bonne et partagé équitablement
Points négatifs	Langage que nous n'apprecions pas spécialement tout les 3 donc difficulté supplémentaire.
Expérience personnelle	Le projet m'a permis d'augmenter mes compétences en C qui étaient peu conséquentes. J'ai également pu participer a la direction et l'organisation du projet qui est un point que j'apprecie Environ XXX heure (**Total** : 70h
Axes d'amélioration	Améliorer la decomposition en sous taches et sous parties afin de ne pas se marcher dessus lors du developpement

Annexe

Sujet

Le sujet comprend deux étapes complémentaires.

Première étape

La première consiste à construire en langage "C" un programme qui permet :

- de proposer à tout usager souhaitant se rendre d'un point A à un point B du territoire, un parcours de charge de son véhicule électrique (identifier les stations qui lui permettent d'attendre sa destination). Chaque véhicule possède des caractéristiques différentes. Vous prendrez un modèle simple : chaque véhicule à une capacité et une consommation (les données sont extractibles de la Electric Vehicle Database). On supposera qu'il existe un chemin direct entre toutes les stations de recharge.
- d'enrichir la fonction précédente avec des paramètres supplémentaires, par exemple, ne jamais laisser le véhicule en dessous d'une charge de 30%, ou si votre modèle le permet, borner les temps de recharge (l'usager ne veut jamais rester plus de 20 minutes en charge)

Afin de construire les fonctions ci-dessus, vous identifierez les structures de données qui permettent de modéliser le problème et réaliserez un état de l'art des algorithmes adaptés à la résolution du problème.

Seconde étape

La seconde étape vise à proposer sur la base des structures précédentes, un mode "simulation" à destination des investisseurs et autorités, simulateur qui va évaluer la charge du réseau en fonction d'un ensemble d'usagers fictifs (ou pas) dont le parcours est connu. Concrètement, il vous est demandé de construire en langage "C" un module de simulation qui gère un ensemble d'usagers (leur nombre, leur parcours, et les caractéristiques de leurs véhicules sont un paramètre du simulateur) et qui, sur la base des bornes de recharges calculées par les fonctions de la première partie du projet, va à chaque étape (le temps étant discrétisé par pas de 10 minutes), calculer le taux de charge des bornes du territoire, identifier celles sur lesquelles des files d'attentes se créent et, si le temps le permet, proposer des chemins alternatifs aux véhicules pour éviter les files d'attentes.

Toujours pour des raisons de simplicité, on pourra supposer que tous les véhicules roulent à la même vitesse sur le territoire entre deux bornes.

Les plus téméraires des groupes, pourront réaliser une fonction de visualisation des parcours proposés et de la simulation.

Annexe

```
CC = clang
CFLAGS = -Wall -Wextra -pedantic -O0 -g3 -fsanitize=address

tout:
    $(CC) $(CFLAGS) -c selectionStation.c GeographicDistances.c ouvrir_csv.c fonctionDijkstra.c Main.c -I ../include
    $(CC) $(CFLAGS) selectionStation.o GeographicDistances.o ouvrir_csv.o fonctionDijkstra.o Main.o -o executable -I ../include

Main:
    $(CC) $(CFLAGS) -c Main.c -I ../include
    $(CC) $(CFLAGS) Main.o -o Main -I ../include

Dijkstra:
    $(CC) $(CFLAGS) -c fonctionDijkstra.c -I ../include
    $(CC) $(CFLAGS) fonctionDijkstra.o -o fonctionDijkstra -I ../include

Csv:
    ulimit -s unlimited
    $(CC) $(CFLAGS) -c ouvrir_csv.c -I ../include
    $(CC) $(CFLAGS) ouvrir_csv.o -o ouvrir_csv -I ../include

Station:
    $(CC) $(CFLAGS) -c selectionStation.c -I ../include
    $(CC) $(CFLAGS) selectionStation.o -o selectionStation -I ../include

GeoDistance:
    $(CC) $(CFLAGS) -c GeographicDistances.c -I ../include
    $(CC) $(CFLAGS) GeographicDistances.o -o GeographicDistances -I ../include

clean:
    rm -f *.o
    rm -f ./CSV_Ouverture/*.o
    rm -f ./CSV_Ouverture/ouvrir_csv
    rm -f ouvrir_csv
    rm -f ./CSV_Ouverture/Station.h.gch
    rm -f selectionStation
    rm -f fonctionDijkstra
    rm -f GeographicDistances
    rm -f executable
    rm -f Main
```