Project : Delivery planning

Mathias SOMMACAL & Yann CAUCHEPIN

Academic year 2018 - 2019

Summary

Introduction

Delivery companies such as *La Poste* to deliver parcels or *Total* to supply petrol service stations need to plan their deliveries to make them optimal. According to researchers Geir Hasle and Knut-Andreas Lie, the use of algorithms would enable these companies to make savings of 5%. These carriers have one major constraint : their trucks have a limited loading capacity. This planning problem is referred to in the literature as **CVRP** : Capacitated Vehicle Routing Problem.
We deal only with the special case where each vehicle has a unique loading capacity and starts from the same depot to make its deliveries, then returns to it at the end of its tour.

Chapter 1

Analysis and design

1.1 Specification details

The project therefore addresses the problem of cost in carrier deliveries. Here, we only consider the costs associated with the distance covered, and not the drivers' salaries or those associated with vehicle maintenance.
We assume that trucks have identical storage capacity, and that they all depart from the same depot to make their deliveries, then return there at the end of their rounds. Similarly, we assume that carriers cannot pass a customer without delivering, and that a customer cannot be partially delivered by a truck.
Our program should therefore return the minimum cost of a delivery, as well as the order of customers for each delivery round. The resulting solution is not necessarily optimal. In fact, the proposed method is based on heuristic methods.
Finally, the input files have a specific format, where the first line corresponds to the number of customers to be delivered, the second to the maximum capacity in units of the vehicles, the third to the delivery requests in units of each customer which must be less than the maximum capacity of the vehicles, and the last lines correspond to the various distances.
Here's an example of a source file :
In this example, the number of customers is 5. The trucks have a capacity of 10 units. Customer C_3 requires 4 units. Between the depot and customer C_5, the distance is 35,

Figure 1.1: Example of a source file

```
5
10
5  4  4  2  7
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
```

while between customer C_3 and C_4, the distance is 25. We'll continue our analysis using this example source file.

1.2 Choice of graph modeling

Given the specifications imposed on us, we decide to model this problem using a graph, in which the vertices represent the customers and the depot, while the edges represent the routes between the different customers, as well as between the customers and the depot, weighted by the distance of the route. The quantities requested by each customer are indicated in superscript on each vertex. Customers are denoted by C_1, C_2, ..., C_n and the depot corresponds to vertex D.

1.3 Algorithm sequence

1.3.1 Giant tour

The Giant Tour algorithm is designed to create a tour that passes through all the customers and covers them only once. We don't consider the depot, just the customers and the distances between them. The giant tour program starts with the first customer given as a parameter and then searches for the second closest customer. For each iteration, the next customer to be traversed is the one closest to the current customer and not already taking part in the giant tour.

Data structure

The algorithm needs the distances separating customers from each other to be able to search for the closest customer to the current one that is not already marked. These distances are read through a text file in the form of a matrix A where the distances separating the customers (or with the depot) i and j are noted as A_{ij} and A_{ji}. To minimize the memory space allocated to this redundant data, we wanted to replace the matrix with a table of successors. However, we quickly realized that traversing this table to determine the minimum distance among the remaining customers would have been too complicated. So, in the example below, between customer C_4 and the depot ($j = 0$), the distance is 40.

$$\begin{bmatrix} 0.0 & 20.0 & 25.0 & 30.0 & 40.0 & 35.0 \\ 20.0 & 0.0 & 10.0 & 15.0 & 40.0 & 50.0 \\ 25.0 & 10.0 & 0.0 & 30.0 & 40.0 & 35.0 \\ 30.0 & 15.0 & 30.0 & 0.0 & 25.0 & 30.0 \\ 40.0 & 40.0 & 40.0 & 25.0 & 0.0 & 15.0 \\ 35.0 & 50.0 & 35.0 & 30.0 & 15.0 & 0.0 \end{bmatrix}$$

Next, for the sequence of customers to be visited on the giant tour, we decided to use a vector T of size n, where n corresponds to the number of customers. This data structure enables translation between the numerical order of customers and the order of the giant tour. For example, here, the first customer on the giant tour (T_0) is customer C_3.

$$T = \begin{bmatrix} 3 & 1 & 2 & 5 & 4 \end{bmatrix}$$

Pseudo-code

Algorithme 1.1: Tour géant

```
1  Fonction tour_geant(debut, n, M) : Vecteur d'Entiers de taille
       n
2    D :
3      debut : Entier {Premier client du tour géant}
4      n     : Entier
5      M     : Matrice de Réels de taille n×n
6
7    L :
8      i     : Entier
9      j     : Entier
10     ind_min : Entier {Indice du plus proche voisin}
11     min   : Réel {Distance du plus proche voisin}
12     courant : Entier {Client courant}
13     mark  : Vecteur d'Entiers de taille n {Tableau des sommets
           déjà visités}
14     T     : Vecteur d'Entiers de taille n
15
16   {Initialisation}
17   Pour i allant de 1 à n Faire
18     mark[i] ← FAUX
19   Fait
20
21   courant ← debut
22
```

```
23    {Parcours de tous les clients}
24    Pour i allant de 1 à n Faire
25      {On ajoute le client courant dans le tableau T et on le
            marque}
26      T[i] ← courant
27      mark[courant] ← VRAI
28
29      {On initialise les variables pour la recherche du plus
            proche voisin}
30      j ← 1;
31      Tantque (mark[j] = VRAI ou M[courant][j] = 0) et (j ≤ n)
            Faire
32        j ← j + 1
33      Fait
34
35      ind_min ← j
36      min ← M[courant][ind_min]
37
38      {On parcourt une colonne à la recherche du plus proche
            voisin non marqué}
39      Pour j allant de 2 à n+1 Faire
40        Si (mark[j] = FAUX) et M[courant][j] ≠ 0 Alors
41          Si M[courant][j] < min Alors
42            min ← M[courant][j]
43            ind_min ← j
44          FSi
45        FSi
46      Fait
47
48      {On change de client}
49      courant ← ind_min
50    Fait
51
52    R :
53      T
54  FFonction
```

1.3.2 Cutting procedure

The Split algorithm is used to determine an auxiliary graph, in which edges represent elementary cycles (tours) and vertices represent transitions between two tours. Edges are weighted by their cost.

Data structure

There are several ways of representing graphs : matrix and tabular. The graph produced by the slicing procedure is then used by Bellman's successor algorithm to calculate the shortest path. A tabular representation, with a table of successors, seemed more logical to us. We therefore defined the following data structure :

Algorithme 1.2: Définition de la structure graphe

```
1  type graphe structure
2    n : Entier {Nombre de sommets}
3    m : Entier {Nombre d'arêtes}
4    head  : Vecteur d'Entiers de taille n
5    succ  : Vecteur d'Entiers de taille m
6    cost  : Vecteur de Réels de taille m {Pondérations des arê
       tes}
7  fin
```

The list of successors to a given vertex is stored contiguously in a dynamic vector of size m. The *head* vector contains the indices for finding a vertex's successors. The *cost* vector works in the same way as the *succ* vector, but stores the weights. The integer n can be predefined at the start of the algorithm, as can the size of *head*, which is $n-1$. As for m, it will be modified with each iteration, as will the dynamic size of the vectors *succ* and *cost*. Adding an arc will therefore require memory reallocations.

$$head = \begin{bmatrix} 0 & 2 & 4 & 5 & 7 \end{bmatrix}$$

$$succ = \begin{bmatrix} 1 & 2 & 2 & 3 & 3 & 4 & 5 & 5 \end{bmatrix}$$

$$cost = \begin{bmatrix} 60.0 & 65.0 & 40.0 & 55.0 & 50.0 & 70.0 & 90.0 & 80.0 \end{bmatrix}$$

In the example above, the H_1 vertex has the $\{2, 3\}$ set as its successors.

Pseudo-code

Algorithme 1.3: Ajout d'un arc

```
1  Action ajouter_arc(G, predecesseur, successeur, cout)
2      D :
3          G    :   Graphe
4          predecesseur    :   Entier
5          successeur  :   Entier
6          cout    :   Réel
7
8      Si G.m = 0 Alors
9      G.head[0] ← G.m
10     G.succ[G.m] ← successeur
```

```
11      G.cost[G.m] ← cout
12      G.m ← G.m + 1
13      Sinon
14      {Tableau head}
15      Si G.head[predecesseur] = −1 Alors
16        G.head[predecesseur] ← G.m
17      FSi
18
19      {Tableau succ}
20      G.succ ← Réallouer G.m+1 cases mèmoires de la taille
            d'un entier
21      G.succ[G.m] ← successeur
22
23      {Tableau cost}
24      G.cost ← Réallouer G.m+1 cases mèmoires de la taille
            d'un rèel
25
26      G.cost[G.m] ← cout
27
28      G.m ← G.m + 1
29    FSi
30  FAction
```

Algorithme 1.4: Procédure de découpage

```
1   Action split(n, T, Q, dist, q, H)
2     D :
3       n : Entier {Nombre de clients}
4       T : Vecteur d'Entiers de taille n {Vecteur obtenu par l'
            algorithme du Tour géant}
5       Q : Entier {Capacité des camions}
6       dist : Matrice de Réels de taille n×n {Distances}
7       q : Vecteur d'Entiers de taille n {Demandes de livraison}
8     D/R :
9       H : Graphe
10    L :
11      i : Entier
12      j : Entier
13      load : Entier {Chargement du camion}
14      cost : Réel {Distance parcourue par le camion}
15
16    Pour i allant de 1 à n Faire
17      j ← i
18      load ← 0
19
```

```
20      Tantque j ← n−1 et load ≤ Q Faire
21        load ← load + q[T[j]]
22
23        Si i = j Alors
24          cost ← dist[1][T[i]] + dist[T[i]][0]
25        Sinon
26          cost ← cost − dist[T[j−1]][0] + dist[T[j−1]][T[j]] +
                dist[T[j]][0]
27        FSi
28
29        Si load ≤ Q Alors
30          ajouter_arc(H, i, j+1, cost)
31        FSi
32
33        j ← j + 1
34      Fait
35    Fait
36  FAction
```

1.3.3 Shortest path algorithm

In order to determine which tour succession is the least costly, we now need to use a shortest-path algorithm on the H graph.
In our graphing and combinatorics course, we learned about two shortest-path algorithms : Dijkstra's algorithm and Bellman's algorithm. We immediately notice that for the graph H, the conditions for using both algorithms are met. Indeed, the values of the weights are positive, since they correspond to distances (condition of Dijkstra's algorithm) and there are no circuits (condition of Bellman's algorithm).
However, we note that the graph H is always in topological order. Moreover, each level is composed of just one vertex ($N_i = \{H_i\}$). This gives the Bellman algorithm an advantage. Indeed, under these conditions, it has a complexity in $O(m)$ versus $O(n^2)$ for Dijkstra's algorithm. We therefore decide to use Bellman's algorithm as the shortest path algorithm for our project.

Data structure

Bellman's algorithm modifies the data/results *potentials* and *pere*. These vectors are allocated with a predefined size of $n + 1$. The *pere* vector contains the indices of the predecessor customers, which are used to propagate the lowest potentials for each customer. The *potentials* vector contains the potentials of each vertex.

$$potentiels = \begin{bmatrix} 0.0 & 60.0 & 65.0 & 115.0 & 185.0 & 205.0 \end{bmatrix}$$

$$pere = \begin{bmatrix} 0 & 0 & 0 & 1 & 3 & 3 \end{bmatrix}$$

In this example, vertex H_4 has a potential of 185. To reach this potential, we must first pass through vertex H_3. The H_3 vertex has a potential of 115. To reach this potential, we must first pass through vertex H_1 and so on. We can therefore easily determine the shortest path with this vector.

Pseudo-code

Algorithme 1.5: Bellman-Ford

```
1  Action bellman(G, potentiels, pere)
2      D :
3          G    :   graphe
4      D/R :
5          potentiels  :   vecteur de Réels de taille G.n
6          pere        :   vecteur d'Entiers de taille G.n
7      L :
8          i    :   Entier
9          j    :   Entier
10         x    :   Entier
11         y    :   Entier
12
13     {Initialisation des vecteurs potentiels et pere}
14   potentiels[0] ← 0
15   pere[0] ← 0
16
17     Pour i allant de 1 à G.n Faire
18     potentiels[i] ← −1
19     pere[i] ← 0
20   Fait
21
22   {Pour tout x successeur de la racine}
23   Pour i allant de G.head[0] à G.head[1] Faire
24     x ← G.succ[i]
25     potentiels[x] ← G.cost[i]
26     Fait
27
28   {Pour tout y successeur des autres sommets}
29   Pour x allant de 1 à G.n−1 Faire
30         Pour j allant de G.head[x] à G.head[x+1] Faire
31             y ← G.succ[j]
32       Si potentiels[x] + G.cost[j] < potentiels[y] ou2
             potentiels[y] = −1 Alors
33         potentiels[y] ← potentiels[x] + G.cost[j]
34         pere[y] ← x
35       FSi
```

```
36      Fait
37    Fait
38
39    {Le dernier sommet n'a pas de successeurs}
40    Si potentiels[G.n-2] + G.cost[G.m-1] < potentiels[G.n-1] ou2
          potentiels[G.n-1] = -1 Alors
41      potentiels[G.n-1] ← potentiels[G.n-2] + G->cost[G.m-1]
42      pere[G.n-1] ← G.n-2
43    FSi
44  FAction
```

1.3.4 Programme principal

At the start, the main program declares the data structures and enables the import of the data contained in the source file received as a parameter, at runtime. It links the various algorithms presented above. Finally, it displays the optimized distance to cover all customers, corresponding to the potential present at the last index of the *potentials* vector. And from the *pere* vector output by Bellman's algorithm, it organizes the tours as follows: The program takes the index i of the giant tower in the last cell of the vector *pere* and compares it with the index in cell i. The difference between these two indices represents the giant tower indices visited in a carrier's tour. And so on, until the first square of the *pere* vector is reached. A translation between the giant tour vector and the customer order is performed.

Pseudo-code

Algorithme 1.6: Programme principal

```
1   Action progPrincipal(param1, param2)
2     D :
3       param1  : Chaîne de caractère {Paramètre 1 : adresse du
            fichier}
4       param2  : Entier {Paramètre 2 : premier client}
5     L :
6       i : Entier
7       j : Entier
8       fic : Fichier {Fichier source}
9       n : Entier {Nombre de client}
10      Q : Entier {Capacité des véhicules}
11      q : Vecteur dynamique d'Entiers de taille n+1 {Demandes
            des clients}
12      M : Matrice dynamique de Réels de taille (n+1)×(n+1) {
            Distances}
13      T : Vecteur dynamique d'Entiers de taille n {Tour géant}
14      H : Graphe {Structure du graphe auxiliaire}
```

```
15      potentiels : Vecteur dynamique de Réels de taille n+1 {
            Potentiels des sommets de H}
16      pere : Vecteur dynamique d'Entiers de taille n+1 {Pré
            decesseurs de coût minimal}
17
18    {Import des données depuis le fichier source}
19    fic ← Ouvrir param1 en mode lecture
20
21    Lire n à partir de fic
22    Lire Q à partir de fic
23
24    q ← Allouer n+1 cases mèmoires de la taille d'un entier
25    Pour i allant de 0 à n Faire
26      Lire q[i] à partir de fic
27    Fait
28
29    M ← Allouer n+1 cases mèmoires de la taille d'un pointeur
          vers un rèel
30    Pour i allant de 0 à n+1 Faire
31      M[i] ← Allouer n+1 cases mèmoires de la taille d'un rèel
32      Pour j allant de 0 à n+1 Faire
33        Lire M[i][j] à partir de fic
34      Fait
35    Fait
36
37    Fermer fic
38
39    {Création d'un tour géant}
40    T ← Allouer n cases mèmoires de la taille d'un entier
41
42    tour_geant(param2, n, M, T)
43
44    {Construction d'un graphe auxiliaire avec la procédure SPLIT
          }
45    init_graphe(H, n+1)
46
47      split(n+1, T, Q, M, q, H)
48
49      {Application de l'algorithme de Bellman}
50
51      potentiels ← Allouer n+1 cases mèmoires de la taille
            d'un rèel
52      pere ← Allouer n+1 cases mèmoires de la taille d'un
            entier
53
```

```
54      bellman(H, potentiels, pere)
55
56      clear_graphe(H)
57
58      {Affichage du résultat final}
59
60      Afficher "Coût : ", potentiels[n]
61
62      i ← n
63      Tantque i≠0 Faire
64        Afficher "Tournée : "
65        Pour j allant de pere[i] à i Faire
66          Afficher T[j]
67        Fait
68        i ← pere[i]
69      Fait
70
71       Libère q
72
73      Pour i allant de 0 à n+1 Faire
74         Libère M[i]
75      Fait
76       Libère M
77
78       Libère T
79       Libère potentiels
80       Libère pere
81  FAction
```

Organization of source and header files

To optimize group work, we have chosen to create source files and their respective header files, then group them together in the main program *main.c*.
The *main.c* file depends on the *tour_geant.h*, *split.h* and *bellman.h* files. Each source file depends on its header file. Finally, the *bellman.h* file uses the graph data structure defined in *split.h*.

Chapter 2

Instructions for use

2.1 Installation

To install it, simply use the *makefile* :

```
make
```

2.2 Use

We want to run our program from a *exemple.dat* :

Figure 2.1: exemple.dat

```
 5
10
 5   4   4   2   7
 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
```

To run the algorithms from client 2, simply do :

```
./main exemple.dat 2
```

The first and second parameters correspond respectively to the path to the source file and the number of the first client.
We thus obtain the following result :

```
$ ./main exemple.dat 2
Coût : 205.000000
Tournée : 4 5
```

```
Tournée : 1 3
Tournée : 2
```

Starting with customer 2, we obtain a cost of 205. There are 3 rounds to do :

- Round 1 : C_2
- Round 2 : C_1, C_3
- Round 3 : C_4, C_5

Chapter 3

Test phase

Using separate compilation, we have tested each algorithm separately, to ensure that they are all valid, taking into account the starting assumptions for each.

3.1 Giant tour

We choose the following parameters for the giant lathe algorithm :

- The first customer is customer 3 ($start = 3$).
- The number of customers is 5 ($n = 5$).
- The distance matrix is as follows :

$$dist = \begin{bmatrix} 0.0 & 20.0 & 25.0 & 30.0 & 40.0 & 35.0 \\ 20.0 & 0.0 & 10.0 & 15.0 & 40.0 & 50.0 \\ 25.0 & 10.0 & 0.0 & 30.0 & 40.0 & 35.0 \\ 30.0 & 15.0 & 30.0 & 0.0 & 25.0 & 30.0 \\ 40.0 & 40.0 & 40.0 & 25.0 & 0.0 & 15.0 \\ 35.0 & 50.0 & 35.0 & 30.0 & 15.0 & 0.0 \end{bmatrix}$$

Applying the algorithm manually, we find a giant tour passing through customers C_3, C_1, C_2, C_5 and C_4 respectively.
This is consistent with the table constructed by the algorithm we implemented :

$$T = \begin{bmatrix} 3 & 1 & 2 & 5 & 4 \end{bmatrix}$$

3.2 Cutting procedure

We choose the following parameters for the cutting procedure :

- The number of customers is 5 ($n = 5$).

- The array T corresponds to the array found with the giant lathe algorithm :

$$T = \begin{bmatrix} 3 & 1 & 2 & 5 & 4 \end{bmatrix}$$

- The vehicle capacity is 10 ($Q = 10$).
- The distance matrix is as follows :

$$dist = \begin{bmatrix} 0.0 & 20.0 & 25.0 & 30.0 & 40.0 & 35.0 \\ 20.0 & 0.0 & 10.0 & 15.0 & 40.0 & 50.0 \\ 25.0 & 10.0 & 0.0 & 30.0 & 40.0 & 35.0 \\ 30.0 & 15.0 & 30.0 & 0.0 & 25.0 & 30.0 \\ 40.0 & 40.0 & 40.0 & 25.0 & 0.0 & 15.0 \\ 35.0 & 50.0 & 35.0 & 30.0 & 15.0 & 0.0 \end{bmatrix}$$

- Customers have the following requests :
  - C_1 : 5
  - C_2 : 4
  - C_3 : 4
  - C_4 : 2
  - C_5 : 7

The implemented algorithm gives the graph in the form of successor tables :

$$head = \begin{bmatrix} 0 & 2 & 4 & 5 & 7 \end{bmatrix}$$

$$succ = \begin{bmatrix} 1 & 2 & 2 & 3 & 3 & 4 & 5 & 5 \end{bmatrix}$$

$$cost = \begin{bmatrix} 60.0 & 65.0 & 40.0 & 55.0 & 50.0 & 70.0 & 90.0 & 80.0 \end{bmatrix}$$

3.3 Bellman algorithm

Our implementation gives the following results :

$$potentiels = \begin{bmatrix} 0.0 & 60.0 & 65.0 & 115.0 & 185.0 & 205.0 \end{bmatrix}$$

$$pere = \begin{bmatrix} 0 & 0 & 0 & 1 & 3 & 3 \end{bmatrix}$$

The shortest path is through H_0, H_1, H_3 and H_5.

3.4 Final results

We then tested our program for each test file given by the subject. For the file *cvrp_100_1_det.dat* we obtain :

- Starting with customer 1 :

```
$ ./main cvrp_100_1_det.dat 1
Coût : 14289.270508
Tournée : 40 59 31
Tournée : 15 82 70 87 45 41 36
Tournée : 76 50 60 99 92
Tournée : 34 11 24 8 91 56 3
Tournée : 64 20 96 39 78
Tournée : 83 58 66 80 86 38
Tournée : 4 17 84 28 67
Tournée : 61 48 57 30 68 54 10
Tournée : 21 79 43 44 62 19 22 37
Tournée : 97 65 85 25 47 81
Tournée : 14 33 27 74 75 77 35 73 100
Tournée : 69 12 51 94 32 6
Tournée : 71 72 13 95 26 46 55 90 5
Tournée : 63 42 16 53 98
Tournée : 2 18 9 7 29 88
Tournée : 1 93 89 49 23 52
```

- Starting with customer 3 :

```
$ ./main cvrp_100_1_det.dat 3
Coût : 14162.331055
Tournée : 10 4 67 83 58 34 11
Tournée : 87 24 8 91 45 41 36
Tournée : 31 76 50 60 99 92
Tournée : 15 86 38 64 40 59
Tournée : 61 48 43 54 68 70 82
Tournée : 44 62 19 22 37
Tournée : 7 9 18 79 21 81 80 57 30
Tournée : 14 20 96 39 78
Tournée : 12 51 94 32 6
Tournée : 97 2 52 46 55 90 5 69
Tournée : 47 33 27 74 75 77 35 73 100
Tournée : 66 17 84 28 65 85 25
Tournée : 23 49 89 93 1
Tournée : 53 98 71 72 13 95 26
Tournée : 3 56 29 88 63 42 16
```

For the file *c*vrp_100_1_r.dat we obtain :

- Starting with customer 1 :

```
$ ./main cvrp_100_1_r.dat 1
Coût : 1014.550049
Tournée : 26 12 76 50 78 34 35 71 66 65 29 24 4 41
Tournée : 2 57 15 43 38 86 13
Tournée : 93 85 91 100 37 98 61 16 44 14 42 87
Tournée : 89 6 94 95 97 92 59 99 96
Tournée : 62 10 90 32 63 64 46 8 45 17 84 5 60 83 18 52
Tournée : 31 88 7 82 48 47 36 49 19 11
Tournée : 80 68 77 3 79 33 81 9 51 20 30 70
Tournée : 73 72 74 22 75 56 39 23 67 25 55 54
Tournée : 1 69 27 28 53 58 40 21
```

- Starting with customer 59 :

```
$ ./main cvrp_100_1_r.dat 59
Coût : 992.699951
Tournée : 26 4 24 29 78 34 35 71 66 65 52
Tournée : 75 56 39 23 67 25 55 54 80 68 12
Tournée : 87 2 57 15 43 38 41 22 74 72 73 21
Tournée : 62 10 90 32 63 64 46 8 45 17 86 44 14 42
Tournée : 31 88 7 82 48 47 36 49 19 11
Tournée : 69 1 50 76 77 3 79 33 81 9 51 20 30 70
Tournée : 53 28 27
Tournée : 18 60 83 84 5 61 16 91 100 97 95 13 58 40
Tournée : 59 92 37 98 85 93 99 96 94 6 89
```

For the file *c*vrp_100_1_c.dat we obtain :

- Starting with customer 1 :

```
$ ./main cvrp_100_1_c.dat 1
Coût : 882.290039
Tournée : 55 57 59 54 53 56 58 60
Tournée : 81 78 76 71 70 73 77 79 80
Tournée : 99 96 95 94 93 92 97 100 98
Tournée : 18 17 13 15 16 14 12 19
Tournée : 34 36 39 38 37 35 31 32 33
Tournée : 20 21 22 23 26 28 27 25 24 29 30
Tournée : 40 41 42 44 45 46 48 50 51 52 49 47 43
Tournée : 63 65 67 66 69 62 74 72 61 64 68
Tournée : 91 89 88 85 84 83 82 86 87 90
Tournée : 1 2 4 3 5 7 8 9 6 11 10 75
```

- Starting with customer 13 :

```
$ ./main cvrp_100_1_c.dat 13
Coût : 856.509949
Tournée : 80 79 77 73 70 71 76 78 81
Tournée : 59 57 55 54 53 56 58 60
Tournée : 34 36 39 38 37 35 31 32 33
Tournée : 20 21 22 23 26 28 27 25 24 29 30
Tournée : 40 41 42 44 45 46 48 50 51 52 49 47 43
Tournée : 63 65 67 66 69 62 74 72 61 64 68
Tournée : 91 89 88 85 84 83 82 86 87 90
Tournée : 98 96 95 94 93 92 97 100 99
Tournée : 10 8 9 6 4 3 5 7 75 1 2
Tournée : 13 17 18 19 15 16 14 12 11
```

All the costs obtained coincide with the expected results given in the subject.

3.4.1 Memory leaks

We also checked that our program didn't cause memory leaks by using the utility *valgrind*.

```
==15006==
==15006== HEAP SUMMARY:
==15006==     in use at exit: 0 bytes in 0 blocks
==15006==   total heap usage: 33 allocs, 33 frees, 1,050,816
   bytes allocated
==15006==
==15006== All heap blocks were freed -- no leaks are possible
```

Chapter 4

Possible improvements

The program we've developed provides an acceptable solution to the problem of deliveries constrained by loading capacity. It does not always give the optimal solution. In fact, no guarantee is given on the quality of the solution obtained, as we don't know the approximation factor of the method used. Experience has shown that for a small number of customers, the deviation from the optimal solution is minimal. To give the optimal solution for any situation, we would have to run the algorithm starting the giant tour with each customer, and compare the costs obtained. This means running the algorithm n times, where n represents the number of customers, at the expense of complexity.
Moreover, our program uses a cutting procedure designed for a single truck capacity Q. However, a delivery company often has vehicles of different types (semi-trailer, van, bicycle) with different capacities. Furthermore, logistics companies often talk about the "last mile" issue, and are experimenting with a number of innovative means of transport, such as drone delivery.
Finally, there are other costs to be taken into account when planning deliveries, such as driver working time. It could also be interesting to introduce a time constraint for deliveries. This is known as *Vehicle Routing Problem with Time Windows* (VRPTW).

Conclusion

Our program is now ready to be used to determine the minimum delivery cost for a given number of customers, and to indicate the different routes to be taken.
To carry out this project, we first went through a phase of analysis, design and then testing. In order to be able to work easily together, we divided our project into several distinct files, and precise specifications had to be defined for each of them. Modifications were then tracked using the Git version management system.
This project, which required skills in graphing, combinatorics and data structure, was very rewarding. Not only did it enable us to consolidate our knowledge in these two disciplines, but it also enabled us to develop skills to better manage a project. For example, we now know how to divide up a project so that we can work together, and we better understand the usefulness of writing specifications.