

Fig. Exemple de résultat d'un programme CVRP

Documentation Technique

CVRP Viewer

04.06.2020

Stefano Cirieco

**CFPT**

**IFA-P3B**

**V1.0**

Table des matières

[1 Introduction 3](#_Toc42186369)

[2 Rappel de l'énoncé 3](#_Toc42186370)

[2.1 Objectifs du projet 3](#_Toc42186371)

[2.2 Inventaire du matériel 3](#_Toc42186372)

[2.3 Inventaire des logiciels 3](#_Toc42186373)

[2.4 Travail à rendre 3](#_Toc42186374)

[2.5 Contraintes 3](#_Toc42186375)

[3 Analyse fonctionnelle 4](#_Toc42186376)

[3.1 Fonctionnement vu par l’utilisateur 4](#_Toc42186377)

[3.2 Description de l’utilisation et des fonctionnalités 4](#_Toc42186378)

[4 Analyse organique 4](#_Toc42186379)

[4.1 Classes 4](#_Toc42186380)

[4.1.1 Node 4](#_Toc42186381)

[4.1.2 DepotManager 4](#_Toc42186382)

[4.1.3 Truck 4](#_Toc42186383)

[4.1.4 Movement 4](#_Toc42186384)

[4.1.5 DataImporter 4](#_Toc42186385)

[4.2 Format des fichiers 5](#_Toc42186386)

[4.2.1 Fichier .vrp 5](#_Toc42186387)

[4.2.2 Fichier .dat 5](#_Toc42186388)

[4.3 Déroulement du programme 6](#_Toc42186389)

[5 Tests 6](#_Toc42186390)

[5.1 Conditions de tests 6](#_Toc42186391)

[5.2 Plan de test 6](#_Toc42186392)

[5.3 Rapport de tests 6](#_Toc42186393)

[6 Conclusion 6](#_Toc42186394)

[6.1 Difficultés rencontrées 6](#_Toc42186395)

[6.2 Améliorations possibles 6](#_Toc42186396)

[6.3 Bilan personnel 6](#_Toc42186397)

[7 Annexes 6](#_Toc42186398)

[7.1 Planning prévisionnel 6](#_Toc42186399)

[7.2 Planning réel 6](#_Toc42186400)

# Introduction

Ce document présente les différents aspects de la conception du projet CVRP Viewer. Ce projet a été développé pendant TPI (Travail Pratique Individuel) du 25 mai au 9 juin 2020.

CVRP Viewer est une application Windows Form qui visualise et résolue le « Capacitated Vehicle Routing Problem ». Qu’est-ce qu’un CVRP ? C’est un problème contenant un dépôt (point de départ), une liste de clients à différentes positions et des camions avec une capacitée maximale. Chaque client contient une demande de marchandise, la somme des demandes dans une tournée ne peut pas dépasser la capacitée maximale du camion. Pour résoudre ce problème il faut trouver les chemins les moins couteux, (sachant qu’une unité de distance est égale à un franc).

# Rappel de l'énoncé

## Objectifs du projet

L’objectif du projet est de pouvoir parser un fichier de type .vrp ou .dat, importer les données dans un objet puis trouver les chemins optimaux en affichant chaque étape sur la fenêtre et afficher le coût total des trajets optimaux.

## Inventaire du matériel

* Un ordinateur
* 2 écrans

## Inventaire des logiciels

* Windows 10
* Microsoft Word (Documentation Technique, Manuel Utilisateur & Rapport TPI)
* Microsoft Excel (Planning)
* C# avec Visual Studio 2019
* Gestion des versions avec GitHub

## Travail à rendre

Les documents à rendre aux experts :

* Documentation Technique
* Manuel Utilisateur
* Journal de bord

## Contraintes

Pendant que l’algorithme tourne il n’y a seulement un changement lorsqu’on trouve une solution moins couteuse que la précédente, cela nous permet de trouver la solution optimale local, puisqu’on commence avec des tournées aléatoires la solution optimale local diffère. Pour arriver à la meilleure solution possible il faut faire des changements qui augmente le coût pour ensuite baisser le coût avec un meilleur changement.

# Analyse fonctionnelle

## Fonctionnement vu par l’utilisateur

L’utilisateur, à l’ouverture du programme, est accueilli par une fenêtre avec des cercles vides pour représenter les clients, un cercle rempli pour le dépôt et des tournées déjà existant créées aléatoirement. Chaque client d’une tournée est relié par un trait avec son précédent et son prochain, sauf entre le dépôt et le premier et dernier client de la tournée. Il y aura aussi un bouton « Start » pour débuter l’optimisation.

## Description de l’utilisation et des fonctionnalités

Quand l’utilisateur appuiera sur le bouton « Start » l’optimisation commencera et il pourra voir un rafraichissement des tournées à chaque étape de cette procédure, jusqu’à la fin du programme. Le coût total de la meilleure solution trouvée sera affiché dans la barre de titre.

# Analyse organique

## Classes

### Node

Les nodes représentes les clients et le dépôt. Chaque node a une position X et Y et une position pour le dessiné qui vaut 4 fois l’originale pour améliorer la visibilité sur la fenêtre, une quantité de marchandise et un node désignant son suivant et un pour son précédent.

Ils ont une méthode permettant de calculer la distance entre eux et un autre node avec le théorème de Pythagore arrondi.

### DepotManager

Le DepotManager est composé d’un tableau de tous les clients et inclue une propriété qui retourne le dépôt, ainsi que la lise des camions.

Il y a une méthode pour ajouter et récupérer un client et une pour déclarer quelle node est le dépôt.

### Truck

Le Truck contient un Node Head qui représente le départ de la tournée à la même position que le dépôt. La propriété Next du Head pointe sur le premier client et lui sur le prochain, jusqu’à celui qui pointe sur le Head.

### Movement

Movement est une classe qui permet de stocker un mouvement de node entre deux tournées, et sauvegarde la réduction du coût des tournées. Cette classe implémente l’interface *IComparable* pour comparer le coût entre deux mouvements.

### DataImporter

La classe DataImporter s’occupe de lire un fichier .vrp ou .dat et de créer un DepotManager avec les données du fichier.

## Format des fichiers

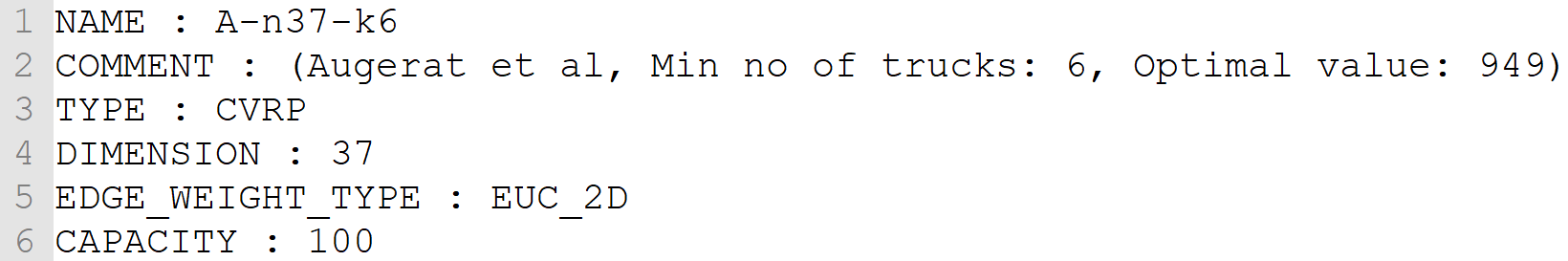
Dans ce projet il y a deux types de fichier (.vrp et .dat) à parser chaque un avec son propre format. Quand la méthode reçoit le fichier, il regarde son extension puis continue avec le parser dont il a besoin.

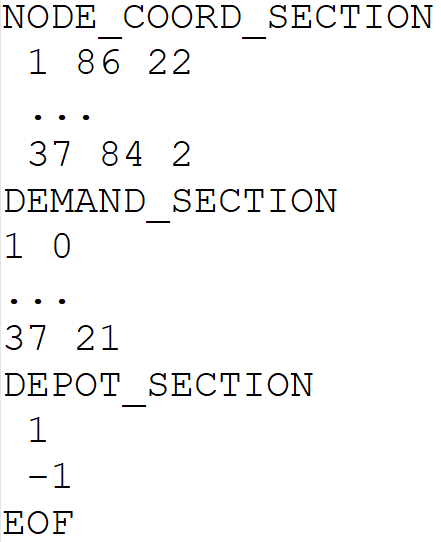
### Fichier .vrp

Les fichiers .vrp débutes avec les spécifications puis les données.

Les spécifications sont sous la forme *<keyword> : <value>*.

1. NAME : contient le nom du fichier.
2. COMMENT : contient l’auteur du problème ou de la solution, le nombre de camions utilisé et la valeur optimale de la solution.
3. TYPE : spécifie le type de donnée (dans mon cas cela sera toujours *CVRP* ).
4. DIMENSION : dans le cas du *CVRP* cela indique le nombre de nodes avec le dépôt.
5. EDGE\_WEIGHT\_TYPE : cela nous indique comment calculer les distances (dans mon cas je peux simplement utiliser le théorème de Pythagore).
6. CAPACITY : ce nombre indique combien de marchandise les camions peuvent transporter.



Fig. Exemple de spécification pour un fichier .vrp

Les données sont composées de trois sections :

1. NODE\_COORD\_SECTION : chaque ligne de cette section est composée de l’index du node puis ses coordonnés X et Y.
2. DEMAND\_SECTION : cette section est composée de lignes contenant l’index du node et la quantité demandée.
3. DEPOT\_SECTION : cela indique quelles nodes peuvent être le dépôt, cette liste est terminée par le -1.

La fin de la section de données est définie par *EOF.*

### Fichier .dat

Fig. Exemple de données dans un fichier .vrp

Les fichiers .dat sont composés d’une seule section. Chaque ligne est composée de l’index du node, suivi par les coordonnées X et Y, puis quatre nombres désignant les fenêtres de livraison (ces données sont inutiles car le temps n’est pris en compte dans mon cas), ensuite la quantité demandée et finalement le temps qu’il prend pour servir le node et si on doit livrer (=0) ou ramasser (=1).

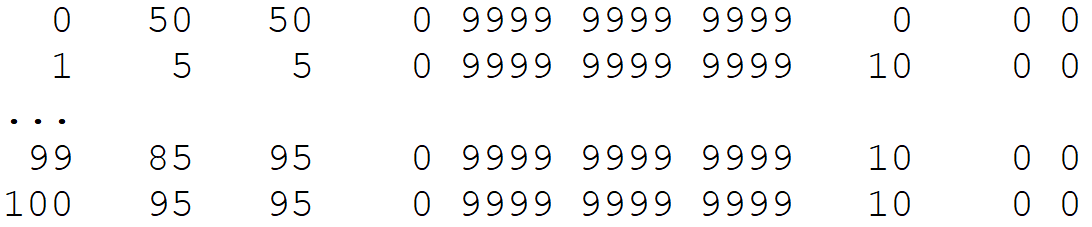


Fig. Exemple de données dans un fichier .dat

## Déroulement du programme

Quand le programme démarre le DataImporter parse un fichier .vrp et crée un DepotManager, par suite de cela une méthode démarre qui crée des tournées de camion aléatoirement. Puis le dépôt et tous les clients sont dessinés sur la fenêtre, ainsi que les tournées des camions.

Quand l’utilisateur appuie sur le bouton « Start », l’algorithme démarre en prenant un node et ses deux suivant dans sa tournée, si un des suivants est le dépôt il ne le prend pas. Cette suite de node est placée dans toutes les tournées à chaque position et sauvegarde le mouvement si le coût est négatif. Après avoir comparé cette suite avec toutes les positions, le mouvement le moins coûteux est appliqué. Cette méthode est appliquée à chaque node. Si à la fin d’avoir parcouru tous les nodes il y a eu un changement à la valeur total des tournées il recommence, s’il n’y a pas eu de changement le programme recommence mais avec deux nodes à la fois. La même chose est faite pour passer à un. Et quand il n’y a plus de changement avec un à la fois, il termine le programme.

À la fin du programme, dans la console, le coût et les nodes de chaque tournée est affichée, ainsi que le coût total des tournées.

# Tests

## Conditions de tests

## Plan de test

## Rapport de tests

# Conclusion

## Difficultés rencontrées

## Améliorations possibles

## Bilan personnel

# Annexes

## Planning prévisionnel

## Planning réel