

Fig. Exemple de résultat d'un programme CVRP

Documentation Technique

CVRP Viewer

08.06.2020

Stefano Cirieco

**CFPT**

**IFA-P3B**

**V1.0**

Table des matières

[1 Introduction 2](#_Toc42545555)

[2 Rappel de l'énoncé 2](#_Toc42545556)

[2.1 Objectifs du projet 2](#_Toc42545557)

[2.2 Inventaire du matériel 2](#_Toc42545558)

[2.3 Inventaire des logiciels 2](#_Toc42545559)

[2.4 Travail à rendre 2](#_Toc42545560)

[2.5 Contraintes 2](#_Toc42545561)

[3 Analyse fonctionnelle 3](#_Toc42545562)

[3.1 Fonctionnement vu par l’utilisateur 3](#_Toc42545563)

[3.2 Description de l’utilisation et des fonctionnalités 3](#_Toc42545564)

[4 Analyse organique 3](#_Toc42545565)

[4.1 Classes 3](#_Toc42545566)

[4.1.1 Node 3](#_Toc42545567)

[4.1.2 DepotManager 4](#_Toc42545568)

[4.1.3 Truck 4](#_Toc42545569)

[4.1.4 Movement 4](#_Toc42545570)

[4.1.5 DataImporter 4](#_Toc42545571)

[4.2 Format des fichiers 4](#_Toc42545572)

[4.2.1 Fichier .vrp 4](#_Toc42545573)

[4.2.2 Fichier .dat 5](#_Toc42545574)

[4.3 Déroulement du programme 5](#_Toc42545575)

[4.4 Déroulement de l’algorithme 6](#_Toc42545576)

[5 Tests 7](#_Toc42545577)

[5.1 Conditions de tests 7](#_Toc42545578)

[5.2 Plan de test 7](#_Toc42545579)

[5.3 Rapport de tests 7](#_Toc42545580)

[6 Conclusion 7](#_Toc42545581)

[6.1 Difficultés rencontrées 7](#_Toc42545582)

[6.2 Améliorations possibles 8](#_Toc42545583)

[6.3 Bilan personnel 8](#_Toc42545584)

[7 Annexes 8](#_Toc42545585)

[7.1 Planning 8](#_Toc42545586)

[7.2 Code source 8](#_Toc42545587)

# 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 permet de visualiser et résout 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.

# 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 (Windows 10)
* 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
* Rapport TPI
* 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 optimum local, puisqu’on commence avec des tournées aléatoires la solution optimum 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. Cette fonctionnalité n’est pas demandée, donc ne sera pas implémentée.

# 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 « Solve » pour débuter l’optimisation et un bouton « Choose file » pour choisir le fichier à parser.

## Description de l’utilisation et des fonctionnalités

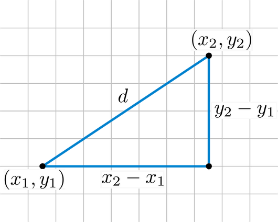
Quand l’utilisateur appuiera sur le bouton « Choose file » une boite de dialogue, permettant de choisir uniquement un fichier .vrp ou .dat, apparait. Après avoir choisie un fichier la fenêtre affichera les nouveaux clients avec des tournées aléatoires.

Quand l’utilisateur appuiera sur le bouton « Solve » 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/Y et pour le dessiner j’ajoute un facteur de 4 à X/Y pour obtenir un nombre de pixels, 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. Ces calcules sont fait dans un plan euclidien 2D, c’est la manière la plus évidente pour mesurer une distance entre deux points. Il n’y a pas de changement de hauteur, donc avoir 2 pixels entre deux points veut dire que leur distance est de 2 pixels. Pour calculer ces distances on utilise le théorème de Pythagore √((x2-x1)2 +(y2-y1)2) puis on arrondi à l’entier le plus proche. Voici comment la formule est implémenté dans la classe Node :



### DepotManager

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

Il y a une méthode pour ajouter et récupérer un client et une pour déclarer quelle node est le dépôt. Il peut créer des tournées aléatoires et trouver l’optimum local.

### 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>*.

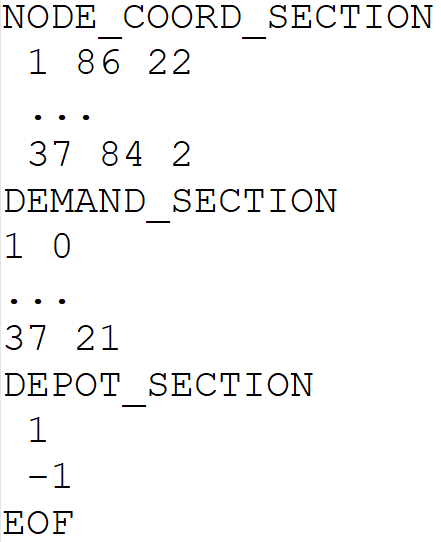
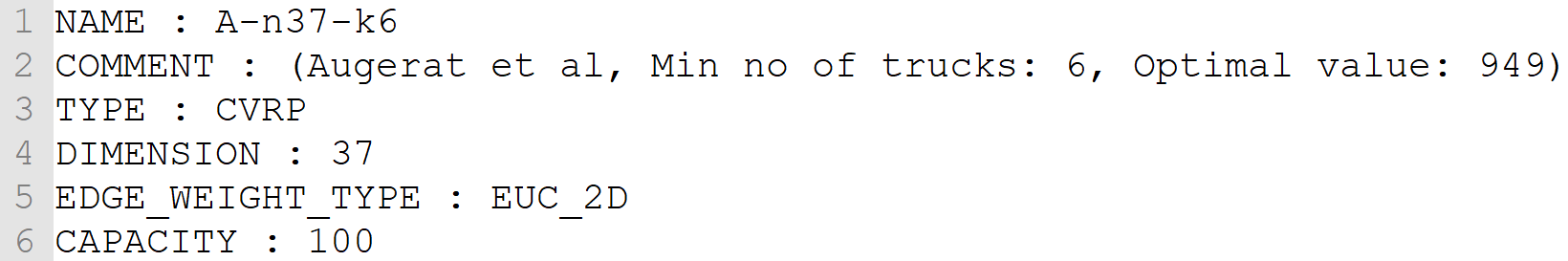
1. NAME : contient le nom du fichier.
2. COMMENT : contient l’auteur du problème ou de la solution, le nombre de camions utilisé et la valeur optimale de la solution.
3. TYPE : spécifie le type de donnée (dans mon cas cela sera toujours *CVRP* ).
4. DIMENSION : dans le cas du *CVRP* cela indique le nombre 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.*

Fig. Exemple de données dans un fichier .vrp

### Fichier .dat

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).

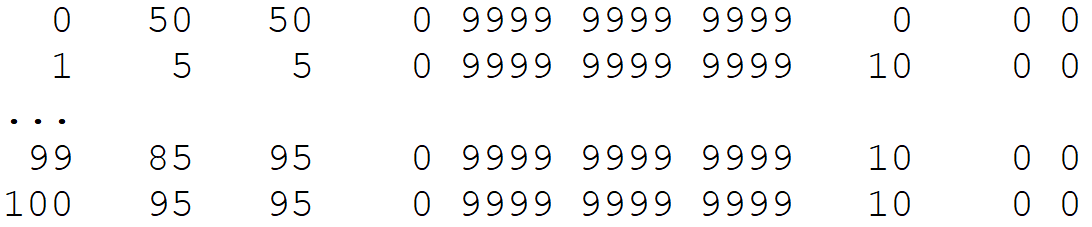


Fig. Exemple de données dans un fichier .dat

## 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.

Si l’utilisateur souhaite utiliser un autre fichier .vrp ou .dat, il peut appuyer sur le bouton « Choose file ». Quand il a choisi un fichier l’affichage se rafraichira pour montrer les nouveaux emplacements des nodes, ainsi que des tournées aléatoirement créées.

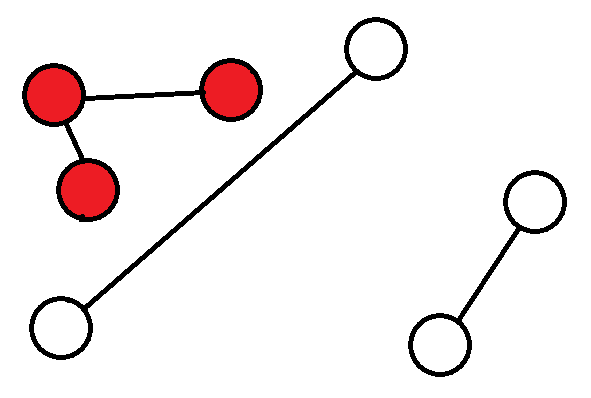
Quand l’utilisateur appuie sur le bouton « Solve », l’algorithme démarre en prenant un node et ses deux suivant dans sa tournée, si un des suivants est le dépôt il ne le prend pas. Cette suite de node est placée dans toutes les tournées à chaque position et sauvegarde le mouvement négatif le moins chère. Après avoir comparé cette suite avec toutes les positions, il passe au prochain node. Quand il a fini de comparer tous les nodes avec tous les nodes, il tri la liste des mouvements pour trouver le moins chère, puis le moins coûteux est appliqué. S’il n’y pas de mouvement négatif 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 node à la fois, il termine le programme.

Une image contenant objet, horloge

Description générée automatiquementÀ la fin du programme le coût total est affiché dans la console et dans la barre de titre de la fenêtre. La console affiche aussi le coût et les nodes de chaque tournée.

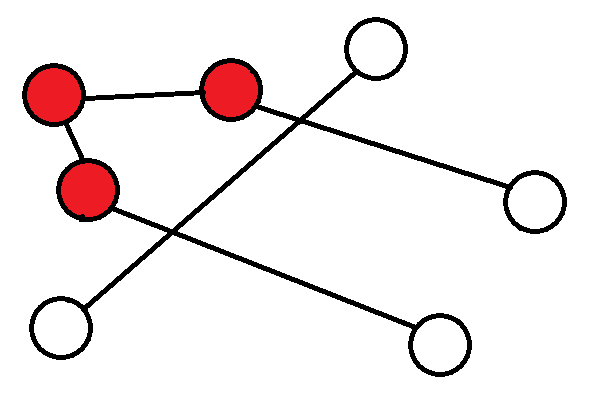
## Déroulement de l’algorithme

Dans l’exemple si dessous, il y a deux tournées de camion un avec cinq nodes et l’autre avec deux, le dépôt n’est pas visible.

Quand l’algorithme débute, il prend un node avec ses deux suivants. Dans l’image si contre il prend les trois nodes rouges. Il sauvegarde le coût de la tournée où ces nodes rouges sont placés.

Ex. Algo Sélection des 3 nodes

Ensuite il les enlève de leur tournée actuelle et sauvegarde le nouveau coût de cette tournée. On se retrouve avec deux tournées qui ont chacun deux nodes, et trois nodes qui n’ont plus de tournée.

La prochaine étape est de placer ces trois nodes dans toutes les positions de chaque tournée même leur chemin original. Pour cet exemple on va les insérer dans l’autre tournée. Avant l’insertion la vérification, que la capacité du camion de la tournée n’est pas dépassée à l’ajout des nouveaux nodes, est effectuée. Si la capacité n’est pas dépassée, le coût de la tournée est sauvegardé.

Ex. Algo Rétirrement des 3 nodes

Après avoir inséré les nodes rouges dans la nouvelle tournée, le coût de la tournée modifiée est sauvegardé.

Ex. Algo Insertion des 3 nodes dans une nouvelle tournée

Pour savoir si ce changement coute moins qu’avant, le programme calcule la différence entre la somme des coûts des deux tournées après et avant la modification.

Si le résultat est négatif, le changement est sauvegardé. Après avoir parcouru toutes les possibilités pour ces trois nodes, le changement qui coute le moins est sauvegardé dans une liste. Quand le programme a trouvé le meilleur changement de chaque node il tri la liste pour récupérer le meilleur des meilleurs, puis l’applique à la solution.

Quand il arrive à un point où il n’y a aucun changement qui améliore la solution, il recommence tout, mais avec deux node à la fois, puis finalement avec un. Quand aucun changement n’est applicable le programme s’arrête.

# Tests

## Conditions de tests

Pour tester le programme j’ai créé une solution de Testes Unitaires pour tout tests non graphiques. Pour tester l’affichage des nodes et des tournées, il y a une fonction qui dessine, sur la fenêtre, la solution optimale d’un fichier .vrp.

## Plan de test

|  |  |  |  |
| --- | --- | --- | --- |
| T**est** | **Auteur** | **Scénario** | R**ésultat attendu** |
| T001 | SC | Un fichier .vrp doit-être parser, et les valeurs de certain nodes sont testés | Un DepotManager est créé avec un tableau de nodes qui ont les mêmes valeurs que dans le fichier |
| T002 | SC | Un fichier .dat doit-être parser, et les valeurs de certain nodes sont testés | Un DepotManager est créé avec un tableau de nodes qui ont les mêmes valeurs que dans le fichier |
| T003 | SC | La solution optimale d’un fichier .vrp est appliqué à une List de Truck | Le coût total des tournées doit-être de 949 |
| T004 | SC | Enlever un node d’une tournée puis le remettre au même endroit | Le coût avant et après le déplacement doivent être les mêmes |
| T005 | SC | Afficher la solution optimale d’un CVRP sur la fenêtre | La comparaison à l’œil entre la fenêtre et une image de la solution doivent se ressembler |

## Rapport de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | T**esteur** | T**est** | R**ésultat obtenu** | **OK** |
| 25.05.2020 | SC | T001 | Un DepotManager est créé avec un tableau de nodes qui ont les mêmes valeurs que dans le fichier | OUI |
| 26.05.2020 | SC | T002 | Un DepotManager est créé avec un tableau de nodes qui ont les mêmes valeurs que dans le fichier | OUI |
| 26.05.2020 | SC | T003 | Le coût total des tournées est de 949 | OUI |
| 26.05.2020 | SC | T004 | Le coût avant et après le déplacement sont les mêmes | OUI |
| 26.05.2020 | SC | T005 | La fenêtre et une image de la solution se ressemble | OUI |

# Conclusion

## Difficultés rencontrées

Cet algorithme parcourt tous les nodes et pour chaque node parcours tout les autres nodes, tout ça pour faire une modification à la solution, donc il fait beaucoup de calcule. Une grande difficulté était quand une boucle infinie apparaissait, car elle surgissait à un autre endroit à chaque lancement du projet. Pour essayer de trouver pourquoi cette boucle infinie se créait il fallait débugger, mais le grand problème était que je ne savais pas où cette boucle se créait, donc je débuggais dès le début de l’algorithme pour ne pas manquer la boucle infinie. Après beaucoup de dessins dans mon journal de bord et de débuggage, j’ai trouvé qu’à un certain moment un node n’était plus pointé dessus dans les tournées, donc en essayant de le retrouver dans les tournées la boucle infinie apparaissait. L’erreur était dans une méthode pour appliquer les mouvements. Après avoir compris d’où venait le problème, il est beaucoup plus simple de le réparer.

## Améliorations possibles

*Recherche par tabou*

## Bilan personnel

Une grande difficulté avec un projet comme celui-ci est de pouvoir comprendre le problème que l’on veut résoudre et comprendre comment le résoudre. Tout d’abord il fallait que je comprenne ce qu’est un CVRP et quel est son but.

# Annexes

## Planning

## Code source