Erasmus School of Economics

Implementation of an adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem

Author:
Laila Kakar
353285

Supervisor:
Dr. Dennis Huisman

June 30, 2014

Abstract

The cumulative capacitated vehicle problem is a transportation problem with the objective to minimize the sum of arrival times at customers, subject to vehicle constraints. This type of problem focuses on satisfying the need of customers, it plays on important role concerning humanitarian aid. The purpose of this research is to implement *an adaptive large neighbourhood search heuristic for the cumulative capacitated vehicle routing problem* by Ribeiro et al. The heuristic is implemented and tested on a varied set of benchmark problems. The removal and insertion heuristics that were based on randomness performed the best. However, the performance of the heuristics were dependent on the benchmark problems.

Contents

1 Introduction

The vehicle routing problem (VRP) is one of the most important and studied combinatorial optimization problems [1],[2]. The aim of the problem is to determine the optimal set of routes for serving a given set of customers. The routes are determined in such a way to achieve one or more objectives while satisfying specific requirements. Total distance and total cost are just two of the many objectives that have been studied widely.

A relatively new objective that has not been studied that much, is the minimization of the arrival times at customers. This variant of the VRP is called the cumulative capacitated vehicle routing problem (CCVRP). Especially after a natural disaster concerning the humanitarian aid, the CCVRP can play an important role in optimizing the routes. One can imagine that for such transportation problems the deliveries should be fast and fair for all the customers [3].

The CCVRP is NP-hard, as it generalizes the NP-hard traveling repair man problem (TRP), by adding capacity constraints and a homogenous vehicle fleet [4]. This means that finding the optimal solution is difficult or at least computationally expensive, even if the instance size is small. As a result of this problem, heuristics have been developed to find solutions of high quality within acceptable time frames. Nguevue et al. proposed two memetic algorithms (MA) and two lower bounds, concerning the CCVRP with homogenous fleet and a single depot [4]. Ribeiro et al. [5] also presented a heuristic for the cumulative capacitated vehicle routing problem, an adaptive large neighborhood search heuristic (ALNS). Both heuristics were compared on test instances, the ALNS outperformed the MAs.

The purpose of this research is to implement the ALNS heuristic and test it on a varied set of benchmark problems. These benchmark problems have been widely accepted to make an unbiased comparison of the performance of the heuristics. In this paper the test instances proposed by Christofides et al. [6] are used. The benchmark problems concern one depot and a homogenous fleet of vehicles. The obtained results will be compared with results obtained by Ribeiro et al [5].

In the first part of this thesis a problem statement and literature review will be given. The remainder of this thesis will discuss the results. Finally, the conclusion will be given.

2 Problem statement and literature review

The cumulative capacitated vehicle routing problem (CCVRP) is a transportation problem with the objective to minimize the sum of arrival times at customers, subject to vehicle constraints. Vehicle constraints can concern the vehicle capacity or the total maximum route time. This type of problem focuses on satisfying the need of customers, one can think of the supply of relief goods or rescue of victims after a natural catastrophe. Unlike commercial supply chains that are putting their focus on profits and quality, humanitarian aid concentrates on minimizing suffering of victims and the number of death [4]. Especially after a extreme disaster has occurred and relief should be sent, the routing of vehicles is of great importance, since it has a great impact on the arrival times to those concerned.

2.1 Mathemetical model

Finding the optimal solution of the CCVRP is too time consuming, so heuristics are being used. In order to find the optimal solution, the formulation of the problem is of importance. The CCVRP is defined on an undirected graph $G = (V, E)$ where $V = \{0, 1, \dots, n, n+1\}$ is the node set, nodes 0 and $n+1$ correspond to the depot and $V' = V \backslash \{0, n+1\}$ is the set of customers. The set $E\{(i,j) : i, j \in V, i < j\}$ is the edge set, with $c_{ij} = c_{ji}$ the travel time associated to each edge $(i,j) \in E$. It is assumed that the travel times are symmetric and satisfy the triangle inequality. R is the set of identical vehicles with vehicle capacity Q and each customer $i \in V$ has a demand q_i. A route is defined as a circuit that starts and ends at the depot. The objective of the CCVRP is to define a set of routes such that every customer is visited exactly once and the sum of the arrival times at the customers is minimized. This is done in such a way, that the total demand served in a particular vehicle route does not exceed the capacity of the vehicle. Let t_i^k be the arrival time of vehicle k at customer i and x_{ij} a binary variable equal to 1 in case vehicle k visits customer j after customer i. As formulated by Ribeiro et al [5] the formulation of the CCVRP is as follows:

$$\begin{array}{llll}
\text{min} & \sum_{k \in R} \sum_{i \in V'} t_i^k & & (1) \\
\text{s.t.} & \sum_{i \in V} x_{ij}^k = \sum_{i \in V} x_{ji}^k & j \in V', k \in R & (2) \\
 & \sum_{k \in R} \sum_{j \in V} x_{ij}^k = 1 & i \in V' & (3) \\
 & \sum_{i \in V'} \sum_{j \in V} q_i x_{ij}^k \leq Q & k \in R & (4) \\
 & \sum_{j \in V} x_{0j}^k = 1 & k \in R & (5) \\
 & \sum_{i \in V} x_{i,n+1}^k = 1 & k \in R & (6) \\
 & (t_i^k + c_{ij}) x_{ij}^k - (1 - x_{ij}^k) M \leq t_j^k & i \in V \backslash \{n+1\}, & (7) \\
 & & j \in V, k \in R & \\
 & t_i^k \geq 0 & i \in V, k \in R & (8) \\
 & x_{ij}^k \in \{0, 1\} & i \in V, j \in V, & (9) \\
 & & k \in R &
\end{array}$$

The arrival times of the customers are minimized by the objective function (1). Constraints (2) and (3) ensure that each customer i is served exactly once by exactly one vehicle k and that the vehicle leaves the customer after arrival. Constraints (4) are capacity constraints that specify that the total demand of a vehicle route does not exceed the vehicle capacity. Constraints (5) and (6) ensure that the depot is at the beginning and end of every route. Constraints (7) makes sure that when customer j is visited after customer i, that the arrival time of customer i plus the travel time between the customers does not exceed the arrival time of customer j. This is done with the help of a large number M, it also ensures that no subtours are created. Finally, constraints (8) and (9) specify the restrictions on the decision variables.

2.2 Lower bounds

Ngueveu et al. [4] have shown that in case $|R| \geq n$ and because of the triangle inequality, the optimal solution is that exactly one customer is visited by exactly one vehicle. In other words, the optimal solution uses n vehicles and every vehicle serves only one customer. The lower bound that yields the lowest objective function value in such a situation is as follows:

$$LB_1 = \sum_{j \in V'} c_{0j} \tag{10}$$

In the optimal solution exactly $min\{n, |R|\}$ vehicles are used. As a result of a decrease in the number of vehicles, the optimal solution will increase and vice versa. Ngueveu et al. [4] have determined a second lower bound, in case that $|R| > n$ and assuming a balanced CCVRP solution. In a balanced CCVRP solution the number of customers of each combination of routes differ by at most 1. As the main purpose is to minimize the objective value, one should take the cost of each route into account. The cost for each vehicle k is calculated as follows:

$$F_k = \sum_{j=1}^{n_k} (n_k - j + 1) c_{j-1,j} \tag{11}$$

In the formula above, $c_{j-1,j}$ equals the travel time between the jth and $j-1$ customer and n_k is the total number of customers in the route. Unlike most vehicle routing problems, a reversed route has different cost. This is the reason that $(n_k - j + 1)$ which is called the coefficient of customer j, is of importance for the cost. In order to get low total cost of a route, the customers with the highest travel time $c_{j-1,j}$ should be placed at the end of a route. Their travel time will be multiplied with a lower coefficient $(n_k - j + 1)$, as j will be higher, resulting to lower total cost of the route. Ngueveu et al. [4] obtained the second bound by using this characteristic. First, the $|R|$ customers having the lowest travel time from the depot, will be visited first in the route. The remaining $n - |R|$ customers will be visited according to their travel time, this concern travel times between customers. Let c'_e be the eth shortest travel time between the depot and a customer and c''_e the eth shortest travel time between customers. The lower bound is as follows:

$$\begin{aligned} LB_2 = &\sum_{e=1}^{|R|} \left(\left\lceil \frac{|R| + n - (n \mathrm{mod} |R|)}{|R|} \right\rceil \right) c'_e \\ &+ \sum_{e=1}^{n-|R|} \left(\left\lceil \frac{|n - (n \mathrm{mod} |R|)}{|R|} \right\rceil \right) c''_e \end{aligned} \tag{12}$$

2.3 Memetic heuristic

The memetic algorithms (MAs) of Ngueveu et al. [4] are the only available heuristics for the CCVRP [5]. The algorithm starts with a solution called the population, consisting of σ chromosomes. Let n be the number of customers, the chromosomes are a sequence (permutation) of n customers without route delimiters. The chromosomes can be interpreted as the order in which exactly one vehicle visits all the customers. These chromosomes are used to create routes, multiple routes can be extracted from the same chromosomes. The cost associated from the best solution, is referred to as the cost of the chromosome.

The algorithm exists of a predefined number of iterations. In each iteration two parents are selected and 8 children ($x = 8$) are created with crossover. The children are created using the cutting points of the OX. The chromosomes with the lowest cost are picked as parents. Unlike the traditional CVRP, a reversed route for the CCVRP has a different cost. This is why two MA implementations are considered: MA$_1$ where OX is used with zero, one or two reversed parents and MA$_2$ that does not make use of the reversed route. In general the MA$_1$ generates better solutions compared to the MA$_2$. A disadvantage of the MA$_1$ is that it is more time consuming.

The child with the lowest cost C_{best} is picked with a probability P_{ls}. Local search will be applied on the child and the trips will be connected in such a way that a chromosome is created. The three methods being used to create the chromosome are: 2-opt, relocation of one or two customers to a different route and exchange of two customers between two routes. As the reversed routes can provide a better solution, this is also checked. Finally, a chromosome belonging to the 50% worst individuals is being replaced by the child chromosome.

The process starts with an initial population of possible solutions, where each solution in the population is called a chromosome. The first solution for the initialization of the population is obtained by performing the nearest neighbor heuristic. Where each customer represents a node in the graph. The second solution is obtained by applying local search on the previous solution. Chromosomes are created by concatenating the routes of the solutions and by random permutation of the customers. Finally, the chromosomes are inserted in the population.

The genetic algorithm is based on mimicking an evolutionary process. Over time the chromosomes evolve iteration by iteration, which causes the population to also evolve in following generations. In each iteration every chromosome in the given population is evaluated by their fitness. The probability of being selected is higher the fitter the chromosome is, because a fitter chromosome enhances the evolution. The fitness is determined by applying an optimal splitting procedure based on the solution of a shortest path problem. The cost corresponding to this best extractable solution represents the quality of the routing scheme, the fitness of a chromosome.

As mentioned before, the memetic algorithms (MAs) of Ngueveu et al. [4] are the only available heuristics for the CCVRP [5]. The algorithm that will be implemented in this paper is the ALNS heuristic by Ribeiro et al [5]. The solution approach will be discussed in the next section.

3 Solution Approach

In this thesis the adaptive large neighborhood search heuristic (ALNS) from Ribeiro et al. [5] is implemented. The aim of the heuristic is to minimize the arrival times t_i^k of the customers, with t_i^k the arrival time of vehicle k at customer i. Given a solution s, at each iteration γ customers are being removed by one of the seven remove heuristics and then reinserted by one of the two insertion heuristics. As it is possible that a customer cannot be inserted because of the capacity constraint, a solution could be infeasible. Allowing infeasible solutions during the process decreases the chance of getting stuck in a local optimum. As a result the overall search is being improved. The customers in the infeasible routes get stored in the U-bank and the solution value $v(s)$ is being punished with the user-defined penalty λ. In the next iteration these customers together with the newly removed customers are reinserted by the insertion heuristics. The objective function is defined as the sum of the arrival times with a penalization depending on the number of infeasible customers;
$v(s) = \sum_{k \in R} \sum_{i \in V'} t_i^k + \lambda |U - bank|$

In order to get a better solution, customers will be removed by a removal heuristic and inserted by an insertion heuristic. The three insertion heuristics are as follows;

- The basic greedy insertion heusristic BGH 3.1.1
- The deep greedy insertion heuristic DGH 3.1.2
- The regret-k insertion heuristic RKH 3.1.3

The seven removal heuristics are as follows;

- The Shaw removal heuristic based on arrival times SHR1 3.2.1
- The Shaw removal heuristic based on distances SHR2 3.2.2
- The random removal heuristic RRH 3.2.4
- The worst removal heuristic WRH 3.2.3
- The cluster removal heuristic CRH 3.2.5
- The neighbor graph removal heuristic NGH 3.2.6
- The request graph removal heuristic RGH 3.2.7

To decide which insertion or removal heuristic are used, probabilities of being chosen are assigned to the heuristics. The probabilties are calculated with the help of scores and weights, the scores are based on the performance of the heuristic. The ALNS consist of a number of segments, after every segment consisting of ϕ iterations the scores of the heuristics are updated. After an iteration when two heuristic are chosen (an insertion and a removal heuristic), their scores are increased depending on their performance. When the heuristics obtain a better solution than the best solution, their values increase with 50. If the obtained solution is better than the previous one, the increase value equals 20. In case the obtained solution s' is worse than the previous solution s, the score will increase with 5 with a certain probability. The probability can be calculated with the following formula; $P(\text{acceptance}) = e^{-(v(s')-v(s))/T}$
With T being the current temperature and the starting value being equal to the objective value of the initial solution without the penalization. We update the temperature in each iteration by multiplying it with a cooling rate c, with $0 < c < 1$. The weights are calculated with the help of the scores, the weights are calculated with the following formula;
$w_{i,j+1} = \begin{cases} w_{ij} & \text{if } o_{ij} = 0 \\ (1-\eta)w_{ij} + \eta\pi_{ij}/o_{ij} & \text{if } o_{ij} \neq 0 \end{cases}$
The reaction factor η reflects how quickly the weights react to a change in the effectiveness of the heuristics. The weight w_{ij} is the weight of heuristic i in segment j and o_{ij} is the number of times heuristic i is chosen in segment j. The scores π_{ij} of heuristic i in segment j are reset after each segment.

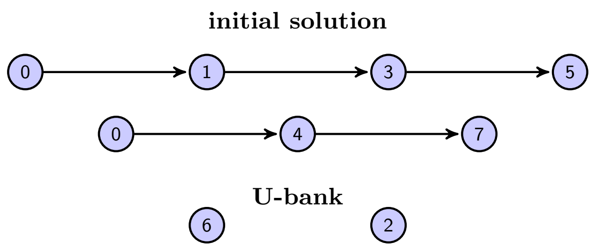
The weights are used to calculate the probability of a heuristic being chosen. Let H be the set of all insertion heuristics and H' the set of all removal heuristics. The probability of heuristic i being chosen in segment j can be calculated as follows;

$P($ insertion heuristic i chosen in segment $j) = \frac{w_{ij}}{\sum_{i \in H} w_{ij}}$

$P($ removal heuristic i chosen in segment $j) = \frac{w_{ij}}{\sum_{i \in H'} w_{ij}}$

In order to get a better view of how the heuristics work, consider the following CCVRP. The CCVRP is defined on an undirected graph $G = (V, E)$ where $V = \{0, 1, \ldots, n-1, n\}$ is the node set with $n = 7$ the number of customers that should be visited. Node 0 corresponds to the depot and $V' = V\backslash\{0\}$ is the set of customers. The set $E\{(i,j) : i, j \in V, i < j\}$ is the edge set, with $c_{ij} = c_{ji}$ the travel time associated to each edge $(i,j) \in E$. It is assumed that the travel times are symmetric and satisfy the triangle inequality and a route is defined as a circuit that starts and ends at the depot. R is the set of 2 identical vehicles with vehicle capacity Q and each customer $i \in V$ has a demand q_i. The vehicle capacity and demand will be left to one side for now, as it does not have an added value concerning the clarification of the heuristics. The capacity and demand are only relevant concerning the insertion heuristics and the same rule is applied for all three heuristics. When the insertion of a customer exceeds the vehicle capacity, the customer will be placed in the U-bank.

Figure 1: Current situation



As is displayed in the Figure 1, customer 6 and 2 are in the U-bank. Using the equation mention before, the objective value can be calculated as follows; $v(s) = (3 \times c_{0,1} + 2 \times c_{1,3} + c_{3,5}) + (2 \times c_{0,4} + c_{4,7}) + \lambda \times 2$

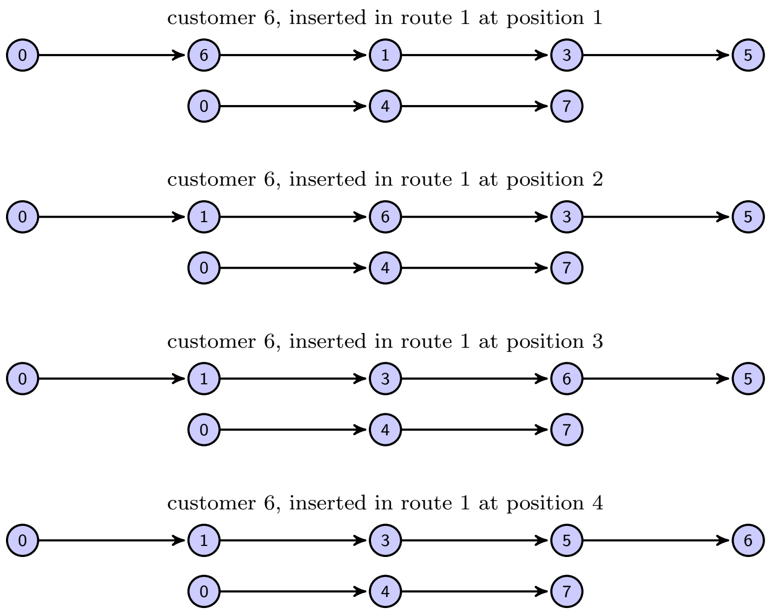
3.1 Insertion heuristics

3.1.1 The basic greedy heuristic

Two of the three insertion heuristics are greedy heuristics, the customers are inserted in the routes that increase the objective function value the least. Let $\triangle v_{ik}$ correspond to this increase, by inserting customer i in route k in the position which leads to the least increase in the given solution s. The basic greedy heuristic determines the best insertion for the first customer in the U-bank. First, the least cost increase $\triangle v_{ik}$ for every route k by inserting the customer to the particular route are determined. If a customer cannot be inserted in a route because of the demand constraint, $\triangle v_{ik}$ will be set equal to infinity and the customer will be left in the U-bank. Finally, the customer will be inserted in the route with the corresponding position that results to the least solution cost increase. The minimum cost increase $f^+(i,s)$ for inserting customer i in the given solution s can be formulated as follows; $f^+(i,s) = \min_{k \in R}\{\triangle v_{ik}\}$.
The process is repeated until all the customers in the U-bank are inserted.

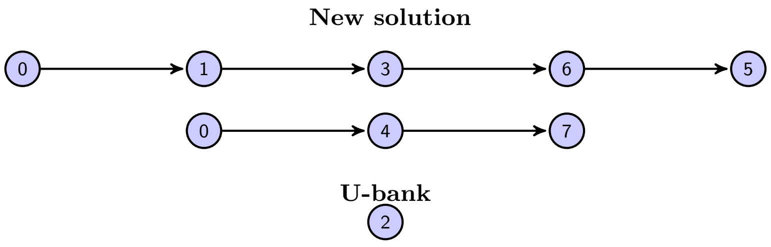
Lets Consider the situation mentioned before, customer 6 is the first customer in the U-bank. This customer will be inserted in all the routes at all possible positions and the route with the position that gives the least objective value increase will be determined. The customer will be inserted in route 1 and route 2, the process of route 1 is shown in Figure 2.

Figure 2: Basic greedy insertion heuristic


customer 6, inserted in route 1 at position 1
customer 6, inserted in route 1 at position 2
customer 6, inserted in route 1 at position 3
customer 6, inserted in route 1 at position 4

As is clear from Figure 2 above, each position has its own relevant objective value. Let the objective value increase be the smallest by inserting customer 6 at position 3. This value will be assigned to $\triangle v_{61}$, the least cost increase when inserting customer 6 in route 1. The same procedure is followed for route 2. In case the cost of route 2 is higher than route 1, customer 6 will be inserted in route 1. Figure 3 shows the new obtained solution s. The same procedure will follow for customer 2.

Figure 3: Solution after insertion



3.1.2 The deep greedy heuristic

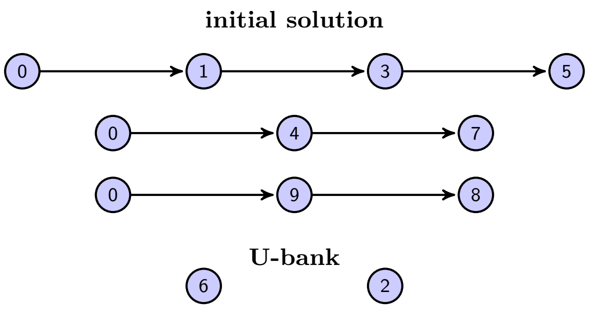
The deep greedy heuristic also inserts the customer in the route which leads to least solution cost increase. However the basic and greedy heuristic differ in choosing customers from the U-bank. The basic heuristic simply picks the first customer in the U-bank and inserts this customer. Next this customer will be removed from the U-bank and the following customer (number two) will be picked and inserted. This is not the case concerning the deep greedy heuristic. The greedy heuristic determines the least cost solution for all the customers in the U-bank. The customer that has the minimum solution cost increase when inserting it in the given solution will be inserted. Let D be the set of customers in the U-bank, the insertion can be formulated as follows; $\min_{i \in D} f^{+}(i, s)$
This process is also repeated until there are no customers left in the U-bank. Because this heuristic determines the least cost position for all of the customers instead of only the first, it has a larger computation time. The example for the basic greedy insertion is also relevant for the deep greedy heuristic. The only difference is that the procedure described is followed at the same time for both customer 6 as customer 2. The customer with the least cost increase taken both route 1 and 2 into account will be inserted in the given solution. And the steps will be performed again for the remaining customers in the U-bank, in this case only one customer will be left.

3.1.3 The regret-k insertion heuristic

Contrary to the greedy heuristics, the regret-k insertion heuristic does not only focus on the best insertion position. In order to avoid possible myopic behaviour, it also takes the other positions in consideration. The regret-k insertion heuristic inserts the customer that has the maximum difference in cost, comparing the cost of the best route and its second to kth best route. The idea behind this is; to insert the customers first that will increase the cost the most when inserting it later in the process. First, the least insertion cost of all the routes for all the elements in D are determined, like the deep greedy heuristic. Next these solution value costs are sorted in a increasing order. The sum of the differences between the best (first element) and the second to kth best inserting route (the kth element) are calculated. These differences are called the regret values. Finally, the customer with the largest difference is inserted in the given solution. In case of a tie, the customer with the least cost insertion is picked. Let $w_{ik} \in R$ correspond to the index of the route for which customer i has the kth lowest cost. The customer that is inserted can be formulated as follows; $\max_{i \in D}\{\sum_{j=2}^{k}(\triangle v_{iw_{ij}} - \triangle v_{iw_{i1}})$
This heuristic is also used when creating the initial solution, with k equal to 3.

To get a better idea of how the heuristic works, consider the example used for the basic greedy heuristic with three vehicles and 9 customers. Figure 4 shows the current solution s. In this case, like the deep greedy heuristic the best insertion position will be determined for both customer 6 and 2. As the process of customer 2 and 6 is the same, we will only take customer 6 as an example. The least cost increase of routes 1,2 and 3 for customer 6 will be determined. The same steps as in the example of the basic greedy heuristic can be followed. Let the cost increase in route 3 be the smallest and in route 2 be the largest for customer 6. The least cost increase for inserting customer 6 in route 3 equals; $\triangle v_{6w_{61}}$
The differences between the cost increase of route 1, 2 and 3 is calculated with the formula;
$\sum_{j=2}^{3}(\triangle v_{6w_{6j}} - \triangle v_{6w_{61}})$
The same steps are followed for customer 2, the customer that has the highest difference is inserted.

Figure 4: Current situation



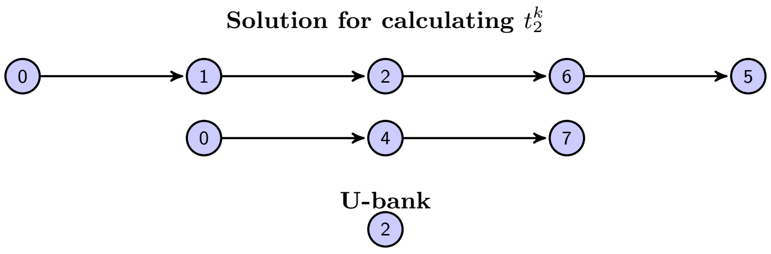
3.2 Removal heuristics

3.2.1 The Shaw removal heuristic based on arrival times

Two Shaw removal heuristics are been used, the SRH1 and SRH2 ([8],[11]). Both heuristics are based on the similarity of two customers, as it is expected that it is easier to reshuffle similar customers and thereby create better solutions. The SRH1 heuristic is based on the arrival times, the customers that are the most similar concerning their arrival times are removed. First a customer is randomly picked from the U-bank. If the U-bank is empty, a random customer is removed from the given solution and put in the U-bank. After a customer is picked, the arrival times are compared using the relatedness measure $R(i, j)$. This degree of similarity can be calculated with the following formula; $R_{SHR1}(i, j) = |t_i^k - t_j^k|$
This value is calculated between the chosen customer and all the other customers in the given solution. Next the values are being sorted, a lower value corresponds to more similarity between the customers. With the help of a user-defined parameter δ that brings randomness in the process, a customer is picked to be removed. This process is repeated until γ customers are removed from the

given solution and put in the U-bank. To get a better view of the process a graphic example is used, let figure 3 be the current solution. As there is only one customer in the U-bank (customer 2), this customer will be chosen from the U-bank. The second step is to calculate the relatedness measures between customer 2 and all the customer in the solution (customers $1, 3, 4, 5, 6$ and 7). Figure 5 shows the solution for calculating the relatedness $R_{SHR1}(2,3)$ measure for customer 3 and customer 2.

Figure 5: Relatedness measure $R_{SHR1}(2,3)$ for the SRH1



With the help of Figure 3 and Figure 5, the relatedness measure $R_{SHR1}(2,3)$ can be calculated as follows; $t_2^k = 2 \times c_{0,1} + c_{1,2}$, $t_3^k = 2 \times c_{0,1} + c_{1,3}$ and $R_{SHR1}(2,3) = |t_2^k - t_3^k = |c_{1,2} - c_{1,3}|$
This is done for all the customers in the solution. After that, all the customers are sorted according to $index_i < index_j$ if $R_{SHR1}(2,i) < R_{SHR1}(2,j)$, with $index_i$ the index of customer i in the sorting list L. Finally a customer is picked from the list with the help of y and $\delta \geq 1$, with y a random number from $[0,1]$. The index of the customer in list L that is going to be removed is calculated as follows: $Remove_{index} = \lfloor y^{\delta}|L| \rfloor$

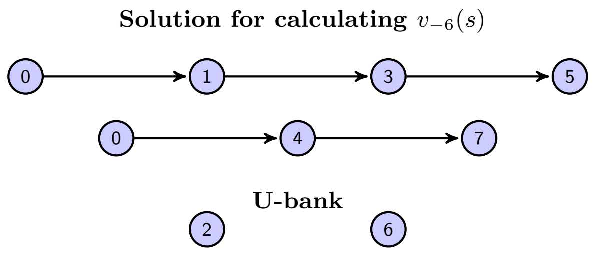
3.2.2 The Shaw removal heuristic based on distances

The main difference between the SHR1 and the SHR2 heuristic is that the SHR2 is based on the distance between customers instead of the arrival times. The other steps are exactly the same, but instead of calculating the difference between the arrival times the geographical distance is used. Let (x_1, y_1) and (x_2, y_2) be the coordinates of the two relevant customers. The degree of similarity is in this case calculated with the following formula; $R_{SHR2}(i,j) = \sqrt[2]{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
In order to fasten the process, a matrix B can be constructed. Where the element b_{ij} corresponds to the relatedness measure, the distance between customer i and j. The example used for the SRH1 can be used with the relatedness measure explained before.

3.2.3 The worst removal heuristic

The worst removal heuristic sorts the customers according to their cost and removes the customers with the highest costs. The idea is to calculate the cost of the customers in the given solution. In order to determine this, the solution value costs are calculated with customer i in the solution and with customer i removed. Where the solution value cost equals the current solution cost $v(s)$ with customer i and $v_{-i}(s)$ without customer i in the solution. Next, the difference between those two values is calculated for all the customers in the solution. The following formula is used for this; $Cost^{-}(i,s) = v(s) - v_{-i}(s)$
This heuristic also sorts the values and removes a customer with the help of the parameter ρ for randomization. The higher the cost, the greater the probability one can insert them in a better position. This process is also repeated with the given solution without the removed customer, until γ customers are removed.
In order to get a better idea of the calculation of the costs, Figure 6 gives a graphic example for calculating $Cost^{-}(6,s)$. The current solution s is displayed in Figure 3. When calculating $v_{-6}(s)$, one should also take into account that there are now 2 customers in the U-bank for the penalization. The sorting is exactly the same as the SRH1 and SRH2, but instead of δ the user-defined parameter ρ, with $\rho \geq 1$ is used.

Figure 6: Worst removal heuristic cost calculation



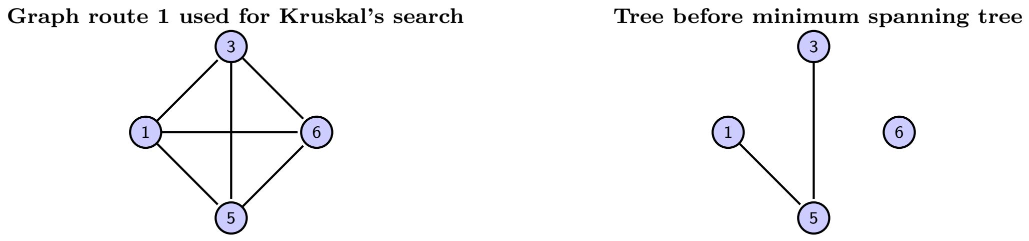
3.2.4 The random removal heuristic

To have some randomness in the removal process, the random removal heuristic is used. This heuristic removes randomly customers from the given solution s until there are γ customers in the U-bank. Although the removed set of customers is not that efficiënt, the heuristic is very useful concerning diversification.

3.2.5 The cluster removal heuristic

] The cluster removal heuristic removes customers within a route that belong to different clusters. As putting customers from the same cluster in the same route leads to smaller arrival times. First a random customer from the U-bank is picked and a route from the given solution is chosen based on this customer. The route with the customer that is geographically the most similar with the U-bank customer is chosen. If the U-bank is empty, a route from the solution is chosen randomly. Kruskal's search [10] is used to divide the customers from the route in two clusters. The algorithm is modified in such a way, that it stops when two connected components are found. The same steps are followed, just before the last step (when there are two clusters left) before a minimum spanning tree is created the algorithm stops. One of the two clusters is chosen randomly and the customers from this cluster are removed from the solution and put in the U-bank. If the number of removed customers is less than γ, a new route is picked with a random customer from the U-bank and the same process follows. This route should be different from the previous chosen route. Let in the situation of Figure 3, customer 2 from the U-bank have the least geographical distance with customer 5. As a result, route 1 is selected and clusters are created with Kruskal's algorithm. Figure 7 shows how the clusters are created.

Figure 7: Creating 2 clusters


Graph route 1 used for Kruskal's search
3
1
6
5
Tree before minimum spanning tree
3
1
6
5

Let $d_{i,j}$ be the distance between customer i and customer j. Concerning Figure 7, $d_{1,5} < d_{5,3} < d_{3,6} < d_{5,6} < d_{1,3} < d_{1,6}$. The right graph of figure 7 shows the 2 clusters created with Kruskal's search. One cluster consists of customers 1,5 and 3 and the other culster consist of customer 6. The second step is to remove all the customers from one of the two randomly picked clusters. The next step is to choose another route (that is different from the previous) and repeat the process until γ customers are removed.

3.2.6 The neighbor graph removal heuristic

The neighbor graph removal heuristic removes customers based on their historic information. The value of edge (i, j) in the neighbor graph represents the best objective value found, given that customer j is directly served after customer i. All the edges are initialized to infinity, at each iteration the values will be updated when a smaller objective value is found. Customers with a high edge value in the given solution, have a higher probability of being inserted in a better position in the solution. All the customers in the given solution will be sorted, based on their values in the neighbor graph. A customer is selected to be removed with the help of a parameter $\phi \geq 1$, that introduces randomness. As γ customers should be removed, the remaining customers should be sorted again according to the new solution.

3.2.7 The request graph removal

The request graph removal is also based on historical information. The b best objective values with their solutions are stored. The value of edge (i, j) in the request graph represents the number of times that customer j is directly served after customer i, in the b best solutions. In this case a lower edge value of a customer means that the customer could probably be inserted in a better position in the solution. The rest of the procedure is the same as the neighbor graph heuristic. The values are also sorted and γ customers are removed and put in the U-bank.

Both graph heuristics can be explained by the example of the SRH1, the values of the graphs are used as relatedness measure. The parameter ϕ randomises the selection of customers.

4 Results

As the purpose of this thesis is to re-implement and test the ALNS heuristic by Ribeiro et al [5] on a varied set of benchmark problems, the same data is used. These benchmark problems consist of one depot and a homogenous fleet of vehicles.

The ALNS heuristic will be tested on the seven $50 - 199$ customer instances, first proposed by Christofides et al.[6] , followed by Nguevue et al. [4] and Ribeiro et al [5]. The data consists of the x and y coordinates of the customers (and depot) with their given demand. The number of homogenous vehicles with their capacity, the maximum route time and drop time are also given.

The values used for the parameter values are the same as Ribeiro et al [5], the values can be found in Table 1.

Table 1: Parameter values for the ALNS

Parameter	**Meaning**	**Value**
γ	Defines the number of customers removed at each ALNS iteration	$\gamma \in [10, 40]$
ϕ	Defines the number of iterations each segment	50
η	The reaction factor of the weights	10^{-2}
λ	The penalization factor for a solution	10^4
δ	Is responsible for randomness in the removal process of the Shaw and Request graph heuristic	2
ρ	Is responsible for randomness in the removal process of the worst removal heuristic	2
ϕ	Is responsible for randomness n the removal process of the neighbor graph removal heuristic	3
κ	Indicates which regret k heuristic is chosen	3

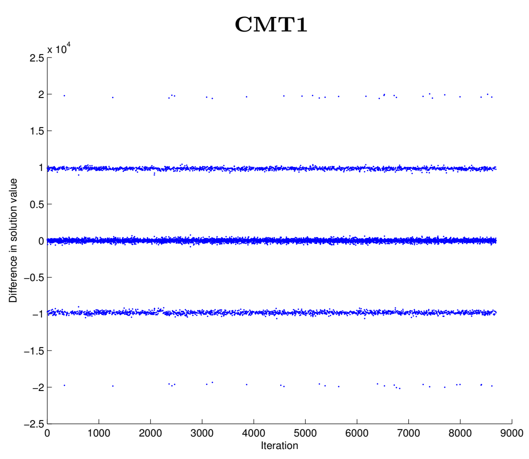
Ribeiro et al [5] repeated the process until one of the two stopping criteria were satisfied. The process stopped when the number of iterations reached 50000 or the temperature reached 10^{-2}. In our case, the ALNS heuristic was coded in Matlab and run on a laptop with Intel Core i5 3210M processor with 4 GB RAM under the Windows operating system. Due to time constraints, each instance is terminated after a running time of 24 hours. The results can be found in Table 2.

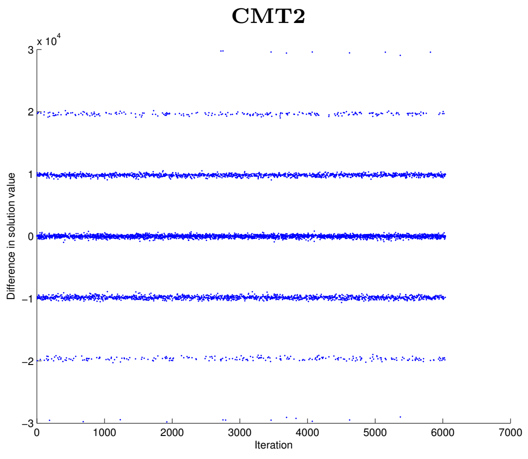
Table 2: Results for the seven instances proposed by Christofides er al. [6]

Name	**n**	**|R|**	**LB**	**Best by Ribero et al.**	**Best**	**Runtime (sec)**	**T**
CMT1	50	5	1873.91	2230.35	2291.37	9.94	303.68
CMT2	75	10	1861.56	2391.63	2624.93	14.30	663.61
CMT3	100	8	2947.74	4045.42	4257.06	17.98	1614.36
CMT4	150	12	3561.67	4987.52	5297.87	151.31	5826.5
CMT5	199	17	4804.2	5838.32	6300.94	59.34	5148.4
CMT11	120	7	6119.66	7315.87	7893.08	26.17	4331.42
CMT12	100	10	2885.48	3558.92	3706.42	16.79	1628.17
Average	113.43	9.86	3436.32	4338.29	4624.524	42.26	2788.02

Column $1, 2$ and 3 in Table 2 show the characteristics of the instances. The number of customers is denoted with n and the number of homogeneous vehicles is denoted with $|R|$. The lower bound (LB) in column 4 is the largest lower bound found with the help of formula (10) and (12)[5]. Column 7 shows the runtime in seconds of one iteration and the last column shows the temperature after 24 hours. The best solution of every instance is feasible, as the U-bank is empty and the demand capacity constraint is not violated. Column 6 shows the best solution value obtained and column 5 the values found by Ribeiro et al. [5]. In order to get a better view of the change in the solution values, figure 8 shows the values for CMT1 and CMT2 for each iteration over the 24 hour period.

Figure 8: Difference in solution value each iteration





As is clear of the figures displayed above, the solution values of each consecutive iteration differ between the -20000 and 20000. This is not that surprising, as we are dealing with a large neighborhood search and not feasible solutions are also accepted with a certain probability. In case the routes do not change much after an iteration, the difference between the solutions value will not be that large. This can be seen in figure 8, as there is a high concentration around the zero line (y-axis). In case an infeasible solution is accepted, the solution value will increase with the number of customers in the U-bank multiplied with the penalization factor. As is also clear from figure 8, there is also a high concentration around the -10000, 10000 , -20000 and 20000. This can be explained with the acceptance of solutions that have 1 or 2 customers in the U-bank, which results to an increase of the solution values with 10000 and 20000.

As is mentioned in section 3, there are three insertion heuristics and seven removal heuristics. Figure 9 shows the weights of removal and insertion heurisics w_{ij}, also noted in section 3. The weights are calculated with the following formula;

$$w_{i,j+1} = \begin{cases} w_{ij} & \text{if } o_{ij} = 0 \\ (1-\eta)w_{ij} + \eta\pi_{ij}/o_{ij} & \text{if } o_{ij} \neq 0 \end{cases}$$

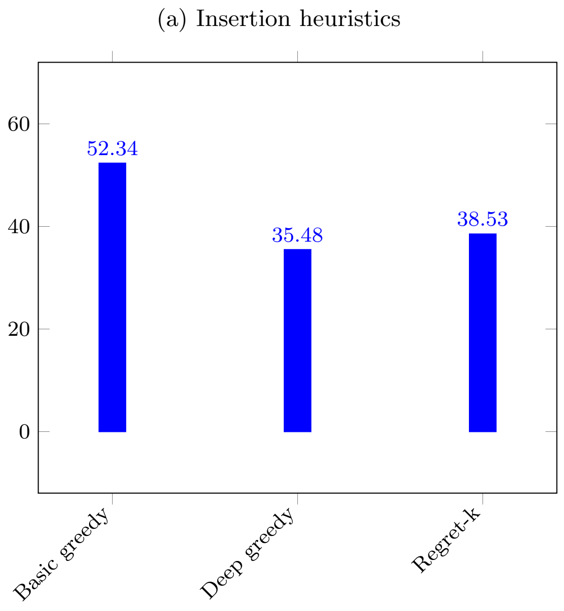
The weight w_{ij} is the weight of heuristic i in segment j and o_{ij} is the number of times heuristic i is chosen in segment j. The scores of heuristic i in segment j are denoted with π_{ij} and the average score of heuristic i is π_{ij}/o_{ij}. The reaction factor η reflects how quickly the weights react to a change in the effectiveness of the heuristics, as the average score is multiplied by this factor. The weights are used to calculate the probability a heuristic is chosen. Let H be the set of all insertion heuristics and H' the set of all removal heuristics. The probability of heuristic i being chosen in segment j can be calculated as follows;

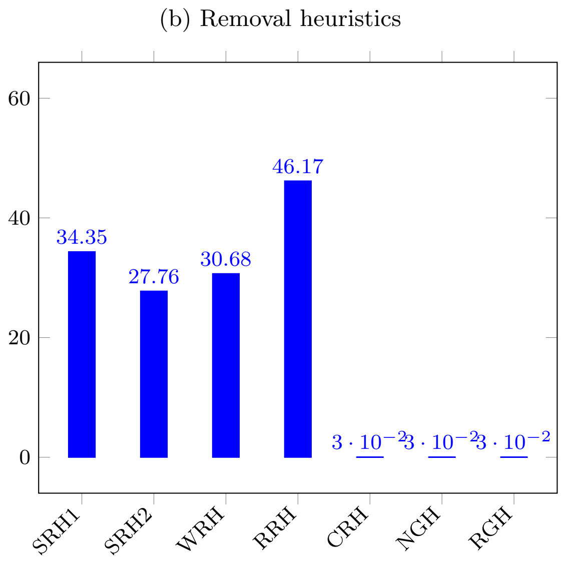
$$P(\text{ insertion heuristic } i \text{ chosen in segment } j) = \frac{w_{ij}}{\sum_{i\in H} w_{ij}}$$

$$P(\text{ removal heuristic } i \text{ chosen in segment } j) = \frac{w_{ij}}{\sum_{i\in H'} w_{ij}}$$

The weights in figure‘9 are averages of the seven instances.

Figure 9: Weights of the heuristics





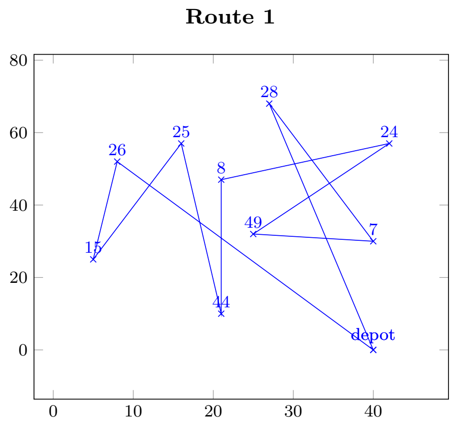
A higher weight of a heuristic corresponds with a higher probability of being chosen during the process. Moreover, a higher weight also corresponds with higher scores. In case a heuristic obtains a better solution, its weight will increase. To sum up, heuristics with higher weights perform better. Figure 9 shows that *the basic greedy heuristic* has the highest weight of all the insertion heuristics and *the random removal heuristic* has the highest weight of the removal heuristics. Randomization is a factor that both heuristics have in common. The basic greedy heuristic is the heuristic with the most randomness concerning the insertion of customers, compared to the other two insertion heuristics. This result can be explained, by the fact that large neighborhood search aims to escape local minima by trying enough random neighborhoods [12].

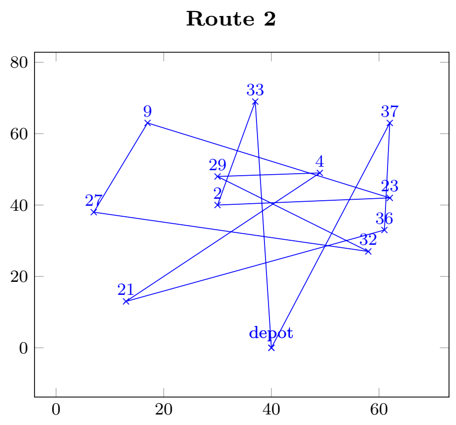
The weights of the cluster removal, neighbor graph removal and request graph removal are almost similar. All three heuristics perform poorly concerning all the instances, compared to the other removal heuristics. As both the neighbor removal and the request graph removal heuristic are based on historical data, we can conclude that this is not a strong factor to depend the removal of customers on.

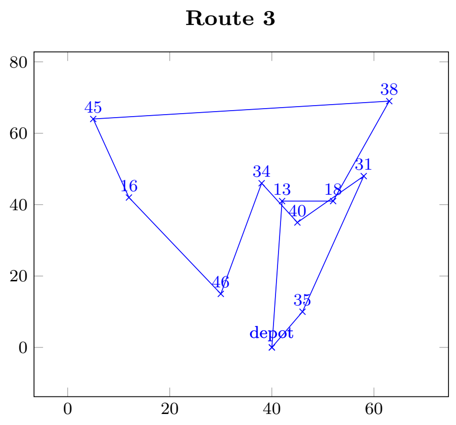
However one should keep in mind that the weights are averages of the seven data instances. The performance of the heuristics depend on the dataset. For example, the regret-k insertion heuristic outperforms the other insertion heuristics concerning the CMT2 dataset. In order to have a large search and escape local minima, one should make use of all the heuristics and keep their performance in mind (with help of their weights).

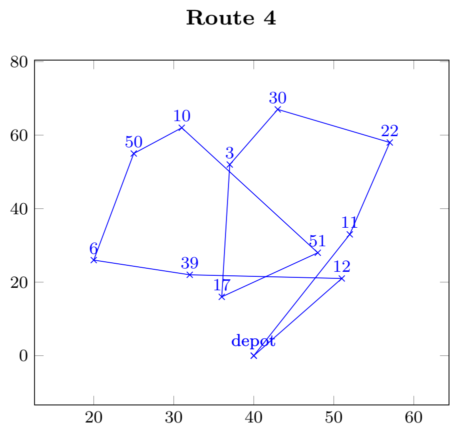
In order to get a better view of the routes, figure 10 shows the best routes obtained for the first instance (CMT1). The numbers in the figures denote the number of the customers in the datasets proposed by Christofides et al. [6].

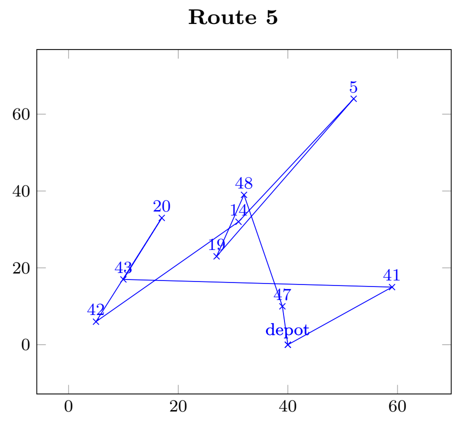
Figure 10: Best route CMT1











5 Conclusion

In this work we have implemented *an adaptive large neighborhood heuristic for the cumulative capacitated vehicle routing problem* by Ribetiro et al. [5]. The CCVRP plays an important role concerning humanitarian aid, as in such situations the arrival times at those in need are of importance. The objective is to minimize the arrival times in order to minimize suffering and loss of life. The large neighbourhood search consist of three insertion and seven removal heuristics, the heuristics are picked based on their performance during the process. Although the performance of the heuristics are dependent of the data instances, there are some heuristics that perform poorly concerning all the instances. These are the heuristics that are based on historical data; the request graph removal and the neighbor graph removal heuristic. The heuristics that insert and remove customers with the most randomization perform the best; the basic greedy insertion heuristic and the random removal heuristic. In order to escape local minima to find better solutions, one should use different heuristics with different characteristics. Concerning further research one can consider replacing the removal heuristics based on historical data, with heuristics that are based on the given solution. The removal heuristics should remove customers based on the given solution with distance and time being an important factor, as the Shaw removal heuristic and the worst removal heuristics perform better. Randomization is also an important factor, most removal heuristics have parameters to avoid determinism. One can also consider adding parameters for randomization concerning the insertion of customers.

6 References

[1] Golden B, Raghavan S, Wasil E. *The vehicle routing problem: latest advances and new challenges* Dordrecht: Springer; 2008.

[2] Laporte G. *The vehicle routing problem: an overview of exact and approximate algorithms.* European Journal Operational Research 1992;59:34558.

[3] Campbell AM, Vandenbussche D, Hermann W. *Routing for relief efforts.* Transportation Science 2008;42:12745.

[4] Ngueveu SU, Prins C, Wolfler-Calvo R. *An effective memetic algorithm for the cumulative capacitated vehicle routing problem.* Computers & Operations Research 2010;37:187785.

[5] Ribeiro and Laporte, 2012, *'An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem'*, Computers & Operations Research, Vol. 39, pp. 728735.

[6] Christofides N, Mingozzi A, Toth P. *The vehicle routing problem.* In: Christofides N, Mingozzi A, Toth P, Sandi C, editors. Combinatorial optimization. Chichester: Wiley; 1979. p. 31538.

[7] Prins C. *A simple and effective evolutionary algorithm for the vehicle routing problem.* Computers & Operations Research 2004;31:19852002.

[8] Ropke S,Pisinger D. *An adaptive large neighborhood search heuristic for the pick up and delivery problem with timewindows.* TransportationScience 2006;40:45572.

[9] Azi N,Gendreau M, Potvin J.-Y. *An adaptive large neighborhood search for a vehicle routing problem with multiple trips.* Technical Report 2010-08,CIRRELT, Montre al, Available at </https://www.cirrelt.ca/DocumentsTravail/ CIRRELT-2010-08.pdf >; 2010.

[10] Potvin J-Y, Rousseau J-M. *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows.* European Journal of Operational Research 1993;66:33140.

[11] Shaw P. *A new local search algorithm providing high quality solutions to vehicl routing problems.* Technical Report,University of Strathclyde,Glasgow;1997.

[12] Blum C., Roli A. *Metaheuristics in Combinatorial Optimization:Overview and Conceptual Comparison* ACM Comput. Surveys 35 (3) (2003) 268308.