

Project Préing 2

Matias Vinkovic Ahmed METWALLY Timothée Kippelen
Cy Tech 2023/2024

Ce document vise à vous expliquer notre méthode de travail en groupe ainsi que les difficultés que nous avons rencontrées.

Répartition des tâches

La répartition des tâches au sein du groupe s'est faite en fonction des compétences et des intérêts de chaque membre, garantissant ainsi une répartition équilibrée et bénéfique du travail.

Dès nos premiers cours d'informatique, nous avons entamé notre collaboration, aussi modeste soit-elle. Notre parcours a débuté par une quête d'apprentissage et de familiarisation avec les concepts inhérents à notre domaine d'étude. Sous la houlette de Matias, reconnu pour son élégance et son esprit aiguisé, nous avons rapidement identifié le véritable défi : la conception d'une structure de données AVL efficace. Bien que la programmation et la compréhension des scripts Bash semblent accessibles, nous avons convenu que la maîtrise de la logique AVL serait la pierre angulaire de notre projet. Ainsi, chacun d'entre nous s'est attelé à la tâche, avec pour objectif commun de développer des fonctions AVL robustes, destinées à être intégrées ultérieurement dans notre projet global.

Timothée et Ahmed se sont concentrés principalement sur la conception et l'écriture des fonctions de rotation et d'insertion. Pendant ce temps, Matias a pris en charge une tâche cruciale : comprendre en profondeur gnuplot et l'ensemble des concepts associés à l'environnement Bash. Pour faciliter la communication et la coordination de nos efforts, nous avons opté pour Discord comme principal canal de communication. À travers des appels et des échanges réguliers sur la plateforme, nous partagions nos difficultés, nos avancées et nos objectifs respectifs. Une fois nos fonctions AVL conçues et prêtes à être utilisées, nous avons amorcé le travail sur l'ensemble des traitements, capitalisant sur nos efforts collectifs et notre expertise individuelle.

Traitements -d1, -d2 et -l

Même avec les traitements considérés comme les plus faciles, à savoir D1, D2 et L, les défis n'étaient pas aussi triviaux qu'ils le semblaient initialement, en particulier compte tenu de la contrainte de temps serré de 8 secondes. Pour aborder le traitement D1, nous avons opté pour une approche initiale consistant à trier et à extraire uniquement la colonne "Driver ID", puis à éliminer les doublons parmi les 6 millions de lignes de données. Nous avons ainsi commencé avec le code suivant :

```
head -300000 data.csv | cut -d';' -f1,6 | sort -t';' -k2 | cut -d';' -f2 | uniq -c
```

Cependant, nous avons rapidement rencontré des obstacles : le code initial prenait un temps d'exécution beaucoup trop long pour être viable. Face à cette impasse, nous avons sollicité l'aide de notre fidèle allié : Internet. Nos recherches nous ont menés à une révélation : l'utilisation des fonctions uniq ou de tri et de coupe s'avérait extrêmement chronophage, surtout avec un ensemble de données de 6 millions de lignes. Heureusement, une solution à nos problèmes s'est présentée : la commande awk. Cette commande, polyvalente et puissante, pouvait effectuer toutes les opérations nécessaires en une seule fois : trier, découper et supprimer les doublons, offrant ainsi une alternative efficace et rapide à nos besoins. Voici la commande que nous avons eu à la fin :

```
awk -F';' '!seen[$1,$6]++ { count[$6]++ } END { for (name in count) print count[name], name }' data.csv
```

Il ne nous restait plus qu'à trier la liste que nous avions et à perfectionner la commande pour finalement générer le graphe gnuplot. Étant donné que les traitements D2 et L suivaient une logique similaire à celle de D1, nous avons simplement réutilisé la commande awk deux fois. Deux membres du groupe se sont concentrés sur les traitements D2 et L, tandis que le troisième membre se focalisait sur la mise en place du graphe gnuplot.

Traitements -t et -s

Après avoir mené à bien les traitements initiaux plus simples, nous avons abordé les traitements S et T, qui exigeaient l'utilisation des fonctions AVL. Bien que nous ayons été chargés de développer ces fonctions avant même les traitements D1, D2, etc., nous avons malheureusement rencontré quelques problèmes avec les fonctions que nous avions écrites lors des tests supplémentaires.

Pour les traitements S et T, la principale difficulté résidait dans la compréhension du processus ainsi que dans la détermination des lignes à lire et des colonnes à conserver. De plus, coder la méthode la plus appropriée pour obtenir les résultats escomptés s'est avéré être un défi de taille.

Notre approche initiale visait à extraire les informations principales que nous souhaitions, telles que les distances minimales ou les villes les plus traversées, et de les stocker dans une structure AVL afin d'éviter les doublons. Étant donné la diversité des caractéristiques de chaque information, telles que l'ID de la route, la distance minimale, la distance maximale, etc., nous avons décidé de stocker ces autres caractéristiques dans une structure AVL distincte. Ainsi, nous nous sommes retrouvés avec une structure AVL imbriquée, où chaque AVL principal contenait des AVL secondaires.

Option -s : L'idée est plutôt simple : un premier avl récupère les données du data.csv, déjà cut en fonction de ce qu'il nous intéresse. Ce premier avl est trié en fonction des routeID. Si un routeID revient, dans la partie "else" de l'algorithme d'insertion de l'avl, au lieu de ne rien faire pour éviter les doublons, nous mettons à jour un compteur, la somme de distance, afin de nous permettre de calculer la moyenne/ le min-max etc... Ensuite, nous passons de cet AVL à une file statique, nous simplifiant la tâche, pour ensuite repasser dans une file dynamique, pour enfin passer à une autre avl, cette fois trié en fonction du max-min. Il ne nous reste plus qu'à réaliser un parcours infixé, et récupérer les 50 dernières valeurs.

Option -t : L'idée est en gros la même que pour l'option -s. Le problème principal était la contrainte indiquant qu'une ville n'est traversée qu'une seule fois par trajet. Pour ce faire, nous avons imbriqué dans chaque noeud du premier avl, trié par ordre alphabétique, un autre avl, ainsi que la taille de l'avl à l'intérieur, et d'un compteur pour voir combien de fois la ville du noeud en question a-t-elle été une ville de départ. Ainsi, à chaque fois qu'une ville récupérée du data.csv apparaît, nous insérons le routeID de cette dernière dans l'avl à l'intérieur du noeud de la ville. Nous évitons alors les doublons de cette manière. Pour ce qui est du compteur du nombre de fois où la ville a été une ville de départ, très simple, si le stepID correspondant à la ligne d'où provient la ville insérée était égal à 1, alors on augmente le compteur. Pour ce qui est de la taille de l'avl à l'intérieur, nous avions alors penser à effectuer un algorithme récursif nous donnant la taille de l'avl, mais cela serait beaucoup trop long. Nous avons alors tout simplement opté pour une solution plus simple : à chaque insertion d'un routeID dans l'avl à l'intérieur du noeud de l'avl principal, nous augmentons un compteur. Un problème est survenu : même quand le routeID existait déjà dans l'avl, et donc que l'insertion n'avait pas lieu, le compteur de l'avl principal ne le savait pas et augmentait quand même, nous avons donc créé un pointeur, donnant une certaine valeur permettant à l'avl principal de savoir si l'insertion a eu lieu ou non.

A savoir qu'il y a eu pour les options -s et -t une collaboration au niveau de la méthode de résolution envisagé avec le groupe de Florian CHAPIN du groupe MI6.

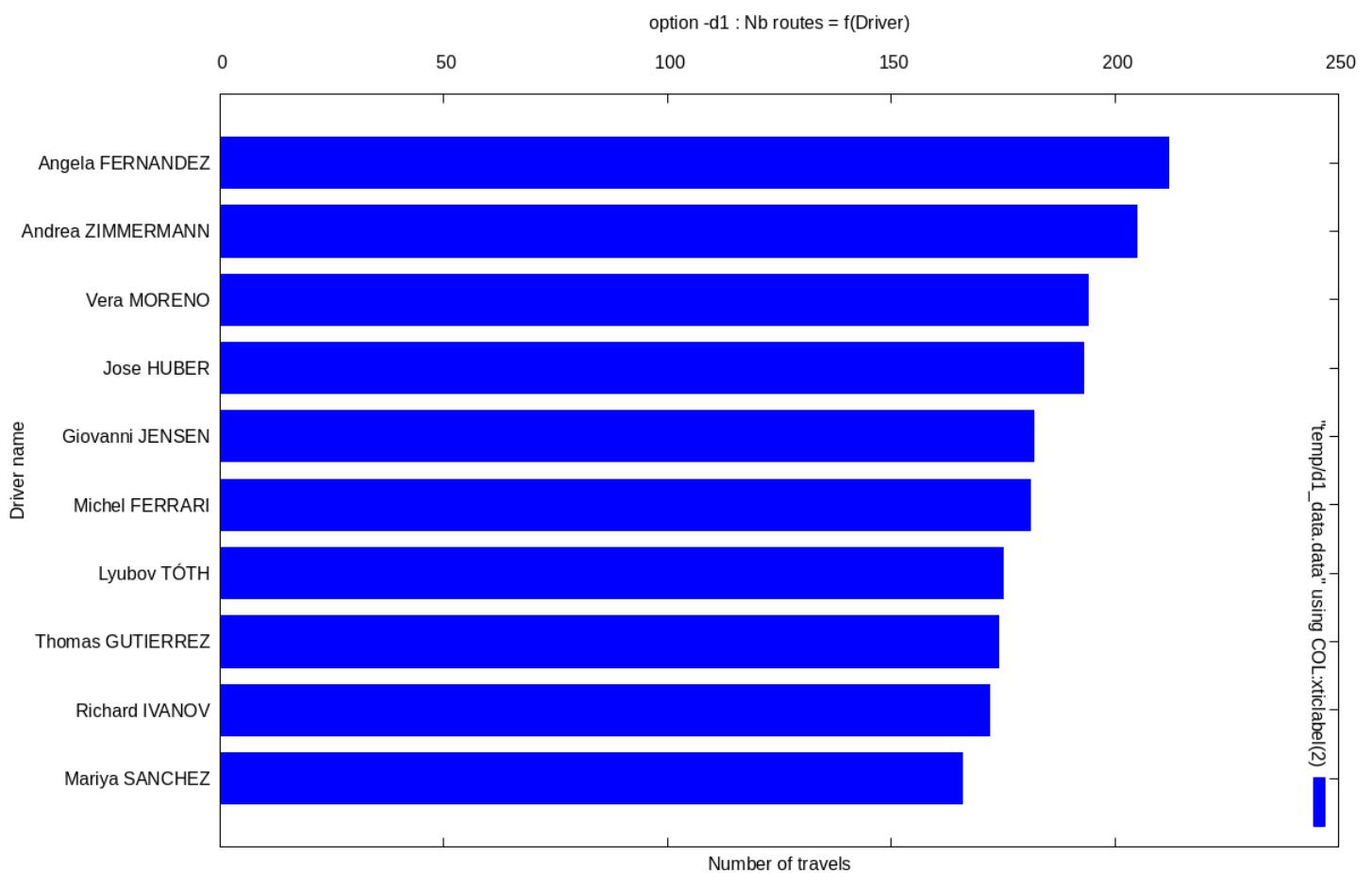
Merci pour votre attention

Voici notre contact en cas de problème(s):

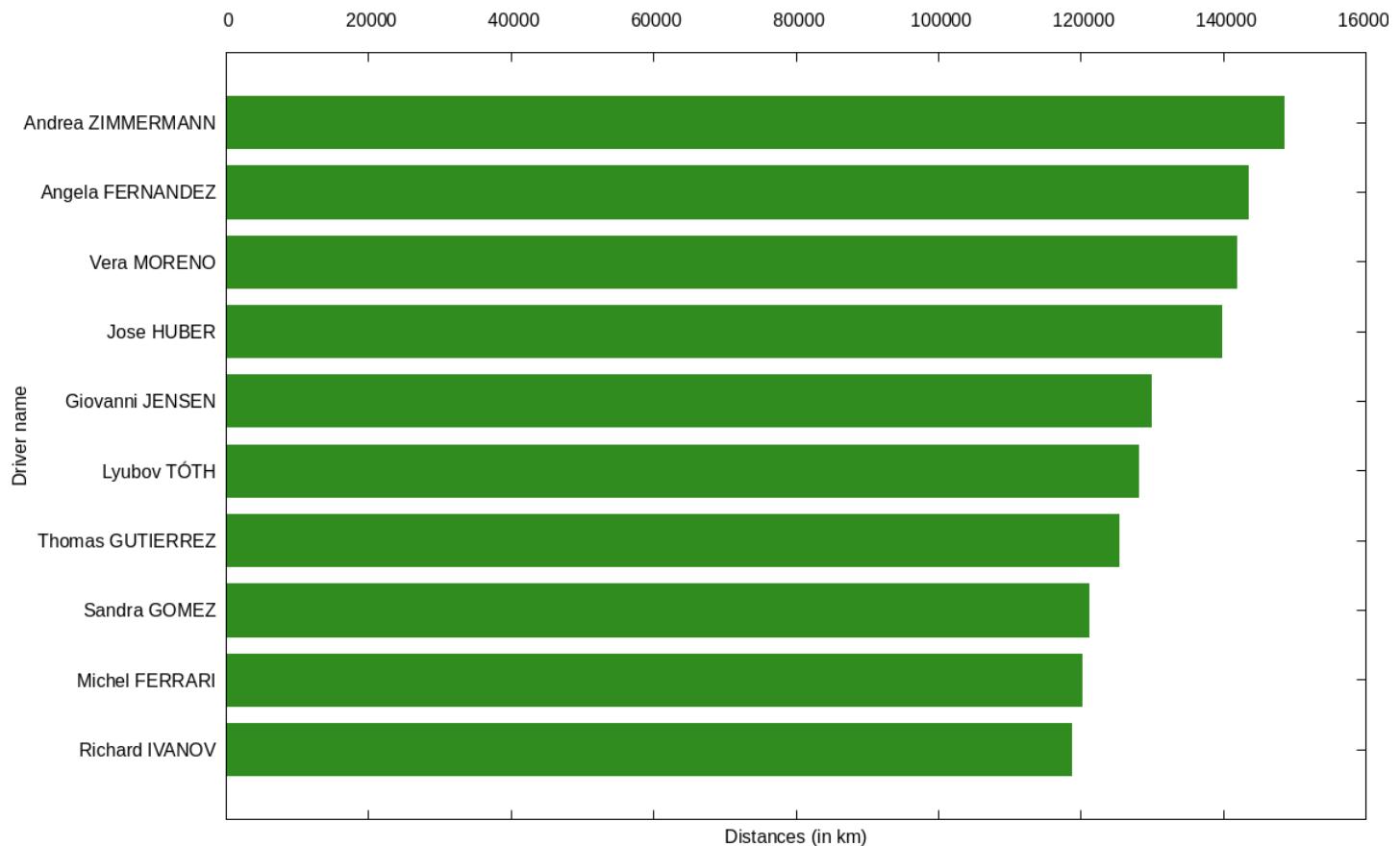
les3bogoss@gmail.com

01 30 37 93 40

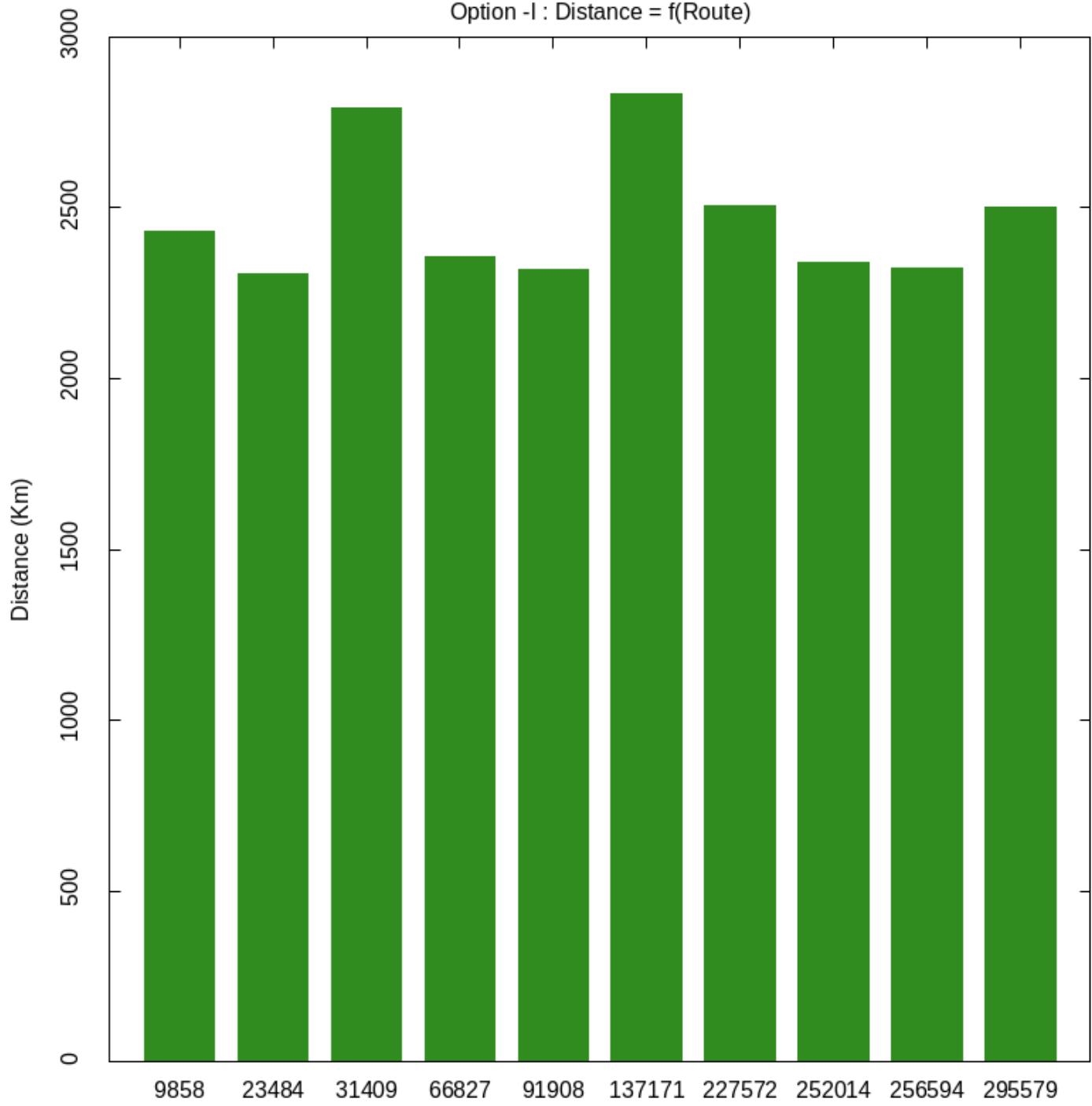
Voici les résultats obtenus



option -d2 : Distance = f(Driver)

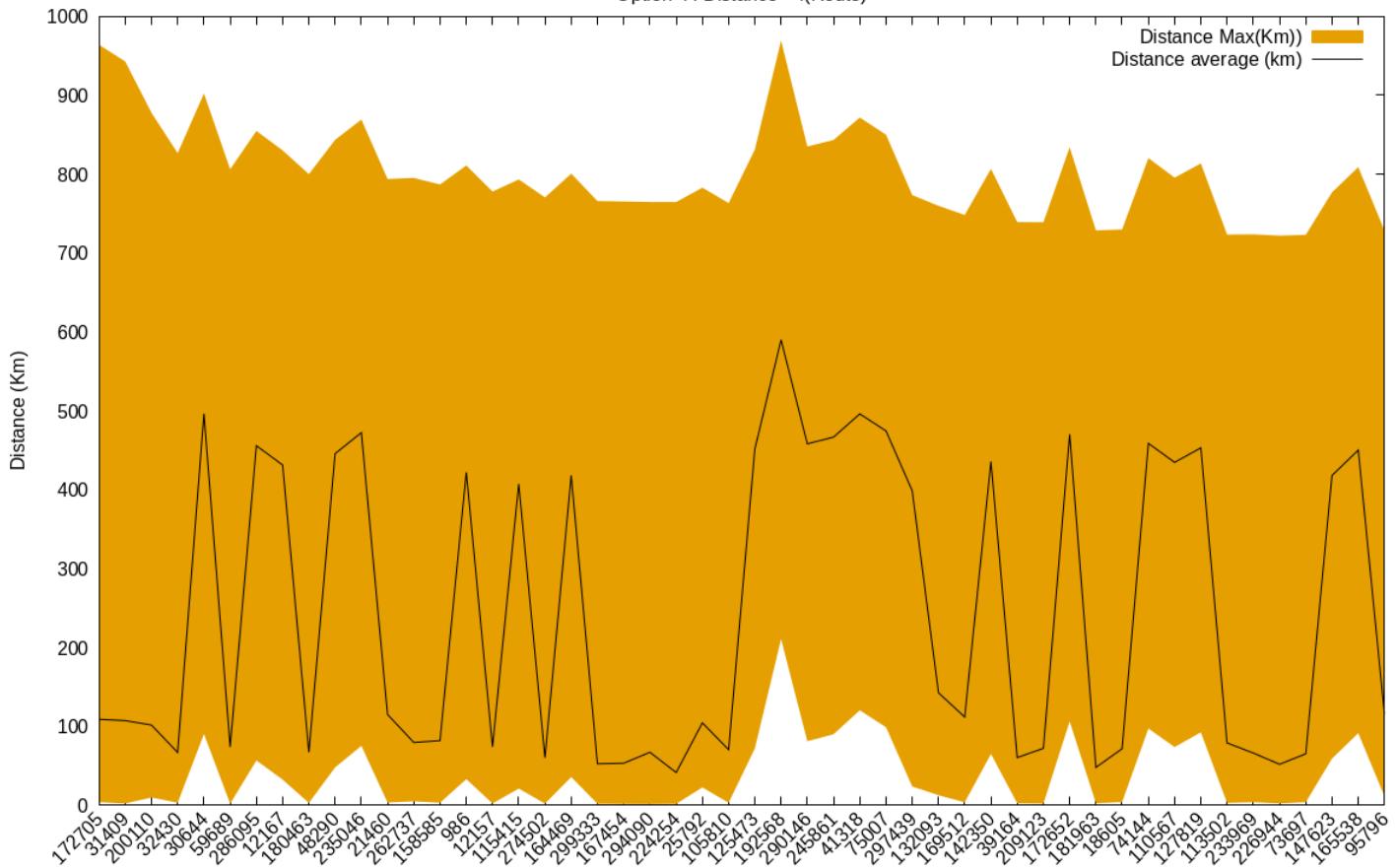


Option -I : Distance = f(Route)



Graphique Min-Max-Moyenne

Option -l : Distance = f(Route)



Option -t : Number routes = f(Towns)

