POLITECNICO DI TORINO

Collegio di Ingegneria Gestionale

Corso di Laurea Magistrale

in Engineering & Management

Tesi di Laurea Magistrale

Solving a food delivery problem with a Vehicle Routing Problem-based approach

Relatore

Fabio Salassa

Candidato

Leonardo Bruno

APRIL 2019

Anno Accademico 2018-2019

Table of content

List of figures

List of tables

Acknowledgements

I would like to thank the thesis supervisor, Fabio Salassa, for giving me the right material for research and for the control of the structure and the content of this work.

Then there are the thanks for my family and for my friends for supporting given to me every day in order to continue this road and against the difficulties I have encountered along the way and providing the best environment as possible for working.

1 ABSTRACT

Nowadays urban transportation is a strategic domain for distribution companies. In academic literature, this problem is categorised as a Vehicle Routing Problem, a popular research stream that has undergone significant theoretical advances but has remained far from practice implementations.

In recent decades there has been a growing use of software packages based on operational research techniques and mathematical programming for the efficient management of goods in distribution systems.

The large number of real applications has amply demonstrated that the use of software for planning distribution processes generates substantial savings (generally in a variable proportion from 5% to 20%) in overall transport costs.

It is easy to see how the impact of this saving on the economic system is significant, since the transport processes concern all stages of the production of goods and the relative costs represent a significant component (around 10% - 20%) of the final cost.

The success in the use of operational research techniques is due not only to hardware and software development in the field of information technology and the growing integration of information systems in the production process and in the commercial one but above all to the development of new models trying to take considering all the characteristics of real problems and the conception of new algorithms that allow us to find good solutions in acceptable calculation times.

In recent years, the focus of local search has shifted to more complicated metaheuristics to increase the power of earlier techniques.

In this thesis we want to specifically investigate the Vehicle Routing Problem, the main tool for modelling the reality of logistics and the transport of goods, through its application in a real use case implemented at the company Sotral, located in Turin, where the writer has had the opportunity to complete the training period.

In particular, Clarke & Wright's Savings and Time Windowed Nearest Neighbour techniques will be used, two approaches that guarantee excellent solutions in relatively short computational times.

2 BACKGROUND

2.1 THE COMPANY

The company where the author has accomplished the internship is called Sotral. It is a firm located in Turin with about 25 years of history. It has about 15 administration resources, 200 drivers and 200 vehicles actives in the Italian territory. It gives transportation logistics services to customers. Its core business refers to deliver meals for all levels of schools (from primary to universities), home delivery, and private companies.

2.2 SOTRAL'S MISSION

The mission of Sotral is to increase its market pool for restoration services, integrating the aspects of quality, sustainability of the environment social responsibility. To reach this goal, the company puts its efforts in minimising the kilometres travelled, so the CO_2 emissions in the atmosphere, but in the meantime using all the available resources in the same way and respecting the constraints forecasted (imposed by the customers, or by the bid's documents). Furthermore, in the future, the company wishes to implement more platforms, tools and technology settings to improve their performances in data management and doing Enterprise Resource Planning (ERP).

Going back again, Sotral creates new travel itineraries for its delivery food services and updates the old ones. These tours generate new costs for the firm and, to obtain

earnings from the delivery service, the payments to the client should be more than the expenses.

So, the most important source of revenue of the firm are the bills that Sotral emits to the customers, but before the formulation of the service offer, other tasks have to be undergone, and these steps are explained in the following chapter.

2.3 THE COMPANY MAIN TASK

Usually, Sotral has to accomplish the following steps when it performs an estimate of the costs:

Step 1. The company receives information about a bid (or a single travel request) by the customer.

Step 2. The request is processed by the Logistics attendant, who has to fulfil the demand searching the nodes on Google Maps, and creating the routes for that bid.

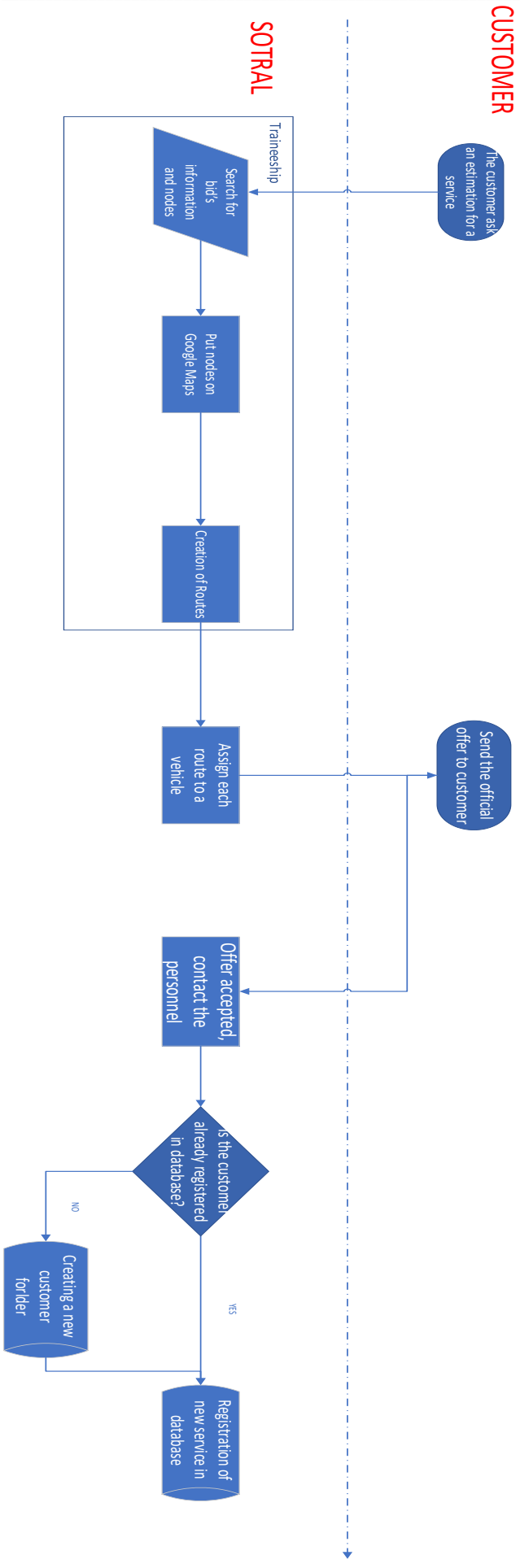
Step 3. The Logistics attendant, in collaboration with the Region's responsible for that bid, assigns for each route a vehicle, and proceed on calculation of costs.

Step 4. The quotation offer is created and transmitted to the customer.

Step 5. When the offer is accepted, the company contacts the personnel to inform about new task to perform.

Step 6. Then the service is recorded in the database, with date, name of customers, number of vehicles used, name of personnel involved in the service.

In the figure below the activities where the writer has been involved during the training period are depicted.



Legend:

Put nodes on Google Maps — Operational tasks performed by a Sotral's employee

Registration of new service in database — Action in which Sotral's database is needed

The customer is already registered in database? — Checkpoint with two possibilities

Search for bid's information and nodes — Tasks that don't give added value to the final price

Send the official offer to customer — Tasks in which the customer is involved, or in which the firm has to wait a customer's answer

Figure 1 Steps of Sotral's activity with legend

3 THE COMPANY PROBLEM AND SOLUTIONS APPROACH

3.1 THE PROBLEM'S CONTEXT

3.1.1 A FOOD-DELIVERY PROBLEM

The real problem that has been faced in Sotral concerns the optimization and organization of the routes from a food production site to numerous delivery points, to minimize the time spent in each destination and, at the same time, reduce the number of vehicles used.

The defined scenario is as follows:

- The Cooking site (depot): the place where the kitchen is placed, and the meals are prepared;
- Delivery Points (customers): these are the places of destination, therefore the school, public or company canteens where meals are consumed;
- Lunch hour: the moment when the employees at the point of delivery should serve meals.
- Delivery Time Interval: the time window within which a delivery must be completed to meet the legislative constraints imposed by customer.

The problem is a typical *Capacitated Vehicle Routing Problem with Time Windows (CVRPTW)* having the constraints depicted in the next sub-sections.

Here is a list of assumptions made for this thesis:

1- There is only one depot;

2- There is no working time limitation for the depot;

3- Each customer is visited exactly once, meeting the whole demand;

4- Each customer has a specific service time window outside which service is not acceptable;

5- Vehicles are all identical, with the same capacity;

6- Drivers are all identical, with the same driving time and working time limitations;

7- Vehicle capacity is at least as high as the largest customer demand.

3.1.2 VRP – IN THE LITERATURE

The problem known as VRP - Vehicle Routing Problem - was proposed in 1959 by Dantzig and Ramser. In the literature, VRP is the generic name used to refer to a whole class of problems inherent in the visit of "customers" by "vehicles". This type of problem has considerable practical implications both in the case of the effective transport of goods, and in many other sectors (collection of mail from mailboxes, school bus service, ...). As we can see in the picture, we have a situation in which there's a depot, a central node (in Sotral's case, it is called cooking site) in which all the quantities of the meals, or products are stored, surrounded by a certain number of nodes, that are delivery point (i.e. stores, schools), so points in which demand is verified. Our goal is finding the solution with the least number of vehicles as possible, to deliver the goods in each delivery point, and with the creation of routes that have almost the same time required to fulfil the deliveries.

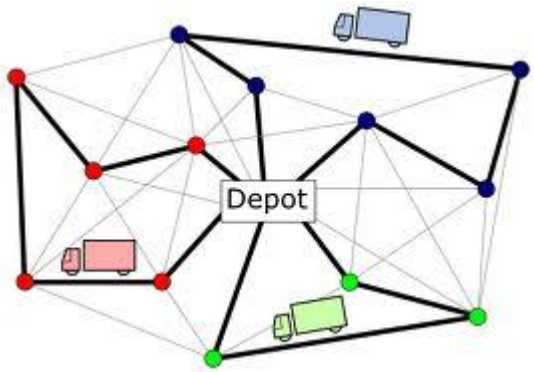


Figure 2 Basic situation of a Vehicle Routing Problem

The goal is to produce a minimum cost routing plan specified for each vehicle. The problem of vehicle scheduling may be stated as a set of customers, each with a known location and a known requirement for some commodity that has to be supplied from a single depot by delivery vehicles, subject to the following conditions and constraints:

- the demands of all customers must be met.
- each customer is served by only one vehicle.
- the capacity of the vehicles may not be violated (for each route the total demands must not exceed the capacity).

- each vehicle must start its route from the depot, and it has to finish the route in the same depot.

3.1.3 CAPACITED VRP

In the CVRP there is a fleet of vehicles with fixed capacity that must serve the requests of the customers, these are known a priori and must be satisfied by a single vehicle. The vehicles are all the same and refer to a single central warehouse (depot).

The objective of the CVRP is to minimize the fleet of vehicles and the sum of travel time, making sure that the total demand for each route does not exceed the capacity of the vehicle serving the route.

The solution is accepted if the total quantity assigned to each route does not exceed the capacity of the vehicle serving that route.

3.1.4 CVRP WITH TIME WINDOWS

An important extension of the CVRP is the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW). The goal of these problems is to minimize the fleet of vehicles and the total travel and waiting time needed to serve all customers respecting their time constraints.

The time interval associated with the client is called, in fact, time window, and is indicated with $[e_i, l_i]$. (Earliest time, latest time) The service of each client must start in a moment t belonging to the time window; in the case of early arrival at the node n_i, the vehicle must wait for the moment and the first to be able to perform the service.

On the other hand, if the vehicle arrives after the latest time, the solution is not feasible. Each of the customers is also associated with a service time, s_i, which represents the duration of the time interval during which the vehicle that performs the service remains stationary at the customer's premises. Each vehicle must then begin and finish its journey respecting the time window related to the deposit from which it departs.

3.1.5 DISTANCE CONSTRAINED VRP

In DCVRP the capacity constraints of each route are replaced by maximum length or time constraints: a non-negative length t_{ij} is associated with each arc or side (i, j) and the total length of the edges of each route cannot exceed a maximum established value of total travel length T.

Another variant is the DCCVRP - Distance-Constrained CVRP - in which both constraint families are present; each route has a maximum length or travel time and, moreover, the vehicle that travels it has a limited transport capacity.

The objective of the problem then corresponds to minimizing the total length of the routes or, if the service time is included in the time costs of the arcs, their duration.

3.1.6 PICK-UP AND DELIVERING VRP

The Vehicle Routing Problem with pick-up and delivery (VRPPD) is a VRP in which customers can return some products. It is therefore necessary to take into account also the products that the customers return in order not to exceed the maximum capacity of the vehicles. This restriction makes the problems of planning more difficult and can lead to a lower exploitation of vehicle capacity, an increase in the length of the journey or a need for a greater number of vehicles.

One way to try to find a solution to these problems is to consider certain situations, for example all the delivery requests start from the deposit and all the pick-up requests must be returned to the warehouse, so there is no exchange of goods between customers.

Another problem can be circumvented if the constraint is removed that each customer must be visited only once of the vehicle.

A further simplification is to require each vehicle to deliver all products before withdrawing any type of goods.

The objective of the VRPPD is to minimize the fleet of vehicles and the sum of travel time, with the limitation that the vehicle must have enough capacity to transport the

goods to be delivered and those collected by the customers to bring them back to the warehouse.

The possible solution is that the total quantity of goods assigned to each route cannot exceed the capacity of the vehicle that supports the service on that route and in addition its capacity must be sufficient to collect the goods from the customers.

3.2 THE PROBLEM IN MATH LANGUAGE

After explaining the general view, we proceed in a deeper analysis. Now it's the time to collect all the data about a general food-delivery problem and express them in an objective mathematical construction; in doing so, the Operational Research's linear programming formulas are used.

Therefore, we will introduce in the next chapters the parameters and the variables of the model, its objective function that the author has to minimise, and the constraints subject to.

3.2.1 VEHICLE CAPACITY CONSTRAINT

For the type of product being delivered, usually hot meals with low volume, it is assumed that:

- the means have a capacity not in terms of m^3, but in terms of maximum amount of meals to be carried;
- the capacity of each vehicle being used is fixed, and the user may decide how many meals the vehicles can carry.

3.2.2 TIME WINDOW CONSTRAINTS

The time window conveys the earliest and latest times that a customer is available to be serviced.

Thus, given a central cooking site (depot) and identical vehicles, routes starting and ending at the depot are designed such that customers, who each have their own demands, are serviced at least once at the least possible cost, all the while satisfying vehicle capacity, route duration and time window constraints.

Vehicles cannot arrive late, that is, after the upper bound of the time windows, but if vehicles arrive too early, they must wait until the allowable service time. This waiting time is included in the total route length calculation.

For the sake of simplicity travel time, travel distance, and travel costs are all used interchangeably. The number of routes can either be a fixed number or be allowed to vary provided that it does not exceed some given upper bound. Customers' demand and time windows are given parameters, whereas the starting times of services are decision variables, that is, determined in the solution.

3.2.3 OBJECTIVE FUNCTION

The focus of the company is centred in minimizing the time variance among routes, in order that employees have to work almost the same number of working hours for each delivery activity. According to the fact that in the text we will use the following terminology:

Model's variables

- $ARRTIME_{ik}$ = time in which the worker comes in node i, inside the route k (if it's 0 in that route that node i isn't inserted)
- $TPART_{ik}$ = time from which the worker comes from node i, with vehicle associated in route k
- $TDEP_k$ = time of starting service of route k
- USE_k = this variable is 1 if I use the vehicle k in a route (0 otherwise)
- $CHOICE_{ik}$ = it has value 1 if the node i is inserted inside route k (0 otherwise)

Model's parameters

- $TIMEMAX_i$ = latest time in the time interval
- $TIMEMIN_i$ = earliest time in the time interval
- BIGM = a very big number, used for BigM constrain to link binary variable with discrete one. ($Choice_{ik}$ with $ARRTIME_{ik}$)

- CAP_k = maximum capacity in quantity of meals of vehicle associated in route *k*
- D_i = demand (in terms of number of meals to deliver) of node *i*.
- AVG SPEED = speed at which all vehicles travel, on average.

We can set the objective function of Sotral's problem in this way:

Objective function

$$min \sum_{k}^{v} [\sum_{i}^{n} [ARRTIME_{ik} - TPART_{ik}] - TDEP_k] * AVGSPEED$$

That stands for minimising the sum of all the distances travelled in serving all the nodes in the graph (there's a multiplication with the AvgSpeed, but it can be neglect, because it's a constant value. If we neglect that, the objective function's meaning becomes the minimum value of the sum of all the times needed to serve all nodes in the graph).

Subject to these constraints:

$$\sum_{k}^{v} ARRTIME_{ik} \leq TIMEMAX_i \qquad \forall i$$

The sum (for the vehicles) of all arrival time (only 1 value is positive) in node i should be equal or less than maximum time in the time interval of that node, for all nodes in the graph.

$$\sum_{k}^{v} ARRTIME_{ik} \geq TIMEMIN_i \qquad \forall i$$

The sum (for the vehicles) of all arrival time (only 1 value is positive) in node i should be equal or more than minimum time in the time interval of that node, for all nodes in the graph.

$$ARRTIME_{ik} \leq CHOICE_{ik} * BIGM \qquad \forall i, k$$

To have a positive arrival time for the node i, with vehicle k, the corresponding CHOICE value must be 1, multiplied by a very high number.

$$TPART_{ik} \leq CHOICE_{ik} * BIGM \qquad \forall i, k$$

This constraint has the same reasoning of the predecessor: to have a positive value in TPART in node i, with vehicle k, the corresponding CHOICE value must be 1.

$$\sum_{i}^{n} D_i * CHOICE_{ik} \leq USE_k * CAP_k \qquad \forall k$$

The sum of all the demands of the nodes served by vehicle k should be equal or less than the maximum capacity of that vehicle, if used.

$$ARRTIME_{ik} - TDEP_k - TPART_{ik} \leq 30 \qquad \forall i, k$$

For each route created, the amount of time invested should not be more than 30 minutes by law.

3.3 THE SOLUTION APPROACHES

3.3.1 INTRODUCTION: HOW TO SOLVE A PROBLEM OF OPTIMISATION?

Operational researchers have to face problems called combinatorial optimisation problems: given a set of data, they must find the best combination of variables to achieve the highest (or the lowest) values of the objective function. To reach that goal, there are 2 possibilities:

1. Exact methods;
2. Heuristic methods;

3.3.1.1 EXACT METHODS

The exact methods for the VRPTW can be classified into three categories: Lagrange relaxation-based methods, column generation and dynamic programming. Exact methods often perform very poorly (in some cases taking days or more to find moderately decent, let alone optimal, solutions even to small instances).

- Lagrange relaxation-based methods. There are a number of papers using slightly different approaches. There is variable splitting followed by Lagrange relaxation, K-tree approach followed by Lagrange relaxation (Fisher et al., 1997; Holland, 1975, and in Kohl and Madsen (1997) presented shortest path with side constraints approach followed by Lagrange relaxation. The relaxes of the constraints ensuring that every customer is served exactly once, that is

$$\sum_{j}^{n}\sum_{k}^{m} x_{ijk} = 1 \qquad \forall i \epsilon C$$

is relaxed and the objective function with the added penalty term then becomes

$$min \sum_{j}^{n}\sum_{k}^{m}\sum_{i}^{C}(c_{ij} - \lambda_j) * x_{ijk} + \sum_{j}^{n}\lambda_j$$

Here λ_j is the Lagrange multiplier associated with the constraint that ensures that customer *j* is serviced.

- Column generation. In Desrosiers et al. (1984), the column generation approach is used for the first time for solving the VRPTW, and more effective version of the same model with addition of valid inequalities solves more instances to optimality in Desrosiers et al. (1992). If a linear program contains too many variables to be solved explicitly, then we can initialize the linear program with a small subset of the variables and compute a solution of this reduced linear program. Afterwards, we check if the addition of one or more variables, currently not in the linear program, might improve the linear program solution. This check can be done by the computation of the reduced costs of the variables. In this case, a variable of negative reduced cost can improve the n-th solution.

- Dynamic programming. The dynamic programming approach for VRPTW is presented for the first time in Kolen et al. (1987) and Christofides and Beasley (1984) are uses the dynamic programming paradigm to solve the VRP. The algorithm of Kohl and Madsen (1997) uses branch-and-bound to achieve optimality, but there are also other examples, far from the topic of this thesis.

3.3.1.2 HEURISTIC METHODS

The non-exact algorithms for the VRPTW have been very active-far more active than that of exact algorithms. A heuristic is defined by Reeves (1995) as a technique which seek good (near-optimal) solutions at a reasonable computational cost without being able to guarantee optimality, to state how close to optimality a feasible solution is or, in some cases, even to guarantee feasibility. Often heuristics are problem-specific, so that a method which works for one problem cannot be used to solve a different one. Here there are the most used heuristic methods.

- Route-building heuristics. The first paper on route-building heuristics for the VRPTW is Baker and Schaffer's, in 1989. Their algorithm is an extension of the saving heuristic of the VRP (Clarke and Wright, 1964). The algorithm begins with all possible single-customer route (depot-i-depot). In every iteration we calculate which two routes can be combined with the maximum saving. A time oriented nearest-neighbourhood algorithm is developed by defining the savings as a combination of distance, time and time until feasibility. Additionally, we have to check for violation of the time windows when two routes are combined. These heuristics have time complexity of $O(n^2 \log n^2)$.

- Route-improving heuristics. The basis of almost every route-improving heuristic is the notion of a neighbourhood. The neighbourhood of a solution S is a set N(S) of solutions that can be generated with a single modification of S. Checking some solutions in a neighbourhood might reveal solutions that are better with respect to objective function. This idea can be repeated from the better solution. At some point no better solution can be found and an optimum has been reached.

- R-Opt algorithms. One of the most used improvement heuristics in routing and scheduling is the r-Opt heuristic. Here r arcs removed and replaced by r other arcs. A solution obtained using an r-Opt neighbourhood that cannot be improved further is called r-Optimal. Usually r is at most 3. Using the 3-Opt on the routes of a solution to the VRPTW problem is not without problems. For all possible 2-Opt interchanges and some of the exchanges in the 3-Opt neighbourhood, parts of the route are reversed. This may very likely lead to a violation of the time windows.

- Simulated annealing. The name ''simulated annealing'' is due to the fact that conceptually it is like a physical process, known as annealing, where a material is heated into a liquid stat then cooled back into a recrystallized solid state.

- Tabu search. The tabu search is one of the old metaheuristics. It was introduced by Glover (1989) and Fisher et al. (1997). At each iteration the neighbourhood of the current solution is explored and the best solution in the neighbourhood is selected as the new current solution. In order to allow the algorithm to escape from a local optimum the current solution is set to the best solution in the neighbourhood even if this solution is worse than the current solution. To prevent cycling visiting recently selected solutions is forbidden. This is

implemented using a tabu list. Often, the tabu list does not contain illegal solutions, but forbidden moves. It makes sense to allow the tabu list to be overruled if this leads to an improvement of the current overall best solution.

- The genetic algorithms. The genetic algorithms are probabilistic procedures of search, which took as a starting point the genetic evolution of a species. The principal idea is to reproduce the natural evolution of organisms, generation after generation, by respecting the phenomena's heredity and law of survival stated by Darwin. Later Holland (1975), Goldbeerg (1989) and Fisher (1994) adapted the genetic algorithms to the resolution of combinatorial optimization problems. Contrary to simulated annealing and tabu search, the genetic algorithms operate in a population of solution and not only one solution (Pirlot, 1996). A population of structures, each one corresponding to a possible solution, represents the space of the solution of the problem. The new structures are generated by application of genetic operators (selection, crossing and mutation) on the potential parents chosen inside the population. The genetic algorithms are based on the principle that best parents produce best children; so, the strongest members of the population have strong probability to be selected as the parents.

3.3.2 SOLUTION APPROACH IN SOTRAL S.R.L

Now it's the time to talk about the approaches introduced in the thesis, both in Sotral and in the application created by the author. First of all, it's better talk about Sotral's solution's algorithm.

Before starting, it is necessary to specify important information: Sotral has adopted this method, totally manually, after years and years of experience in the sector, and after numerous attempts with different methods. This method resembles a lot the Sweep algorithm theorised by Wren in 1971, and then developed by Gillet and Miller in 1974.

The proposed solution is divided into the following steps:

Step 1. From the cooking site, an imaginary vertical line is drawn;

Step 2. Create a new route

Step 3. While all nodes have been processed, rotate the imaginary line clockwise until the line reaches a delivery site;

a. While the current route is fulfilled
   i. if the node satisfies the constraints, insert the node in the route;

b. Create a new route.

c. Go to step 3.

Step 4. Analyse each route created and try to find a new organisation with lower cost.

The flow chart of the algorithm is depicted in Flow Chart of Sotral's Adopted method .

3.3.2.1 COMPUTATIONAL COMPLEXITY

Assuming there are *n* nodes to be sorted in the problem of vehicle routing, an estimation of the amount of necessary operations to find a solution to the problem is:

1. Creation of a new empty route: 1 operation required.
2. A node is chosen from the graph. One operation is required, but since we are in a network with *n* points, the operation will be requested *n* times, then O (*n*) operations will be required.
3. Insertion of nodes in empty routes: 1 operation.
4. Calculation of the route load and time taken when a new node is to be added. 2 * O (*n*) = O (*n*) operations.
5. Check whether the route load and the journey time, respectively, do not violate the load constraints of the vehicle and the delivery interval of the currently open route. This process is done for all the nodes of the network: O(*n*) operations.
6. Repetition of the control process an amount of times equal to that of the routes created. Usually we have a number less than *n* (up to a minimum of 1) of routes created, so we repeat *n* times (at most) a process that requires O(*n*) operations. n * O (n) = O(n^2).
7. After the node assignment phase to the routes, we move on to the sorting. Choosing an unordered route requires an operation.

8. The search for a new order is purely instinctive: you order the node vector from the beginning so that it may seem like an improving solution. An operation of this kind requires O (n) operations, usually repeated twice. O(n^2).
9. Time control obtained with the new solution with the previous one. 1 operation.
10. Repeat steps 8 and 9 until all the routes have been ordered. O(n) * (O(n^2) + 1)

The computational complexity of the method is:
$1 + 1 + (O(n) + O(n) + O(n^2)) + (O(n^2) + 1) * O(n) = O(n^3) + O(n^2) + O(3n) + 2 \approx O(n^3)$.

3.3.3 CLARK AND WRIGHT'S ALGORITHM

The first algorithm inside the Java program is the Clarke and Wright's one (further can be called also CW). This algorithm has an initial setup in which every route has a single node inside, so the number of routes is equal to the number of nodes of graph. In the parallel version we have the addition, in each repetition, of a link between two delivery points, before belonging to 2 different routes, and to do this, we consider the savings due to link two different nodes:

$$sav_{ij} = d_{i0} + d_{0j} - \mu d_{ij} \quad \mu \geq 0$$

For example, if μ=1, sav_{ij} is the distance (or time) saved when a delivery in node i and j is fulfilled in a unique route instead of serving them separately.

According to the presence of time intervals, it's needed to decide the route orientation. Two partial routes that have as final customers i and j have compatible orientations if the node i is the first (last) in its route and j is the last (first) in its route: the admissible links start from the last customer of a route to the first one of another route. In each step of the algorithm, it's necessary to check that time constraints (linked to time intervals) are not violated.

The heuristic algorithm described in this way could find a good link in two nodes very close in space, but far away in time. Connections like these will lead to waiting times that will increase costs: the vehicle, in fact, during waiting that node is ready to be served, may work in other places.

3.3.4 NEAREST NEIGHBOUR

This second heuristic algorithm, the Nearest Neighbour (successively named also NN), is a sequential one. It starts each route finding the "nearest" non-assigned customer to depot. In each repetition we search for the "nearest" node from the last customer inserted. The research is done among all customers that can (respecting the capacity and time constraints) be added to the tail of the considered route. A new route starts when the research fails or there are no more nodes to insert.

This kind of approach want to consider both the nodes "near in time" and "near in space". It's also checked if inserting the customer in a route can be accepted, and whatever external node *j* that may be visited in the next steps of the method.

The parameter C_{ij} will take in account of:

- distance from two nodes (d_{ij});
- difference between time spent to finish the service in node i and the starting time in customer *j* (T_{ij});
- the urgency to deliver to customer *j* *j*(v_{ij}), expressed like the remaining time before the last service may start.

Formally:

$T_{ij} = b_j - (b_i + s_i)$,

$v_{ij} = l_j - (b_i + s_i + t_{ij})$,

and

$c_{ij} = \delta_1 d_{ij} + \delta_2 T_{ij} + \delta_3 v_{ij}$

it is defined by weights satisfying $\delta_1 + \delta_2 + \delta_3 = 1$, $\delta_1 \geq 0$, $\delta_2 \geq 0$, $\delta_3 \geq 0$.

3.4 SOFTWARE ALGORITHMS SPECIFICATION

Opposed to the manual solution adopted in the company Sotral, it is proposed a solution by adopting a computer software tool in order to solve the problem by mean a personal computer.

The application should be simple and intuitive to navigate as the users are most likely unfamiliar with the topic of route optimisation. In addition, it should provide an easy way to add and edit input data, offer a way to display the solution found, and be able to save and load the current problem configuration.

The software tool shall exploit two different types of algorithms according to the two methods mentioned above:

- Time-Oriented Clarke & Wright;
- Time-Oriented Nearest Neighbour.

3.4.1 TIME ORIENTED CLARK&WRIGHT

This procedure begins with *n* distinct routes in which each customer is served by a dedicated vehicle. The parallel version of this tour-building heuristic is characterized by the addition at every iteration of a link of distinct, partially formed routes between two end customers.

For example, when $i = 1$, sav_{1j} is the savings in distance that results from servicing customers *1* and *j* on one route as opposed to servicing them individually, directly from the depot.

Due to the existence of time windows, the algorithm shall take in account the route orientation. Two partial routes with end customers *i* and *j*, respectively, have compatible orientations if *i* is first (last), and *j* is last (first), i.e. the admissible links are from the last customer (*l*) on one route to the first customer (*f*) on another.

Furthermore, in addition to considering the vehicle capacity constraints, the algorithm shall check the time window constraints for violation at every step in the heuristic process.

3.4.1.1 ALGORITHM

The algorithm computes all the savings s_{ij} between customers *i* and *j*. Assuming that c_{i0} is the cost of travelling from the depot to customer *i* and c_{ij} is the cost of travelling from customer *i* to *j*.

The Clarke and Wright algorithm is a greedy one: at each iteration it selects the route merger that yields the largest saving. The deterministic nature of the Clarke and Wright algorithm results in the algorithm producing the same solution every time it is run on the same instance.

A straightforward extension of such a greedy algorithm is to add a controlled randomization in the greedy selection rule, to allow the algorithm to generate a different solution at each iteration [Pichpibul, T. Kawtummachai, R., 2012].

Before each run, the algorithm randomizes the current savings list using a process the authors call "a combination of tournament and roulette wheel selection." The randomized savings list is formed by iteratively choosing the next customer pair using tournament selection (i.e., proportionally to the savings of this customer pair) among the first T elements in the savings list. When a customer pair is chosen, it is removed from the current savings list and added to the randomized savings list. When all customer pairs have been selected, a new solution is constructed by the algorithm using this randomized savings list. If the solution found using the randomized savings list is both valid and better than the best solution found so far, the randomized savings list replaces the current savings list.

Step 1. Initialisation: read the matrix containing the information about the X and Y coordinates, demands, time windows and service time of the points;

Step 2. Compute the savings $s_{ij} = t_{i0} + t_{0J} - t_{ij}$ for $i,j = 1,\ldots, n$ and $i \neq j$. Rank the savings s_{ij} and list them in descending order;

Step 3. While r is less than the number of repetitions decided by the user

a. Create a "*randomized savings list*". Process the savings list beginning with the topmost entry in the list (the largest s_{ij}). For the savings under consideration (s_{ij}), include link (*i, j*) in a route if no route constraints will be violated through the inclusion of (*i, j*). The following three cases need to be considered:

   - *Case 1*: If neither *i* nor *j* have already been assigned to a route, then a new route is initiated including both *i* and *j*;
   - *Case 2*: If exactly one of the two points (*i* or *j*) has already been included in an existing route (interior point) and the other one point is not interior to that route (extension point), then:

     If the extension point is not violating the capacity and time window constraints, then add the point to the same route;

     Otherwise make a new route with the point *i*;
   - *Case 3*: If both points are extension points and, if the service time of the last node of the first route is compatible with the time interval of the first node of the second route and capacity constraints are satisfied, then merge the two routes;

b. If the savings list s_{ij} has not been exhausted, return to Step 2, processing the next entry in the list; otherwise, stop.

c. Compute the cost of found solution.

Step 4. Print resulting routes of best solution found.

Step 5. Depict unused nodes if any.

The flow chart is depicted in Flow Chart of the Clarke Wright algorithm.

3.4.1.2 COMPUTATIONAL COMPLEXITY OF CALCULATION

1. Reading and saving the data inside the application (so time intervals, service time, and time needed to go from a node to another). $O(n^2) + 2*O(n)$ operations required
2. For each link between two nodes, create a value of savings with the formula, and saving it in a new matrix. $O(n^2)$ operations
3. Choosing a value of savings, we find the 2 nodes in matrix. $2*O(n^2)$ operations (it's done for each saving once at least) = $2*O(n^2)$ operations.
4. Check if the two nodes are in already created routes. $O(n^2)$ operations.
5. Find the value of meals delivered and checking if the number is lower than maximum capacity of vehicle. $2*O(n)$ operations.
6. Find the processing time and check if it respects time constraint. $O(n^2)$ operations.
7. Inserting the two nodes in routes, according to different cases. $O(n)$ operations required.
8. Repetition of this process up to the running out of the list of customers to serve and repeat again from 1 to 8 for the number of iterations declared. (Standard 100 iterations)
9. Printing the best solution found solution. $O(n)$ operations required.

$100 *[O(n^2) + 2*O(n) + O(n^2) + 2*O(n^2) + O(n^2) + O(2n) + O(n^2) + O(n) + O(n) \approx O(600n^2) + O(600n)] = O(600n^2)$ operations.

3.4.2 TIME ORIENTED NEAREST NEIGHBOUR

This method incorporates the procedure of a classic neighbouring system, with the only difference that our problem involves delivery intervals, so it is necessary to consider them in the solution. The nearest-neighbour heuristic generally starts every route by finding the un-routed customer "closest" (in terms of a measure to be

described later) to the depot. At every subsequent iteration, the heuristic searches for the customer "closest" to the last customer added to the route. This search is performed among all the customers who can feasibly (with respect to time windows, vehicle arrival time at the depot, and capacity constraints) be added to the end of the emerging route.

A new route is started any time the search fails, unless there are no more customers to schedule.

3.4.2.1 THE ALGORITHM

The nearest-neighbour heuristic starts every route by finding the un-routed customer "closest" (in terms of a measure described later) to the depot. At every subsequent iteration, the heuristic searches for the customer "closest" to the last customer added to the route.

The metric used in this approach tries to account for both geographical and temporal closeness of customers. Let the last customer on the current partial route be customer *i*, and let *j* denote any un-routed customer that could be visited next. Then the metric used, c_{ij}, measures the direct distance between the two customers, d_{ij}, the time *difference between the completion of service at i and the beginning of service at j,* T_{ij}*, and the urgency of delivery to customer j,* v_{ij}*, as expressed by the time remaining until the* vehicle's last possible service start.

Measures

The new route can be set on various criteria (e.g. the farthest un-routed customer, un-routed customer with the earliest deadline or un-routed customer with the biggest demand). Generally, the decision is for the lowest distance travelled, but we use the method with a weighted average between the time difference to serve a new node (how much the vehicle is early), the urgency of serving that node (how much time is left before the vehicle comes to the latest time in the time interval of a node), and the lowest distance. The selected customer is then inserted in the route and the calculation and selection is repeated until the time or capacity resource is exhausted. New resources are generated with new route/vehicles. It is not trivial to find the best coefficient for the weighted average for real-world problems. A good starting point for the tuning algorithm can be found in the original paper (Solomon, 1987).

Coefficient Weighted Distance Time Heuristics for CVRPTW

Based on the assignment of weights to the closing part of a time windows and distances to the serving places, the Coefficient Weighted Distance Time Heuristics (CWDTH) has been developed (Galić et al., 2006b; Carić et al., 2007). In each iteration the algorithm simultaneously searches for the customer with the soonest closing time of requested delivery and minimum distance from the current vehicle position. The route is designed starting with one vehicle. In each subsequent iteration the customer who best matches the given criteria is served. When the vehicle has used all the available capacity that can be utilized regarding the amount of demands, it returns to the depot. A new vehicle is created, and the described process is repeated. In the moment when all customers have been served, the algorithm stops. Automatic parameter adjusting is implemented for the weighting of distance over delivery closing time.

Note: in the application software the weighted coefficients are set hard coded as follow:

- Time Weight = 0.5
- Distance Weight = 0.5
- Urgency Weight = 0.0

That's because they seem to be the best effective set.

Step 1. Initialisation: read the matrix containing the information about the X and Y coordinates, demands, time windows and service time of the points.

Step 2. Create a distances matrix containing the distances among:
- nodes *i* and *j*
- nodes *i* and depot

Step 3. Create a new route with starting depot node

Step 4. While remaining nodes' list size is equal to 0:

a. While all nodes have been visited

i. Find the node closest to the last inserted node in the current route

ii. If the capacity and time window constraints are respected, then insert the node in the current route.

b. Create a new route.

Step 5. Print resulting routes of best solution found.

Step 6. Depict uninserted nodes if any.

The flow chart is depicted in Flow Chart of the Nearest Neighbour algorithm.

3.4.2.2 COMPUTATIONAL COMPLEXITY

1. The software gets in input a matrix. 1 operation required
2. Save information about times taken, delivery intervals and service time in the class. 2*O(n) + $O(n^2)$ operations required.
3. Perform a sort of all nodes, inserting, after the deposit, the node closest to him, so the one with the shortest time to reach it.
4. Repeat step 3 until the sorting of the nodes' array is complete. In the best situations, the number of operations needed to sort a matrix are O(n*log n), in the worst cases $O(n^2)$.
5. Initialize the first tour, to which the vehicle number 1 is assigned, with the first node in the multidimensional array. 1 operation required
6. Look for the node closest to this. If the sum between the departure time from node 1 and the travel time reaches the delivery interval of node 2, and the quantities requested are lower than the load limit of the vehicle, then you can add the node 2 in tour one.
    a. The node is inserted in the first tour. If insertion cannot be performed, then proceed with node 3 and repeat the check. And so on, up to node n^{th} of the graph, or up to the moment in which the number of meals delivered is equal to maximum capacity of vehicle. 2*O(n) operation required.
7. This procedure is repeated (point 7 and 8) with new vehicles, until each node has a tour that reaches it. 2*n*O(n)= $2*O(n^2)$ operation required totally.

$1 + O(n^2) + 2*O(n) + O(n^2) + 1 + 2*O(n^2) = O(3n^2) + O(2n) + 2 \approx O(3n^2)$ operations needed.

3.5 SOFTWARE DEVELOPMENT

The software application developed in order to solve the problem in this thesis was conducted in the programming language Java.

3.5.1 GRAPHICAL USER INTERFACE

The design of graphical user interface is done using Eclipse for Java Developer 2018-12.

The following screenshot depicts the main panel of the tool, where the user is allowed to:

- create a new VRP problem by scratch;
- save the current problem as named project;
- load a saved project;
- select the method to execute;
- run the code concerning the method selected and view the results.

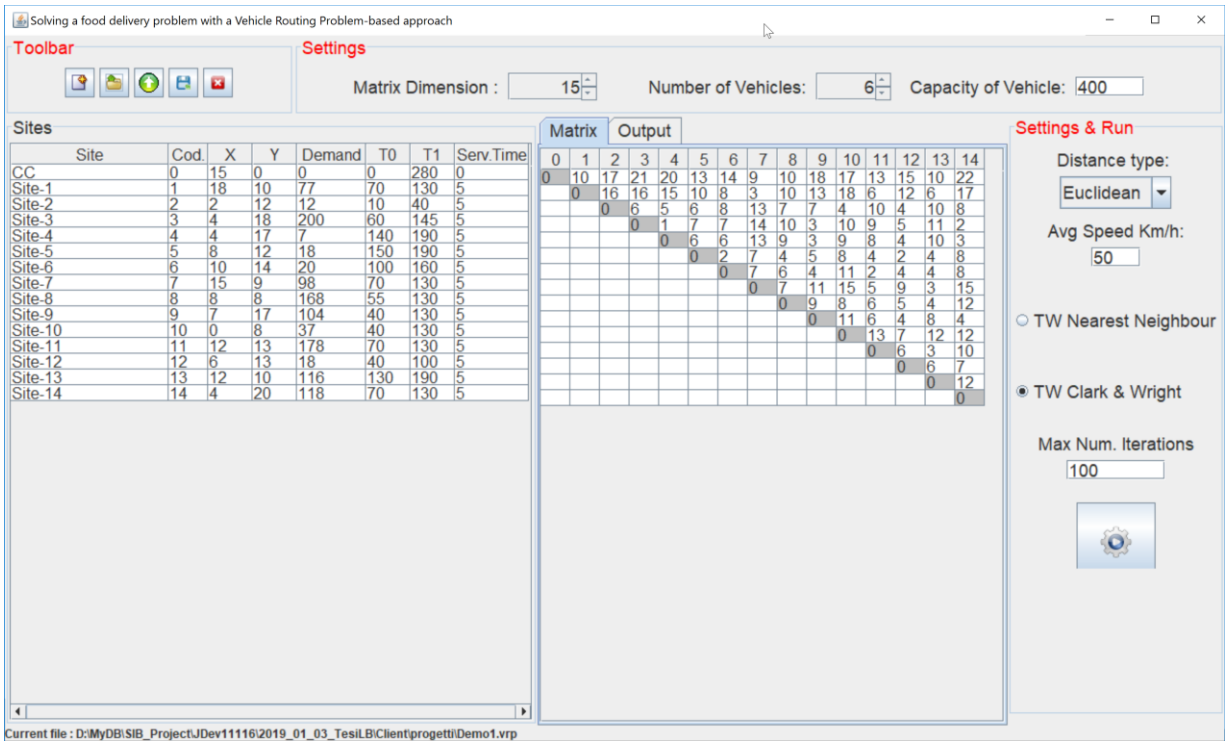


Figure 3 Graphical User Interface

The full description of the software application is depicted in Appendix B: Software User Manual

Examples of outputs produced by the software application are depicted hereafter:

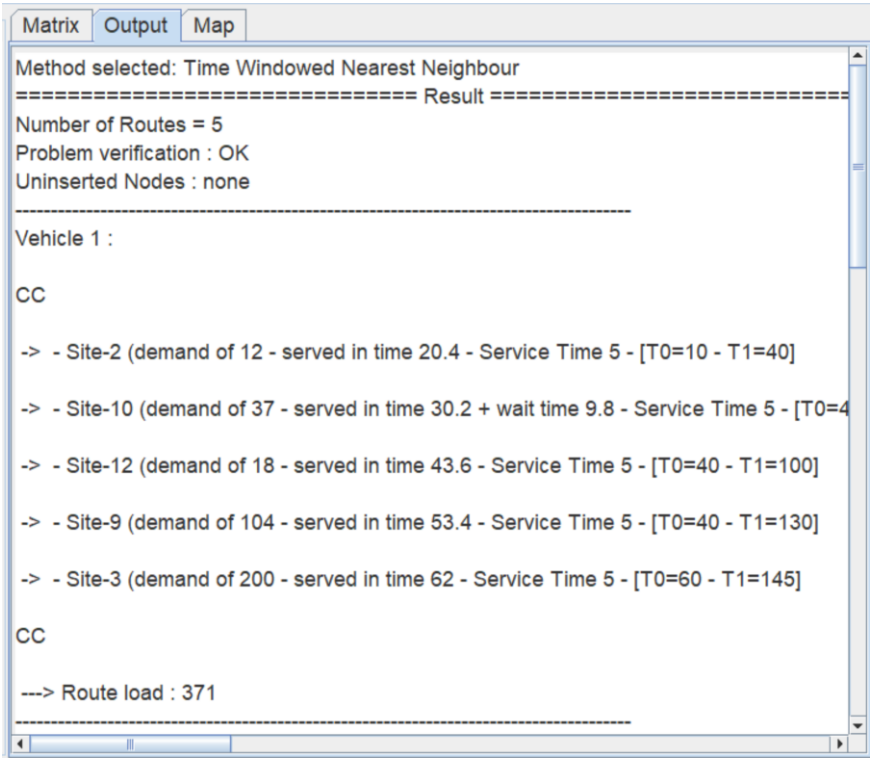


Figure 4 Output solution

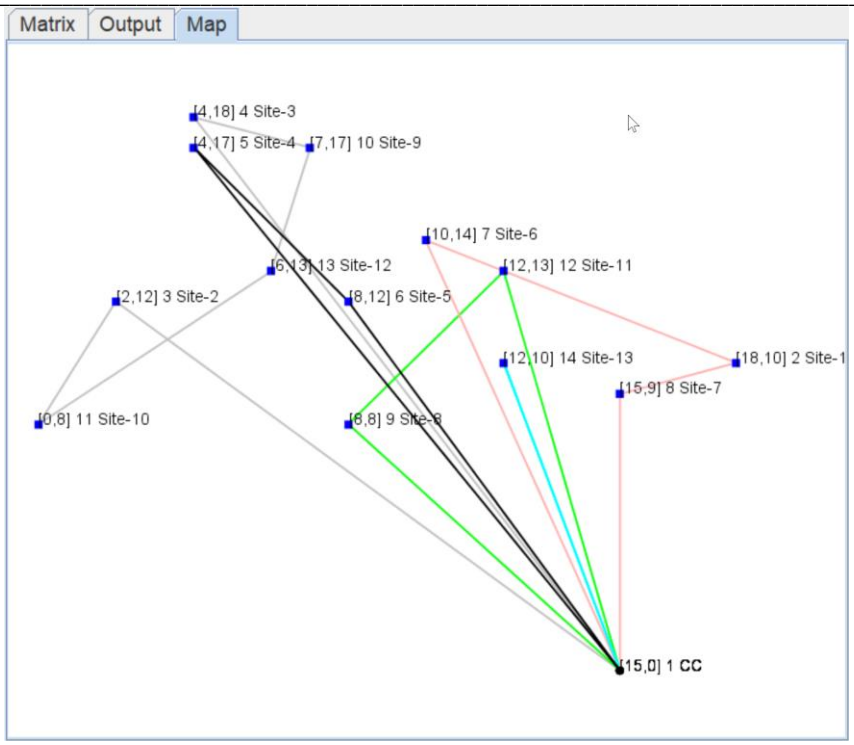


Figure 5 Graph solution

3.5.2 HARDWARE REQUIREMENTS

The application has been developed on a PC having the following features:

- Intel i5 7th Generation
- RAM Memory 8 GB
- Operating System Windows 10, 64 bit

Anyway, the application will run on a more barely configured PC as well. For example:

- Pentium(R) Dual Core CPU T4200 @2.00 GHZ
- Memory (RAM) 1.00 GB
- System 32-bit Operating System Windows

4 RESULTS

In order to give evidence of goodness of work performed, firstly, the developed software has been used to produce solutions about the two real cases faced in Sotral, i.e. the requirements included in the invitation to tender concerning the bid about the delivery of foods in two towns.

After that, it is performed the run of the tool against some of Solomon and Homberger's instance files; then the results are compared against the best know solution available in literature for these benchmarks.

A summary of the computational results obtained are presented in this section. As mentioned before, the purpose of this thesis is to look at possible gain concerning Sotral delivery system, as the goal is not to develop superior VRP solver.

4.1 PREAMBLE

Any heuristics are non-deterministic and contain some random components such as randomly chosen parameter values. The output of separate executions of these non-deterministic methods on the same problem is in practice never the same. This makes it difficult to analyse and compare results. Using only the best results of a non-deterministic heuristic, as is often done in the literature, may create a false picture of its real performance.

The results are usually ranked according to a hierarchical objective function, where the number of vehicles is considered as the primary objective, and for the same number of vehicles, the secondary objective is often either total travelled distance or total duration of routes. Therefore, a solution requiring fewer routes is always considered better than a solution with more routes, regardless of the total travelled distance. According to Bräysy (2001b) these two objectives are very often conflicting, meaning that the reduction in number of vehicles often causes increase in total travelled distance.

4.2 COMPARISON WITH SOTRAL'S ACTUAL METHOD USED

As a starting point, the routes that have been computed by adopting the manual method adopted by Sotral.

Then, they have been defined the problem data used as input of the software tool. The two figures below depict the data as they come from the two real world customer data concerning the bid. To maintain the data as much neutral as possible, they are called City A and City B.

Note: the sites' names have been pseudo-coded, as for contractual privacy statement.

SITE	COD	X	Y	DEMAND	T0	T1	SERVICE TIME
CC	0	15	0	0	0	280	0
Site-1	1	12	10	77	70	130	5
Site-2	2	0	8	12	10	40	5
Site-3	3	4	18	200	60	145	5
Site-4	4	4	17	7	140	190	5
Site-5	5	8	12	18	150	190	5
Site-6	6	8	12	20	100	160	5
Site-7	7	9	9	98	70	130	5
Site-8	8	8	8	168	55	130	5
Site-9	9	7	17	104	40	130	5
Site-10	10	0	8	37	40	130	5
Site-11	11	6	17	178	70	130	5
Site-12	12	6	13	18	40	100	5
Site-13	13	12	10	116	130	190	5
Site-14	14	4	18	118	70	130	5

Figure 6 City A – Problem data

SITE	COD	X	Y	DEMAND	T0	T1	SERVICE TIME
CC	0	17	13	0	0	0	0
Site-1	1	17	14	35	315	345	5
Site-2	2	13	13	265	315	345	5

SITE	COD	X	Y	DEMAND	T0	T1	SERVICE TIME
Site-3	3	7	19	390	315	345	5
Site-4	4	0	5	120	315	345	5
Site-5	5	13	0	60	315	345	5
Site-6	6	20	6	30	315	345	5
Site-7	7	13	13	200	330	375	5
Site-8	8	3	14	150	330	375	5
Site-9	9	8	17	120	330	375	5
Site-10	10	7	19	150	330	375	5
Site-11	11	2	4	290	330	375	5
Site-12	12	7	8	115	330	375	5
Site-13	13	7	8	200	330	375	5
Site-14	14	7	19	50	390	420	5
Site-15	15	3	14	20	390	420	5
Site-16	16	7	8	65	390	420	5
Site-17	17	18	15	65	390	420	5
Site-18	18	9	11	120	390	420	5
Site-19	19	2	4	120	390	420	5
Site-20	20	2	4	100	390	420	5
Site-21	21	19	3	120	390	420	5
Site-22	22	19	3	215	330	375	5
Site-23	23	19	3	100	390	420	5

Figure 7 City B – Problem data

Finally, the software tool has been run by using different settings:

- vehicle capacity: 400, 600, 800;
- average speed of vehicle Km/h: 25, 50, 75.

Concerning the Clarke&Wright method, a further parameter about the number of iterations has been set:

- max num iterations: 100, 500, 1000.

As expected, different results have been produced by adopting each value of above settings. The average of these values has been computed in order to have a single value to include in comparision.

Note: it is assumed that a vehicle must not wait more than 10 minutes in the same node, before serving it.

4.2.1 CITY A

The method adopted by Sotral, in this file, produces the solutions:

SOTRAL	SOLUTION COST	# OF VEHICLES NEEDED
MAX CAP 400, AVG SPEED 25	402	11
MAX CAP 400, AVG SPEED 50	328	9
MAX CAP 400, AVG SPEED 75	194	6
MAX CAP 600, AVG SPEED 25	402	11
MAX CAP 600, AVG SPEED 50	328	9
MAX CAP 600, AVG SPEED 75	194	6
MAX CAP 800, AVG SPEED 25	402	11
MAX CAP 800, AVG SPEED 50	328	9
MAX CAP 800, AVG SPEED 75	194	6

Table 1 Method adopted by Sotral – City A

It can be noticed that there is no difference if maximum capacity is increasing: both the solution cost and the number of vehicles are dependent only on the average speed of vehicles, since the number of meals delivered are not so much to justify a higher vehicle capacity.

NN	SOLUTION COST	# OF VEHICLES NEEDED
MAX CAP 400, AVG SPEED 25	150	4
MAX CAP 400, AVG SPEED 50	212	6
MAX CAP 400, AVG SPEED 75	214	6
MAX CAP 600, AVG SPEED 25	168	4
MAX CAP 600, AVG SPEED 50	191	5
MAX CAP 600, AVG SPEED 75	187	5
MAX CAP 800, AVG SPEED 25	140	4
MAX CAP 800, AVG SPEED 50	199	6
MAX CAP 800, AVG SPEED 75	179	5

Table 2 NN algorithm – City A

The table above depicts the solution produced by the tool with the Nearest Neighbour algorithm. This time, instead, different values have been found also for different settings of capacity of the vehicles, as expected.

CW	100 ITERATIONS	500 ITERATIONS	1000 ITERATIONS	AVG SOLUTION COST	# VEHICLES
MAX CAP 400, AVG SPEED 25	165; 5	170; 5	157;5	164	5
MAX CAP 400, AVG SPEED 50	182; 5	186; 5	186; 5	184,7	5
MAX CAP 400, AVG SPEED 75	185; 5	186; 5	185; 5	185	5
MAX CAP 600, AVG SPEED 25	135; 4	119; 4	135; 4	129,7	4
MAX CAP 600, AVG SPEED 50	157; 4	156; 4	150; 4	154,3	4
MAX CAP 600, AVG SPEED 75	158; 4	157; 4	156; 4	157	4
MAX CAP 800, AVG SPEED 25	110; 3	109; 3	109; 3	109	3
MAX CAP 800, AVG SPEED 50	148; 4	147; 4	145; 4	146,7	4
MAX CAP 800, AVG SPEED 75	149; 4	148; 4	148; 4	148	4

Table 3 CW algorithm – City A

Concerning the Clarke & Wright method, from a summary analysis of the above table, it can be deducted that by increasing the number of iterations is not a statement for a better solution: indeed, it can be noted the settings MAX CAP = 400 and AVG SPEED = 75 brings to the same best solution for every number of iterations.

The results depicted in the above table give evidence that:

- the manual method adopted in Sotral is both a long-running process and strongly "greedy". Furthermore, it gives a solution that provides systematically more routes than the NN and the CW method implemented in the software application;
- by increasing the MAX CAP, the CW method provides better solutions than the others, while instead with the NN the solutions found are quite the same.

  Here there are 2 bar charts to depict the situation. Sotral's method, especially in cases of low capacity and average speed, finds solution with double costs

respect with the other two algorithms. In case of high average speed, all the solutions are quite comparable.

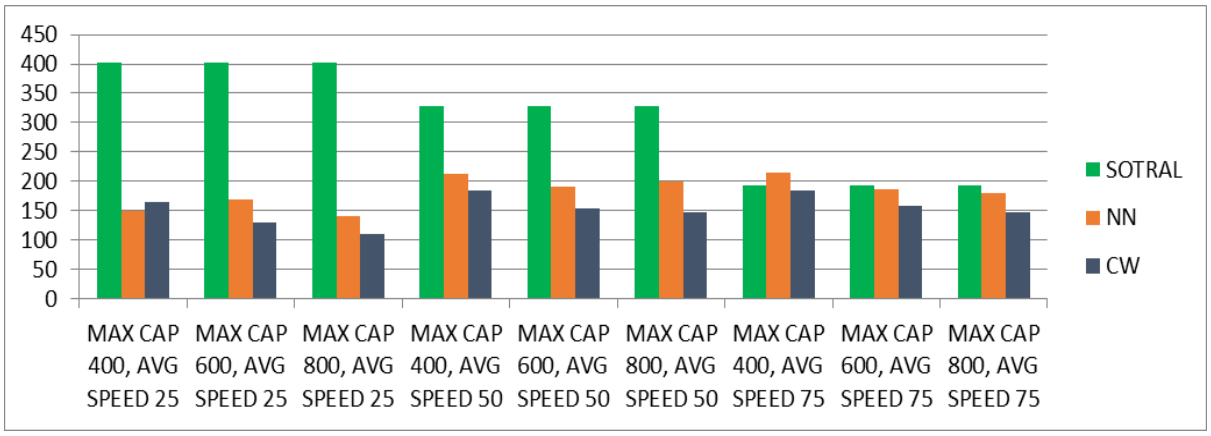


Figure 8 City A – Solution cost

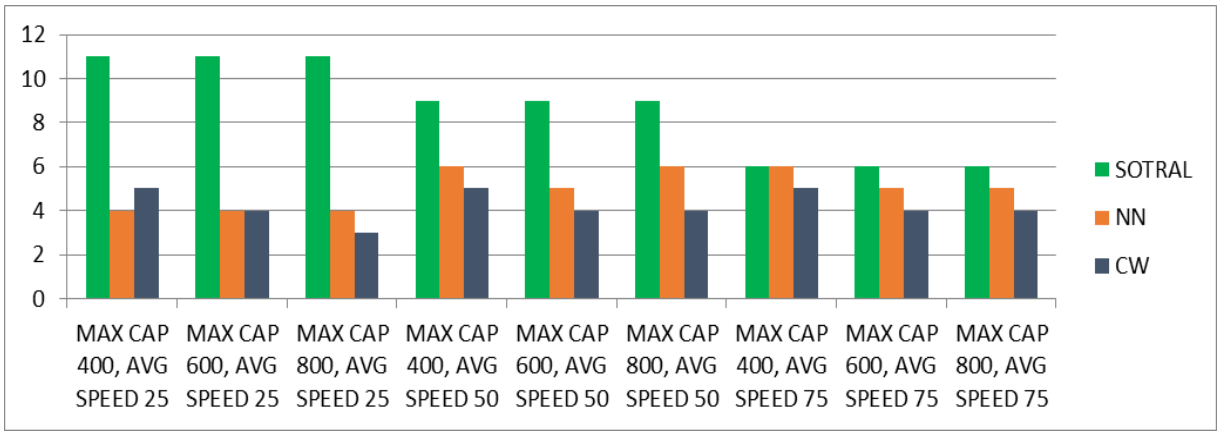


Figure 9 City A – # of vehicles

4.2.2 CITY B

Also, in this case, we can notice that the manual method used in Sotral provides solutions that are more dependent on the AVG SPEED than on the maximum capacity of the vehicles.

SOTRAL	SOLUTION COST	# OF VEHICLES NEEDED
MAX CAP 400, AVG SPEED 25	351	13
MAX CAP 400, AVG SPEED 50	316	12
MAX CAP 400, AVG SPEED 75	312	10
MAX CAP 600, AVG SPEED 25	351	13
MAX CAP 600, AVG SPEED 50	323	12
MAX CAP 600, AVG SPEED 75	296	11
MAX CAP 800, AVG SPEED 25	351	13
MAX CAP 800, AVG SPEED 50	323	12
MAX CAP 800, AVG SPEED 75	257	9

Table 4 Method adopted by Sotral – City B

Instead, the NN method, provides worse solution when AVG SPEED increases. This happens because going faster you may arrive to the same node earlier, but with a risk in increasing waiting time. The addition of a node in a pre-existent route balances this increasing of solution cost.

NN	SOLUTION COST	# OF VEHICLES NEEDED
MAX CAP 400, AVG SPEED 25	309	12
MAX CAP 400, AVG SPEED 50	329	11
MAX CAP 400, AVG SPEED 75	318	10
MAX CAP 600, AVG SPEED 25	254	10
MAX CAP 600, AVG SPEED 50	266	9
MAX CAP 600, AVG SPEED 75	260	8
MAX CAP 800, AVG SPEED 25	265	9
MAX CAP 800, AVG SPEED 50	252	8
MAX CAP 800, AVG SPEED 75	228	6

Table 5 NN algorithm – City B

As well as results for City A, using the CW method will lead to better results than the other two methods. It is useful to remark the fact that, very often, increasing the number of iterations lead to worse results; this is due to the random-based “shaking” performed on savings before each iteration, then the best solution found after just 100 iterations.

CW	100 ITERATIONS	500 ITERATIONS	1000 ITERATIONS	AVG SOLUTION COST	# VEHICLES
MAX CAP 400, AVG SPEED 25	281; 10	273; 11	273; 11	275,7	11
MAX CAP 400, AVG SPEED 50	267; 10	267; 10	267; 10	267	10
MAX CAP 400, AVG SPEED 75	267; 10	267; 10	267; 10	267	10
MAX CAP 600, AVG SPEED 25	220; 8	220; 7	219; 7	220	7
MAX CAP 600, AVG SPEED 50	216; 8	218; 8	218; 8	217	8
MAX CAP 600, AVG SPEED 75	215; 8	215; 8	215; 8	215	8
MAX CAP 800, AVG SPEED 25	204; 7	207; 6	196; 6	202,3	6
MAX CAP 800, AVG SPEED 50	207; 8	203; 7	207; 8	205,7	8
MAX CAP 800, AVG SPEED 75	206; 7	206; 7	196; 6	202,7	7

Table 6 CW algorithm – City B

Here we have two bar charts for the comparisons. We can notice that the results with the Sotral's method don't change too much, while with the other two methods the numbers have a dramatic fall, both in solution cost and in number of vehicles.

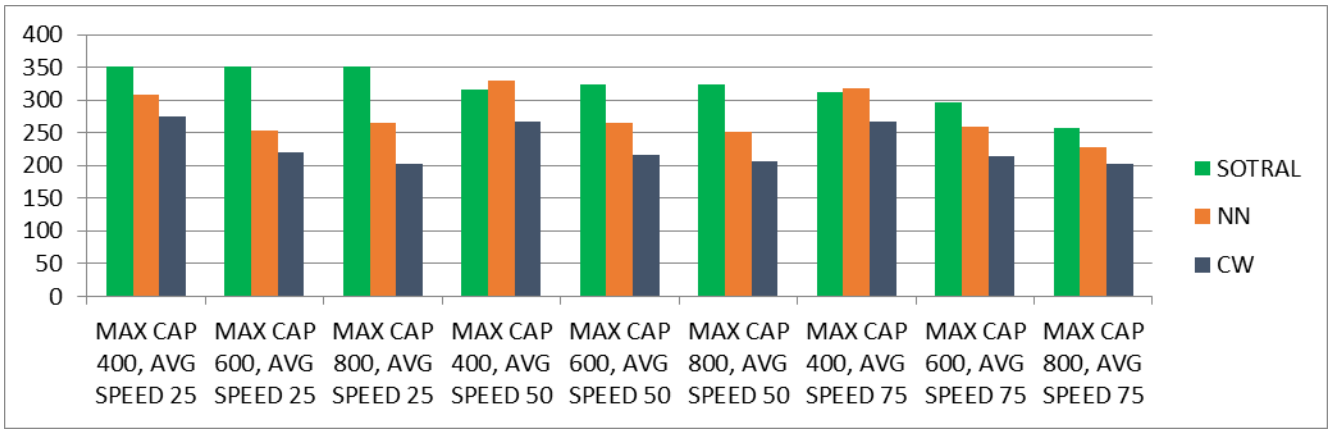


Figure 10 City B – Solution Cost

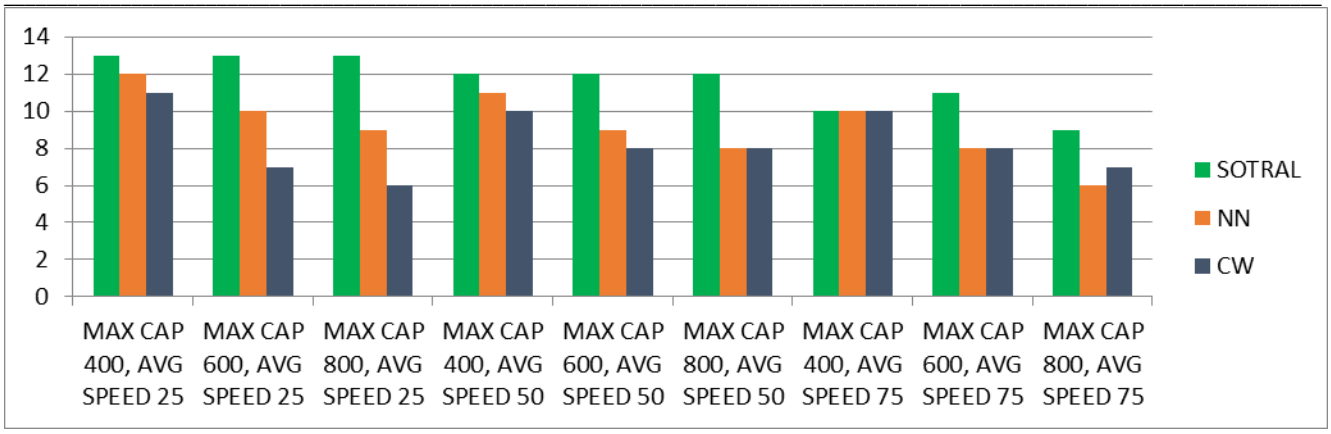


Figure 11 City B – # of Vehicles

4.3 COMPARISON AGAINST SOLOMON BEST KNOWN RESULTS

The following sections depict the performance of the implemented methods in the software application in terms of accuracy intended as the deviation of a solution value to the optimal solution that exists. Thus, to determine the performance of the algorithm in terms of accuracy, the costs generated by the algorithm are compared with the best-known solutions in available literature.

In the VRPTW context, the most common way to compare heuristics is the results obtained for Solomon's and Homberger's benchmark problems. These problems have a central depot, capacity constraints, time windows on the time of delivery.

Concerning the number of customers, they are taken in account the Solomon instances[1], benchmark files with 25, 50 and 100 customers;

These instances are divided into three categories:

- C-type, C1 and C2 having customers located in clusters;
- R-type, R1 and R2 having the customers located at random positions;
- RC-type, RC1 and RC2 having a mix of both random and clustered customers.

[1] Source: http://www.bernabe.dorronsoro.es/vrp/index.html?/Problem_Instances/CVRPTWInstances.html

The C1, R1 and RC1 problems have a short scheduling horizon, and (usually) require 9 to 19 vehicles. Short horizon problems have vehicles that have small capacities and short route times and cannot serve many customers at one time.

Classes C2, R2 and RC2 are more representative of "long-haul" delivery with longer scheduling horizons and fewer (2−4) vehicles.

Both travel time and distance are given by the Euclidean distance between points and the average speed of vehicles (AVG SPEED) is set to 60.

Notice that before we put the table with data found in testing, and then 2 comparison bar charts, one for the numbers of routes found, and one for the solution cost.

Note: it is assumed that a vehicle must not wait more than 10 minutes in the same node, before serving it.

4.3.1 SOLOMON 25 CUSTOMERS

Here there are the results found using the Solomon's file with 25 customers

	Clarke-Wright		Nearest Neighbour		Best Known Solution*	
Problem	#Vehicles	Distance	#Vehicles	Distance	#Vehicles	Distance
C101.txt	7	439	4	277	3	191,3
C102.txt	8	402	4	357	3	190,3
C103.txt	7	419	6	368	3	190,3
C104.txt	9	466	5	344	3	186,9
C105.txt	8	462	4	277	3	191,3
C106.txt	10	545	5	297	3	191,3
C107.txt	8	404	3	257	3	191,3
C108.txt	5	290	4	326	3	191,3
C109.txt	4	266	3	268	3	191,3
C201.txt	9	570	4	332	2	214,7
C202.txt	6	432	5	419	2	214,7
C203.txt	8	533	5	401	2	214,7
C204.txt	7	466	5	427	1	213,1
C205.txt	5	392	3	301	2	214,7
C206.txt	5	388	4	364	2	214,7
C207.txt	5	340	6	512	2	214,5
C208.txt	4	346	3	279	2	214,5
R101.txt	12	870	15	877	8	617,1
R102.txt	10	716	11	777	7	547,1

R103.txt	10	683	9	681	5	454,6
R104.txt	7	543	8	637	4	416,9
R105.txt	8	711	11	708	6	530,5
R106.txt	7	618	8	737	5	465,4
R107.txt	7	610	7	630	4	424,3
R108.txt	6	530	6	581	4	397,3
R109.txt	5	502	8	672	5	441,3
R110.txt	5	501	7	648	4	444,1
R111.txt	4	526	6	649	4	428,8
R112.txt	4	471	5	525	4	393,0
R201.txt	7	789	9	868	4	463,3
R202.txt	5	782	8	791	4	410,5
R203.txt	8	714	8	730	3	391,4
R204.txt	6	537	4	545	2	355,0
R205.txt	5	606	3	653	3	393,0
R206.txt	6	698	5	682	3	374,4
R207.txt	7	661	3	586	3	361,6
R208.txt	6	588	3	466	1	328,2
R209.txt	3	532	6	725	2	370,7
R210.txt	4	670	5	683	3	404,6
R211.txt	3	465	3	575	2	350,9
RC101.txt	7	715	5	603	4	461,1
RC102.txt	7	775	6	558	3	351,8
RC103.txt	6	657	7	642	3	332,8
RC104.txt	6	658	7	676	3	306,6
RC105.txt	7	684	9	800	4	411,3
RC106.txt	4	435	3	354	3	345,5
RC107.txt	4	393	4	553	3	298,3
RC108.txt	3	330	4	457	3	294,5
RC201.txt	7	935	11	1218	3	360,2
RC202.txt	4	786	11	1148	3	338,0
RC203.txt	7	822	6	797	3	326,9
RC204.txt	7	731	4	604	3	299,7
RC205.txt	6	833	11	1214	3	338,0
RC206.txt	2	676	12	1251	3	324,0
RC207.txt	4	648	10	1138	3	298,3
RC208.txt	3	334	8	855	2	269,1

Figure 12 Solomon 25 customers – Summing table

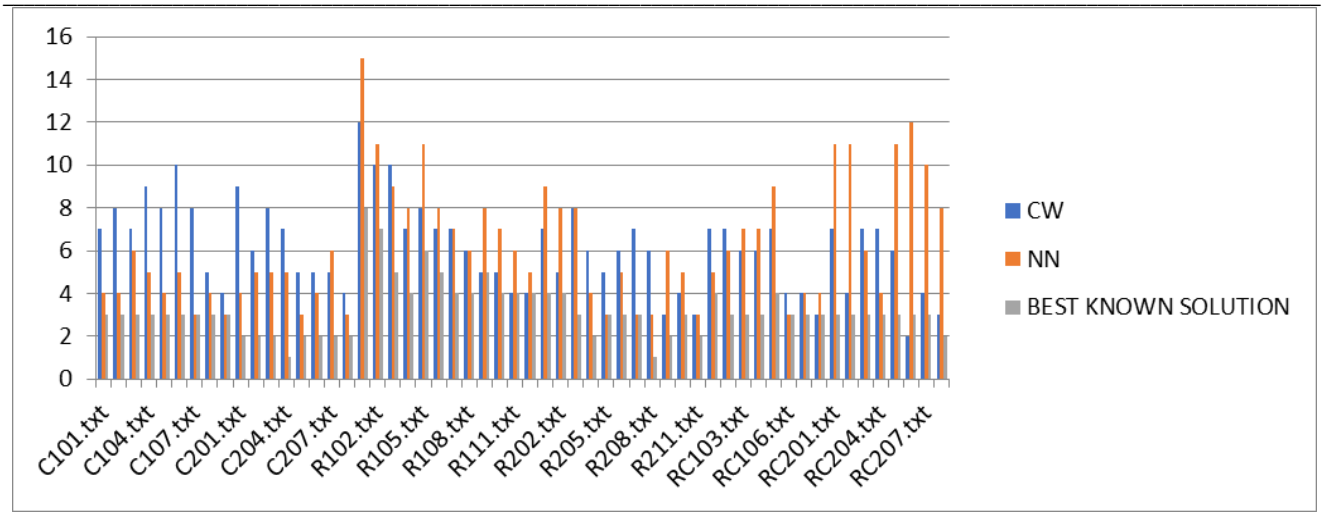


Figure 13 Solomon 25 customers - # of Vehicles

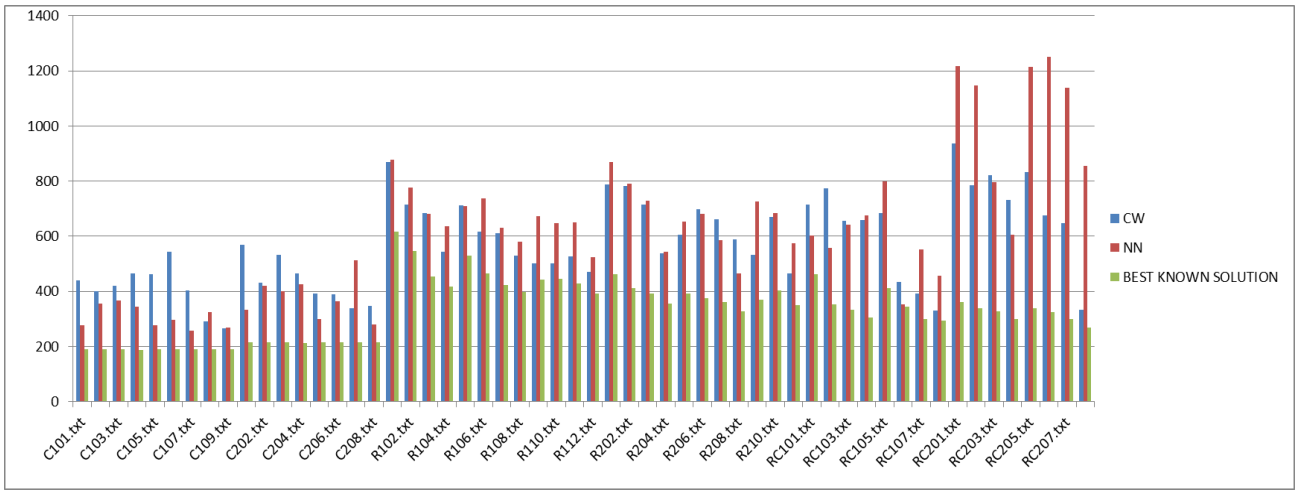


Figure 14 Solomon 25 customers – Solution cost

The NN's algorithm finds the worst solution with the R101.txt file. After that the algorithm gets a stable equilibrium around 10 vehicles used and 700 solution costs. The CW alternates quite good solutions (for example in C109.txt and RC108.txt) to very bad ones.

4.3.2 SOLOMON 50 CUSTOMERS

	Clarke-Wright		Nearest Neighbour		Best Known Solution*	
Problem	#Vehicles	Distance	#Vehicles	Distance	#Vehicles	Distance
C101.txt	14	962	5	484	5	362,4
C102.txt	16	1059	9	779	5	361,4
C103.txt	11	894	11	854	4	361,4
C104.txt	11	699	9	681	5	359
C105.txt	10	759	5	451	5	362,4
C106.txt	14	934	5	451	5	362,4
C107.txt	9	648	5	447	5	362,4
C108.txt	7	627	6	453	5	362,4
C109.txt	7	515	6	494	5	362,4
C201.txt	16	1064	7	542	3	360,2
C202.txt	14	1044	6	554	3	360,2
C203.txt	14	1068	10	988	3	359,8
C204.txt	11	835	9	819	2	353,4
C205.txt	7	697	7	546	3	359,8
C206.txt	9	844	9	624	3	359,8
C207.txt	8	742	9	698	3	359,6
C208.txt	5	614	6	535	2	350,5
R101.txt	21	1680	21	1511	13	1047
R102.txt	14	1321	18	1576	12	944,9
R103.txt	12	1205	15	1168	9	772,9
R104.txt	8	901	12	1078	6	631,2
R105.txt	14	1248	11	1192	10	906,6
R106.txt	10	1083	13	1206	8	793,6
R107.txt	11	1089	14	1147	7	720,4
R108.txt	7	868	8	923	6	618,2
R109.txt	10	980	12	1183	8	803,2
R110.txt	9	934	9	1009	8	724,9
R111.txt	10	990	10	1089	8	724,9
R112.txt	6	783	7	856	6	651,1
R201.txt	10	1353	14	1576	6	800,7
R202.txt	10	1227	10	1413	5	712,2
R203.txt	11	1144	9	1270	5	606,4
R204.txt	8	904	5	901	2	509,5
R205.txt	7	1132	6	1089	5	703,3
R206.txt	6	1027	7	1030	5	647
R207.txt	5	981	2	920	4	584,6
R208.txt	9	899	3	842	2	487,7
R209.txt	5	1007	5	1004	4	600,6
R210.txt	6	965	8	1172	5	663,4
R211.txt	5	832	5	975	3	551,3
RC101.txt	12	1454	14	1474	9	957,9

RC102.txt	11	1393	10	1164	8	844,3
RC103.txt	9	1118	10	1116	6	712,6
RC104.txt	8	903	9	960	5	546,5
RC105.txt	13	1263	13	1246	9	888,9
RC106.txt	9	1014	7	897	7	791,9
RC107.txt	8	867	7	824	6	664,5
RC108.txt	7	698	6	749	6	598,1
RC201.txt	8	1712	15	2015	5	684,8
RC202.txt	9	1667	18	2235	5	613,6
RC203.txt	6	1295	10	1369	4	555,3
RC204.txt	8	1071	4	755	3	444,2
RC205.txt	8	1452	14	1905	5	631
RC206.txt	5	1213	17	2016	5	610
RC207.txt	7	1157	10	1520	4	558,6
RC208.txt	3	334	12	1460	3	828,14

Figure 15 Solomon 50 customers – Summing Table

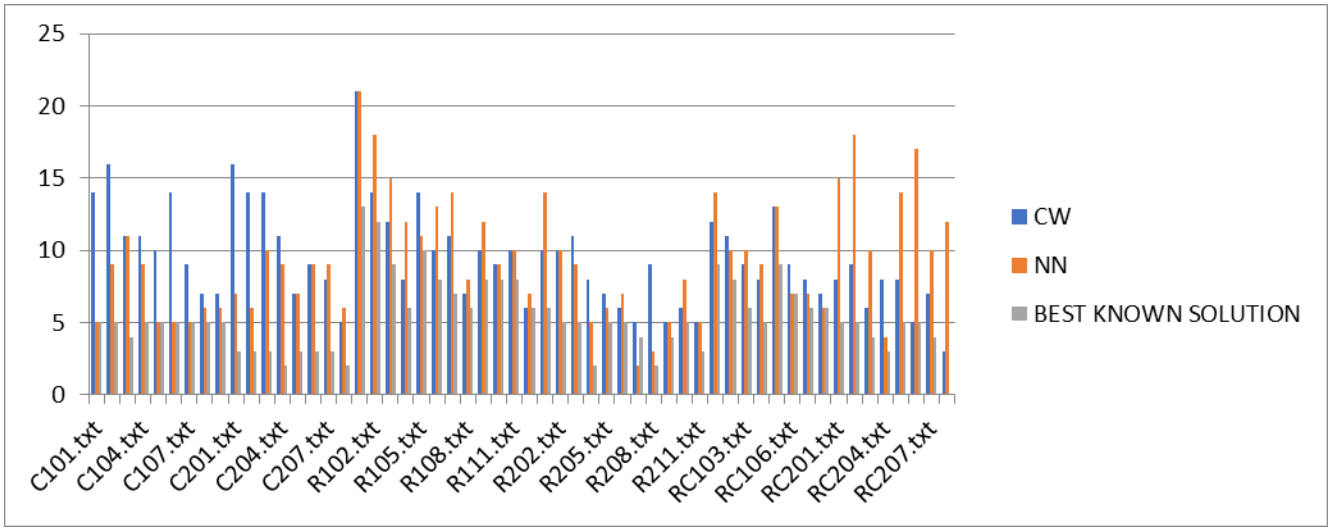


Figure 16 Solomon 50 Customers - # of Vehicles

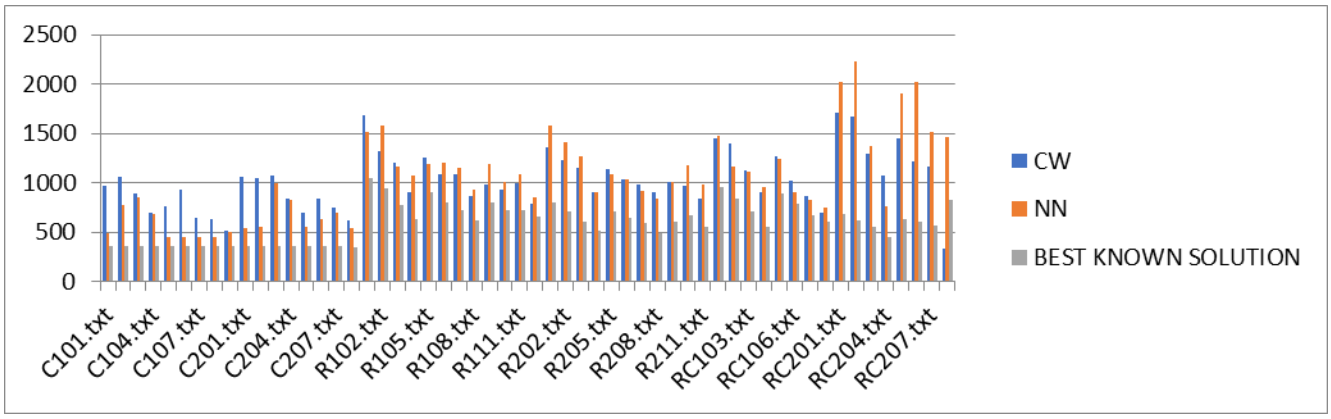


Figure 17 Solomon 50 customers – Solution Cost

It seems that the CW's algorithm performs very bad, in some cases the number of vehicles found are the triple than in the best known solution. Generally, the NN's algorithm finds good solutions, but in the "RC" categories file the performances are worse than with the CW's one.

4.3.3 SOLOMON 100 CUSTOMERS

	Clarke-Wright		Nearest Neighbour		Best Known Solution*	
Problem	#Vehicles	Distance	#Vehicles	Distance	#Vehicles	Distance
C101.txt	23	2309	11	894	10	828,94
C102.txt	18	2011	14	1341	10	828,94
C103.txt	19	1931	17	1734	10	828,06
C104.txt	16	1501	16	1744	10	824,78
C105.txt	16	1993	10	913	10	828,94
C106.txt	18	1775	11	1239	10	828,94
C107.txt	14	1758	10	1006	10	828,94
C108.txt	15	1447	11	1294	10	828,94
C109.txt	13	1152	11	1259	10	828,94
C201.txt	24	2504	3	565	3	591,56
C202.txt	20	2075	13	1513	3	591,56
C203.txt	21	1902	12	1567	3	591,17
C204.txt	14	1604	10	1246	3	590,6
C205.txt	11	1201	5	661	3	588,88
C206.txt	9	1206	5	725	3	588,49
C207.txt	13	1454	6	785	3	588,29
C208.txt	10	1093	4	725	3	588,32
R101.txt	26	2381	32	2433	19	1645,79
R102.txt	20	2018	29	2352	17	1486,12
R103.txt	19	1787	26	2070	13	1292,68
R104.txt	14	1395	20	1594	10	982,01
R105.txt	22	1848	18	1794	14	1377,11
R106.txt	14	1556	20	1906	12	1252,03
R107.txt	13	1479	15	1631	10	1104,66
R108.txt	12	1331	13	1411	9	960,88
R109.txt	16	1502	17	1490	11	1194,73
R110.txt	15	1423	15	1544	10	1118,84
R111.txt	15	1380	17	1654	10	1096,72
R112.txt	12	1204	11	1234	9	982,14
R201.txt	11	2107	14	2199	4	1252,37
R202.txt	14	1993	19	2212	3	1191,7
R203.txt	13	1734	13	1800	3	939,5
R204.txt	13	1437	8	1398	2	825,52
R205.txt	10	1522	10	1784	3	994,43
R206.txt	8	1406	7	1530	3	906,14

R207.txt	9	1384	13	1748	2	890,61
R208.txt	12	1318	6	1211	2	726,82
R209.txt	10	1465	9	1448	3	909,16
R210.txt	12	1530	12	1720	3	939,37
R211.txt	10	1228	6	1308	2	885,71
RC101.txt	17	2122	21	2284	14	1696,94
RC102.txt	17	2124	19	2134	12	1554,75
RC103.txt	14	1806	15	1730	11	1261,67
RC104.txt	12	1497	18	1912	10	1135,48
RC105.txt	20	2162	21	2244	13	1629,44
RC106.txt	15	1654	16	1748	11	1424,73
RC107.txt	14	1607	13	1598	11	1230,48
RC108.txt	12	1301	13	1469	10	1139,82
RC201.txt	11	2342	21	2882	4	1406,94
RC202.txt	12	2187	14	2486	3	1365,65
RC203.txt	10	1865	14	2111	3	1049,62
RC204.txt	11	1619	9	1486	3	798,46
RC205.txt	14	2414	14	2676	4	1297,65
RC206.txt	10	1940	22	2607	3	1146,32
RC207.txt	9	1697	16	2484	3	1061,14
RC208.txt	7	1097	7	1462	3	828,14

Figure 18 Solomon 100 Customers – Summing table

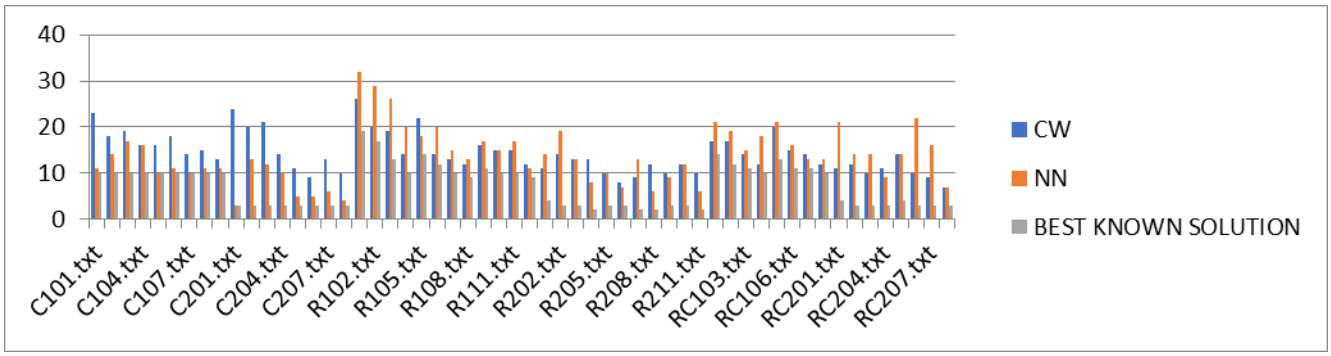


Figure 19 Solomon 100 customers - # of Vehicles

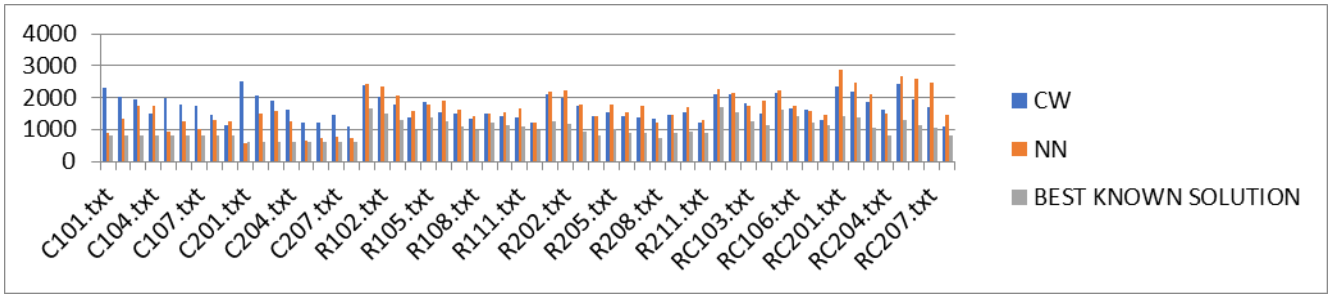


Figure 20 Solomon 100 customers – Solution cost

In the 100 customers' case, it looks like the NN's algorithm follows better the solution found in literature than the CW's one, even there are files in which we find exceptions (for example R102.txt and R202.txt). In the graph it is shown that the CW's algorithm, especially in the "C"'s files, overcomes the solution of 10 times.

4.3.4 COMPARING SOLUTION FOUND WITH WAITING TIME = 0

As explained by the title, now we want to set that the deliveries can't wait in the site to make a service, so the waiting time will be set to 0. We expect that the solution may get worse during this test, but how much they do?

Note: For this testing, we consider only the number of vehicles.

4.3.4.1 SOLOMON 25 CUSTOMERS

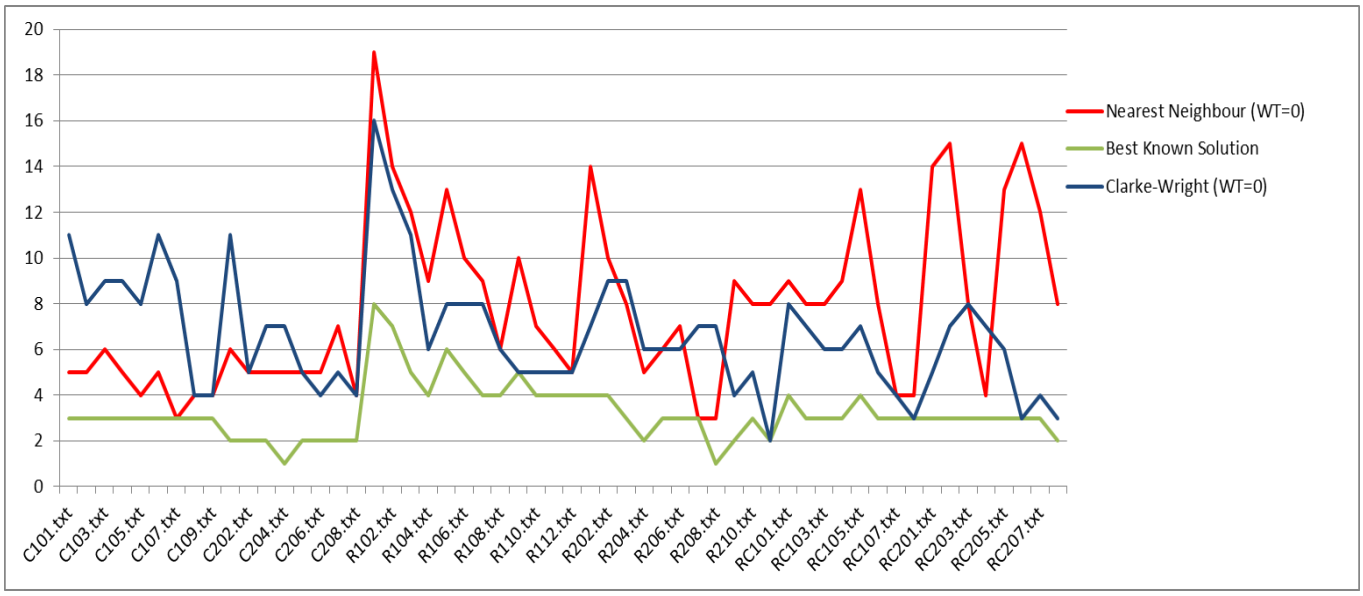


Figure 21 Solomon 25 customers – # of Vehicles with WT=0

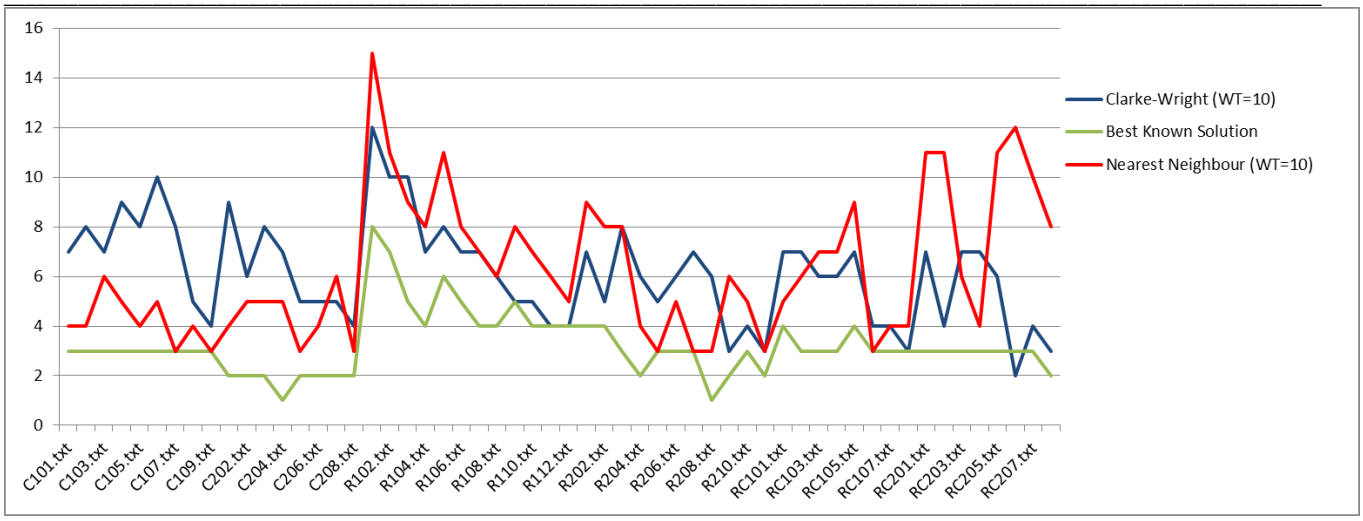


Figure 22 Solomon 25 customers – # of Vehicles with WT=10

From these graphs, we can say the NN's algorithm performs better in the "C"'s cathegory file than the CW, and the second one in the other two cathegories. Using the WT=0 setting, we can notice that the NN and the CW's graphs are shifted upwards, sometimes of more than 5 vehicles.

4.3.4.2 SOLOMON 50 CUSTOMERS

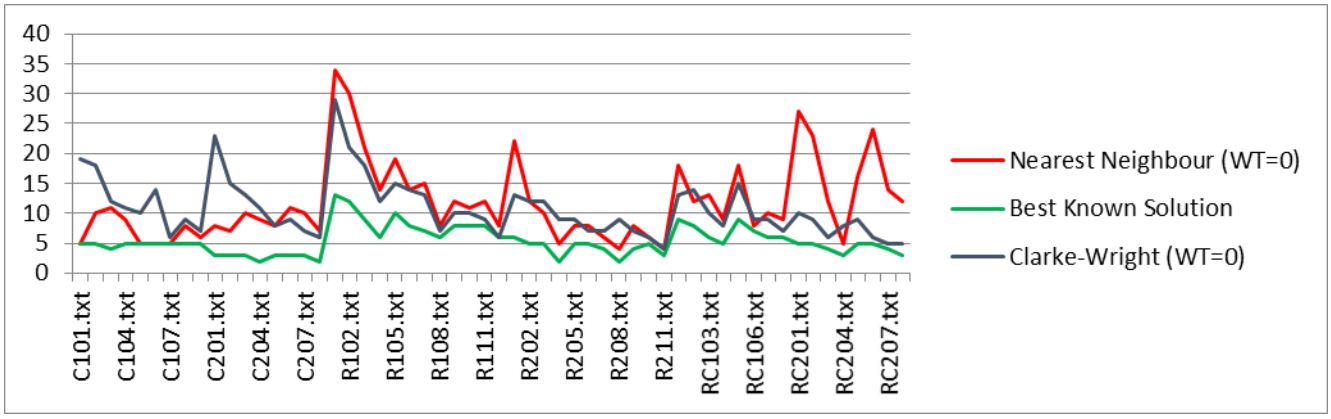


Figure 23 Solomon 50 customers – # of Vehicles with WT=0

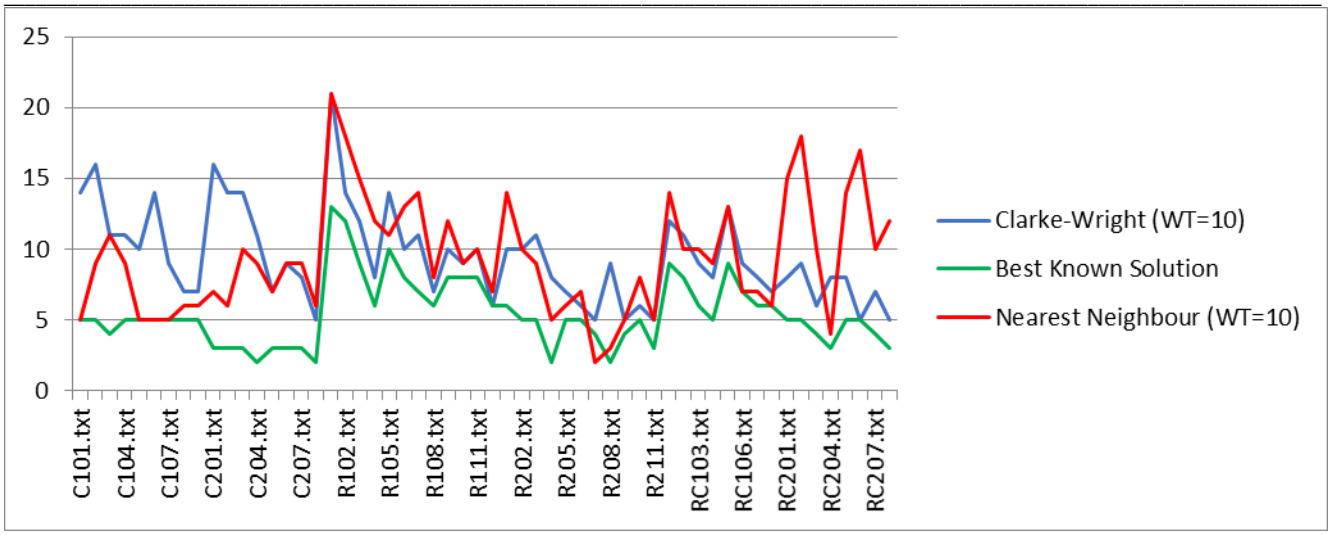


Figure 24 Solomon 50 customers – # of Vehicles with WT=10

From the testing with the 50 customers, the solution found with WT=0 with the two algorithms are quite similar, and they have almost same trend; that is a peculiar situation, because in WT=0 we have a quite oscillating results with the NN's method, and a more stable solution in CW's one.

4.3.4.3 SOLOMON 100 CUSTOMERS

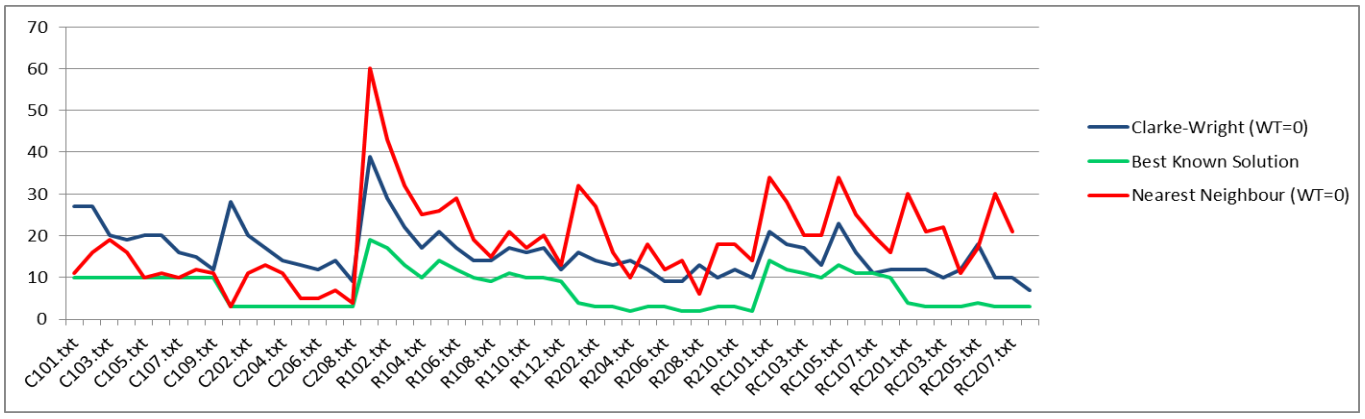


Figure 25 Solomon 100 customers – # of Vehicles with WT=0

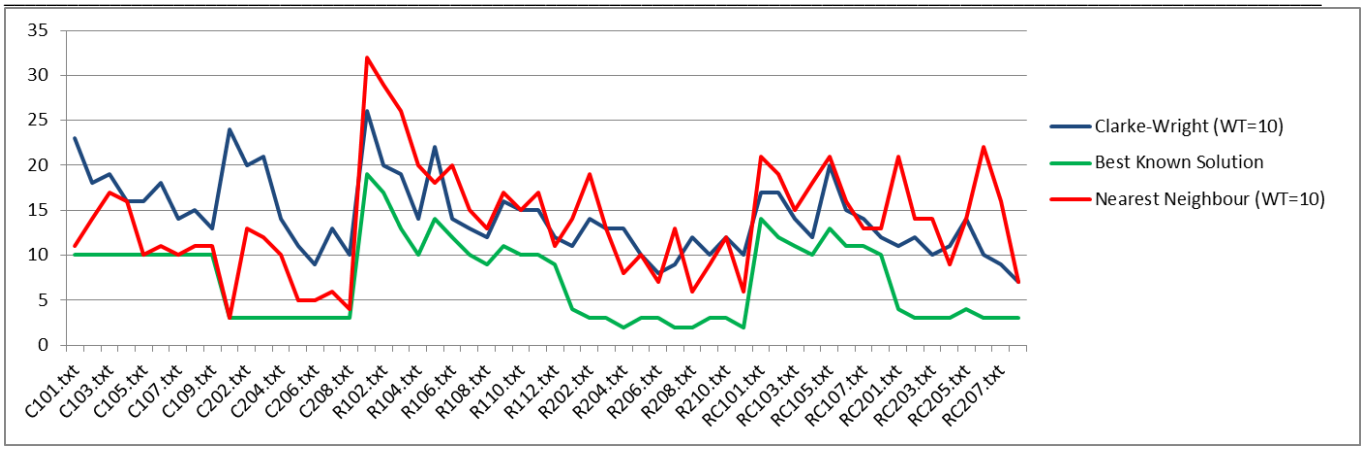


Figure 26 Solomon 100 customers – # of Vehicles with WT=10

In this graph, there is the same trend from the two different settings: as usual, the NN finds a quite oscillating situation, while the CW finds quite stable solutions.

5 RECOMMENDED FUTURE DEVELOPMENT

Although the application developed in this thesis is able to complete the tasks it was designed for, it still has plenty of room for improvements. In the next sections the author proposes some of these suggestions.

5.1 ROUTE OPTIMISATION

The most popular classical heuristics are naturally divided into two groups, *constructive heuristics* and *local search heuristics*.

Both the implemented methods Clark & Wright and Nearest Neighbour belong to constructive category and are purely greedy algorithms (although the Clark & Wright foresees a controlled randomization in the greedy selection rule, to allow the algorithm to generate a different solution at each iteration), as such a great enhancement of the tool could be adding to the first solution other code parts, to produce better results.

A post-optimisation procedure called “Route Optimizer”, based on improvement *heuristic* methods could be applied to each route of the best solution so far using or-opt. A such method would involve removing strings of three, two or one consecutive customers and reinserting them either in the same route or another route. Or-opt

exchanges are actually a subset of 3-opt exchanges and perform well for problems involving time windows.

For example, Laporte (2007) describes two types of improvement heuristic algorithms that can be applied to VRP solutions:

- Intra-route;
- Inter-route.

Intra-route heuristics post-optimize each route by using improvement heuristic e.g. 2-opt or 3-opt as presented by Kernighan and Lin (1973).

Inter-route heuristics consist of moving vertices to different routes.

Laporte also states that the most common moves are simple transfers from one route to another and transfers involving several routes and vertex exchanges between two or more routes.

5.2 GOOGLE MAPS INTEGRATION

The system could interface the Google Maps API for sites' compilation, with reference to latitude and longitude.

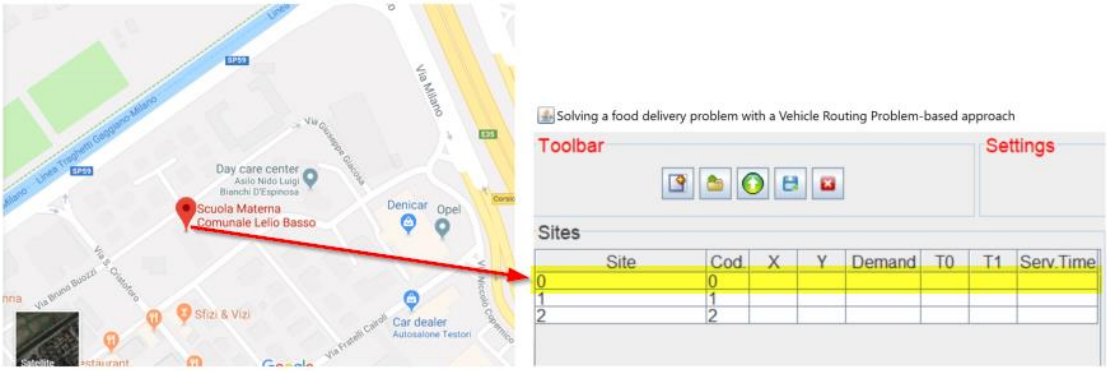


Figure 27 Adding nodes from Google Maps sheet

This interfacing will allow also a more accurate computing of distance between delivery points, against Euclidean distances foreseen.

Once the routes are computed then it could be available a Google map showing them together with the delivery sites.

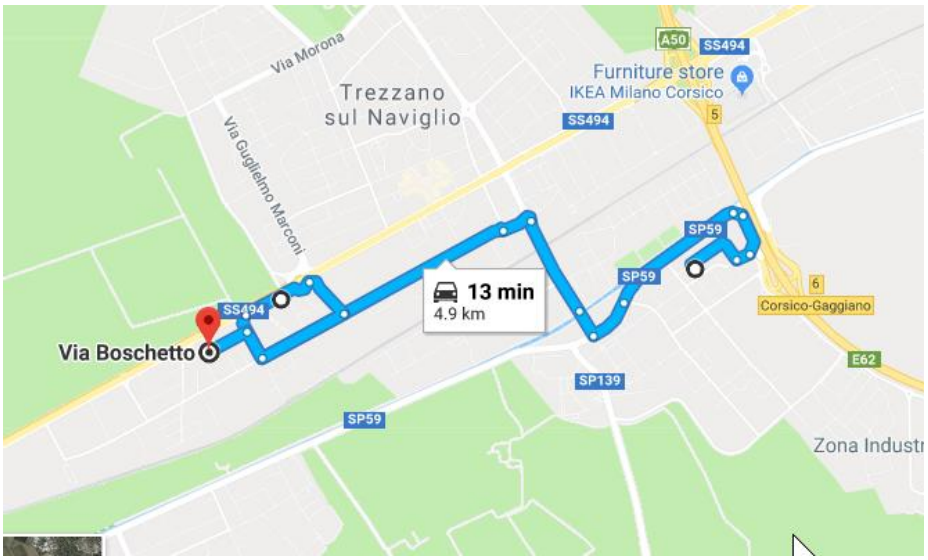


Figure 28 Route presentation with Google Maps

5.3 TIME FORMAT

Another issue, concerning display of computed routes, is the format used when printing times. Actually, the application always displays them as 'mmm', i.e. number of minutes.

It could be nice to have the time displayed in standard format 'hh:mm:ss'. This would lead to a better understanding of the time scheduling that the application gives to the solution.

6 CONCLUSIONS

In this thesis, it was addressed the capacitated vehicle routing problem and extended it to a capacitated vehicle routing problem with time windows. A practical vehicle routing problem faced by Sotral was presented. The problems were modelled to improve the distribution efficiency and to increase customer service quality.

According to the fact that a VRP is known to be NP-hard, two heuristics methods have been adopted to solve these problems. The corresponding solutions are highly pragmatic and realistic.

Indeed, based on the results of application it can be concluded that:

- the method used by the company Sotral is certainly effective for finding a solution to the problems of Vehicle Routing, but it is not efficient. The time of execution of solving the problem is inversely proportional to the cube of the number of nodes inserted and as such the time of processing could be get unacceptable;
- the proposed tool is able:
    - to improve the current routes of Sotral delivery system, with using, on average, 35% less of vehicles and spending 33% less on travel time;
    - to produce optimal routes in a minimal amount of computational time (for a graph of 23 nodes, the average time of completion of the task was of 5 seconds, against the 1 day and a half of the Sotral's method).

Then the thesis has succeeded in demonstrating that by applying theory concepts regarding the operative research on a real type problem, it is possible to greatly improve the user search performances and to perform the same actions in a very short time.

The algorithms are relatively simple and can easily be replicated and adapted to new instances. Regarding computation time, the Clarke & Wright algorithm was not particularly competitive for number of nodes higher than 600, however, an average of 30 seconds for iteration was reasonable considering the nature of the problem.

In summary, despite improving on some of the best-known solutions, improvements were not consistent across all instances. Overall, the implemented algorithm cannot be considered as two of the best methods. It appears that simple implementations cannot always compete with the more cumbersome approaches.

Now there are potential for improvement. Section Recommended future development covers suggested recommendations, where improvements are possible at the

expenses of simplicity, given that a minimum level of complexity is necessary to improve on the algorithms in its current form.

7 APPENDIX A: FLOW CHARTS

7.1 FLOW CHART OF SOTRAL'S ADOPTED METHOD

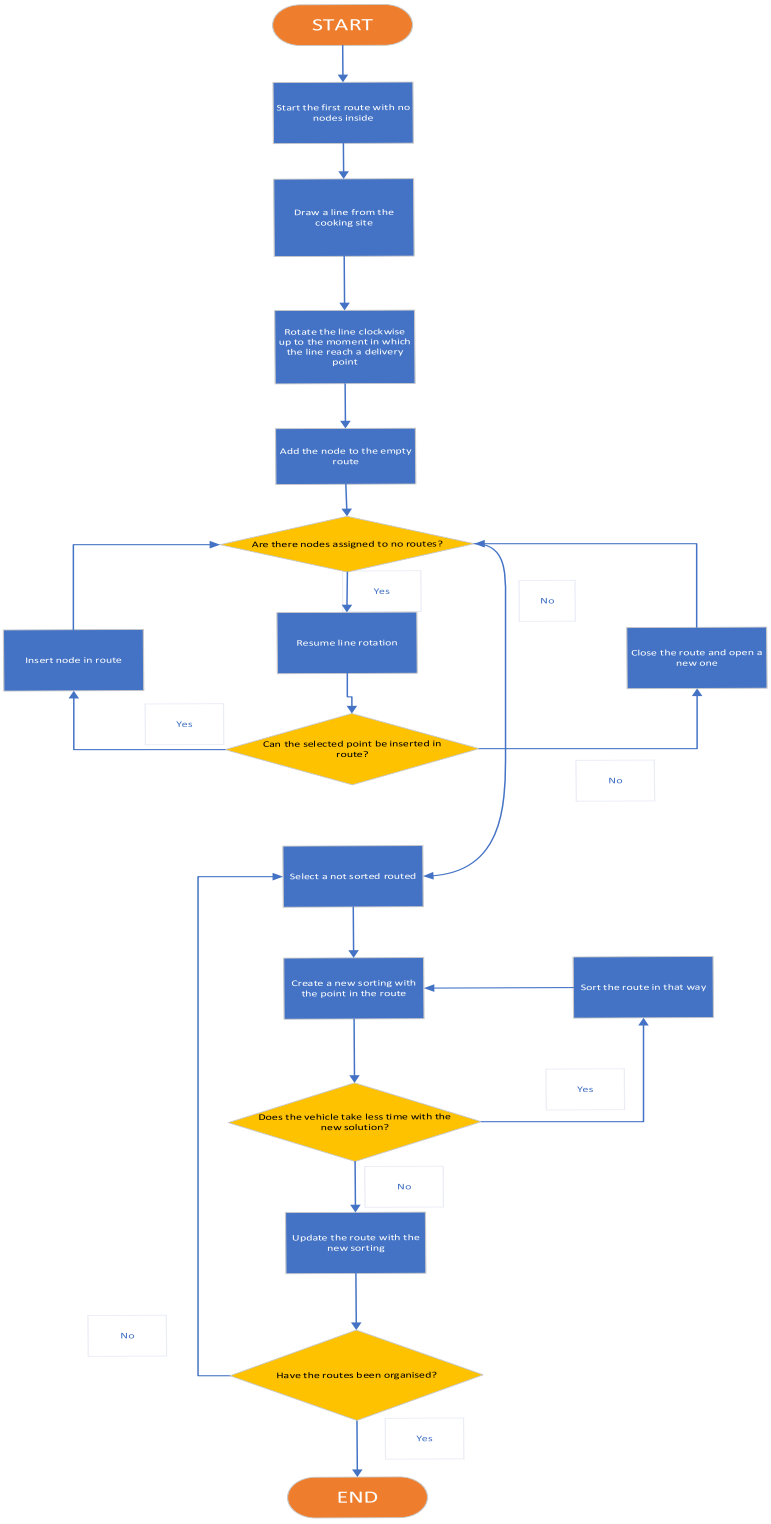


Figure 29 Flow Chart Adopted method Sotral

7.2 FLOW CHART OF THE CLARKE WRIGHT ALGORITHM

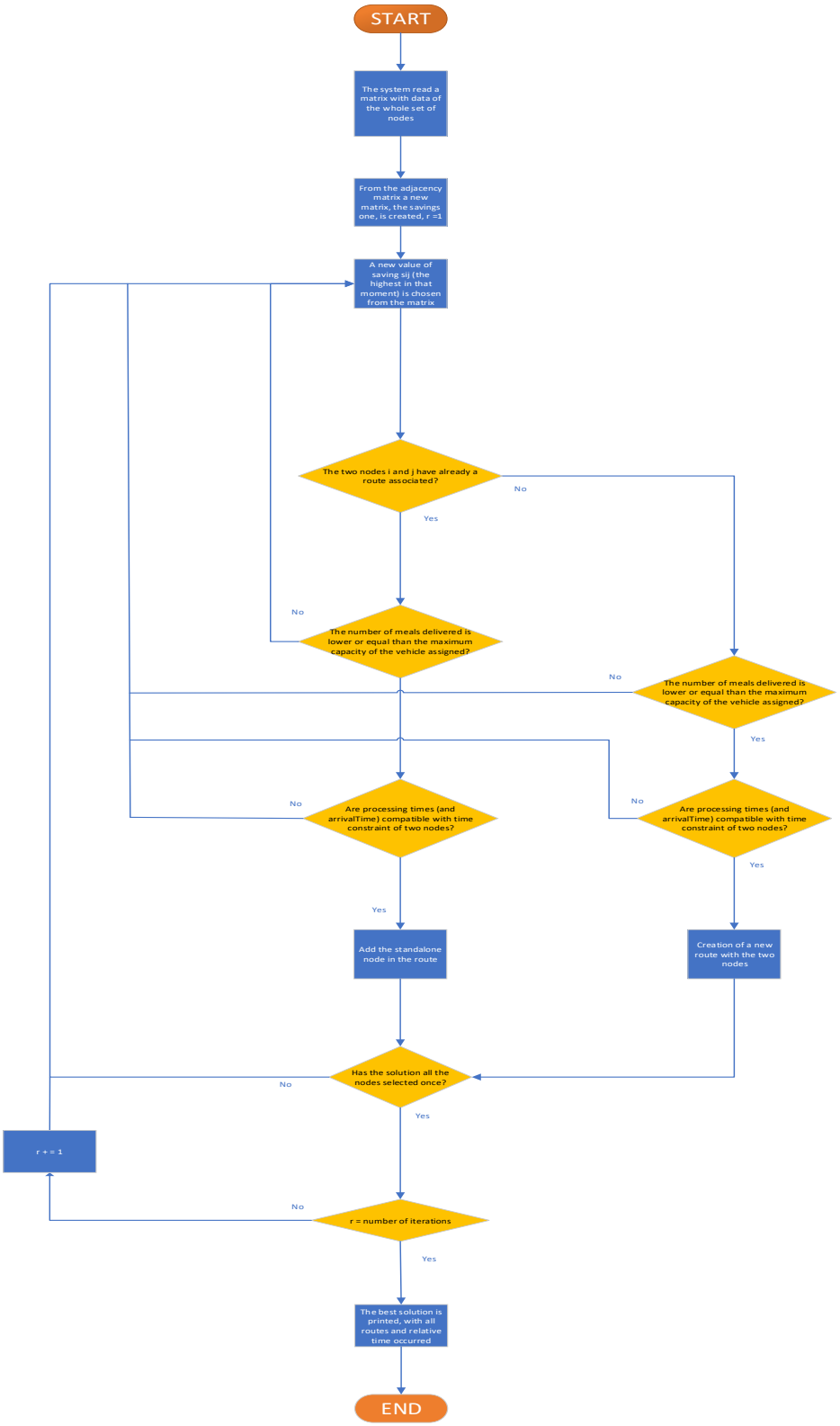


Figure 30 Flow Chart of the Clarke Wright algorithm

7.3 FLOW CHART OF THE NEAREST NEIGHBOUR ALGORITHM

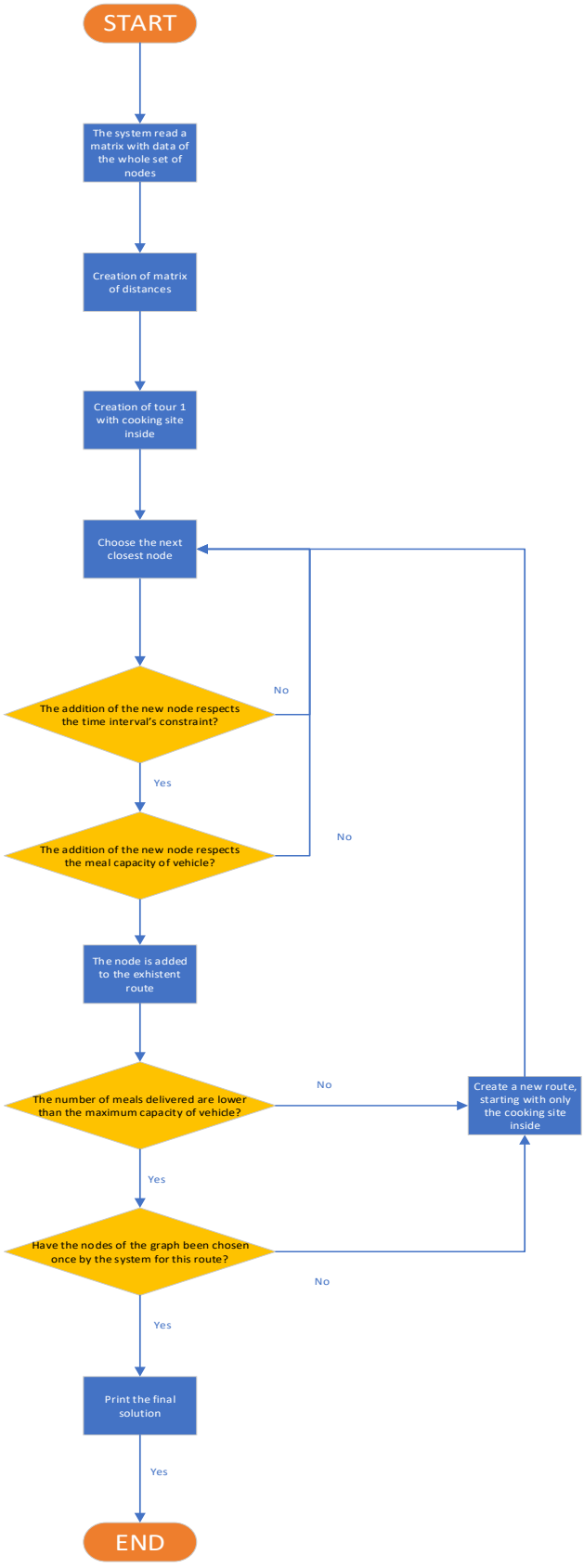


Figure 31 Flow Chart of the Nearest Neighbour algorithm

8 APPENDIX B: SOFTWARE USER MANUAL

Solving a food delivery issue with a Vehicle Routing Problem-based approach

Software User Manual

V1.0 March 2019

8.1 INTRODUCTION

This document is a support tool to exploit at maximum of the possibilities the experience during utilization of the Java application. In the next chapters, we will see all the reference documents used to create this guide, the theory and the steps taken from literature of the algorithms from which the writer got the idea, how the Graphical User Interface looks like, and all the buttons and the forms needed to fill before launching the application.

8.2 REFERENCE DOCUMENTS

ID	Reference
RD1	G. B. Dantzig, J. H. Ramser, *The truck dispatching problem,* Management Science, Vol. 6, No. 1, pp. 80-91, 1959
RD2	N. Christofides, S. Eilon, *An algorithm for the vehicle dispatching problem,* Operational Research, Vol. 20, No. 3, pp. 309-318, 1969
RD3	M. Solomon, *Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints*, INFORMS, Vol. 35, No. 2, pp. 254-265, Mar. 1987
RD4	M. Abdul-Niby, L. Caccetta, M. Alameen, *An Improved Clarke and Wright Algorithm to Solve the Capacitated Vehicle Routing Problem,* Engineering, Technology & Applied Science Research, Vol. 3, No. 2, 413-415
RD5	Pichpibul, T., Kawtummachai, R., 2012. An improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem. ScienceAsia 38, 3, 307–31

8.3 IMPLEMENTED METHODS

8.3.1 TIME ORIENTED CLARK&WRIGHT

This algorithm has an initial setup in which every route has a single node inside, so the number of routes is equal to the number of nodes of graph. In the parallel version we have the addition, in each repetition, of a link between two delivery points, before belonging to 2 different routes, and to do this, we consider the savings due to link two different nodes:

$$sav_{ij} = d_{i0} + d_{0j} - \mu d_{ij} \quad \mu \geq 0$$

For example, if μ=1, sav_{ij} is the distance (or time) saved when a delivery in node *i* and *j* is fulfilled in a unique route instead of serving them separately.

According to the presence of time intervals, it's needed to decide the route orientation. Two partial routes that have as final customers *i* and *j* have compatible orientations if the node *i* is the first (last) in its route and *j* is the last (first) in its route: the admissible links start from the last customer of a route to the first one of another route. In each step of the algorithm, it's necessary check that time constraints (linked to time intervals) are not violated.

The heuristic algorithm described in this way could find a good link in two nodes very close in space, but far away in time. Connections like these will lead to waiting times that will increase costs: the vehicle, in fact, during waiting that node is ready to be served, may work in other places.

8.3.2 TIME ORIENTED, NEAREST NEIGHBOUR

This second heuristic algorithm is a sequential one. It starts each route finding the "nearest" non-assigned customer to depot. In each repetition we search for the "nearest" node from the last customer inserted. The research is done among all customers that can (respecting the capacity and time constraints) be added to the tail of the considered route. A new route starts when the research fails or there aren't no more nodes to insert.

This kind of approach want to consider both the nodes "near in time" and "near in space". And the last customer of a route with all the others linked with, and whatever external node *j* that may be visited in the next steps of the method.

The parameter C_{ij} will take in account of:

- distance from two nodes (d_{ij});
- difference between time spent to finish the service in node i and the starting time in customer *j* (T_{ij});
- the urgency to deliver to customer *j* *j*(v_{ij}), expressed like the remaining time before the last service may start.

Formally:

$T_{ij} = b_j - (b_i + s_i)$,

$v_{ij} = l_j - (b_i + s_i + t_{ij})$,

and

$c_{ij} = \delta_1 d_{ij} + \delta_2 T_{ij} + \delta_3 v_{ij}$

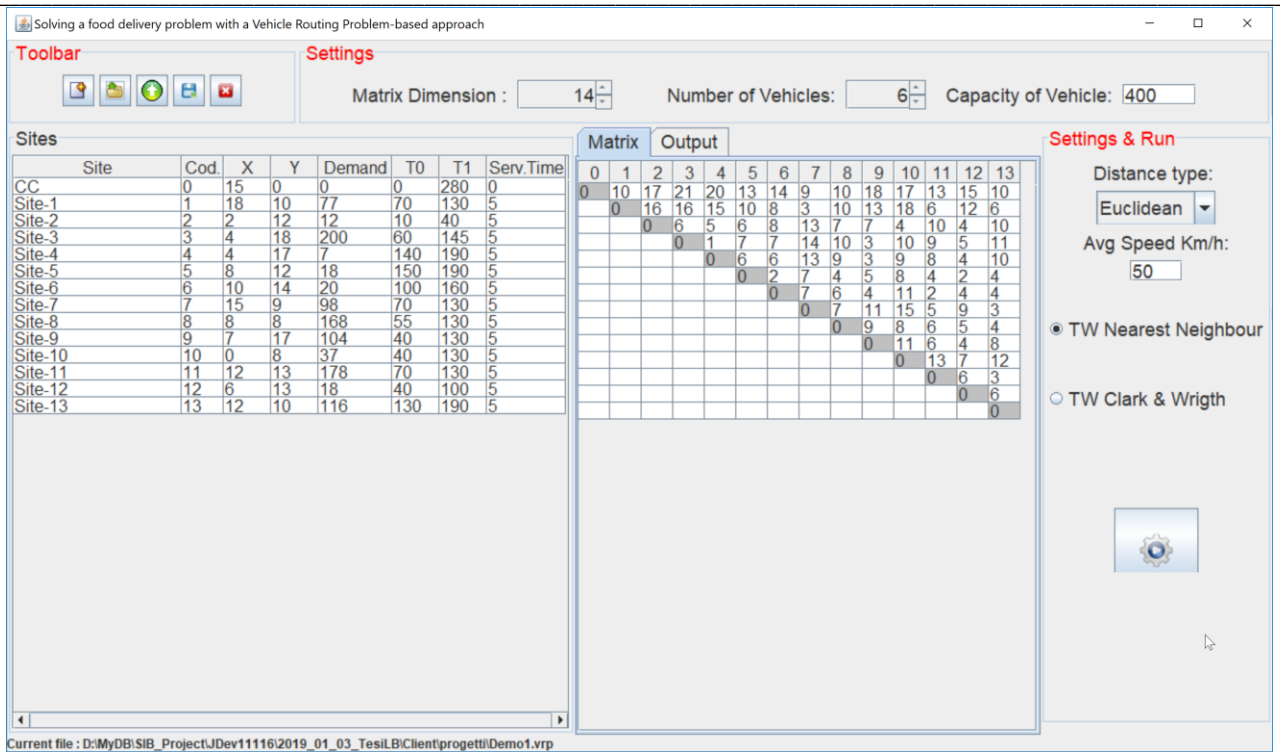
it is defined by weights satisfying $\delta_1 + \delta_2 + \delta_3 = 1$, $\delta_1 \geq 0$, $\delta_2 \geq 0$, $\delta_3 \geq 0$.

8.4 GRAPHICAL USER INTERFACE

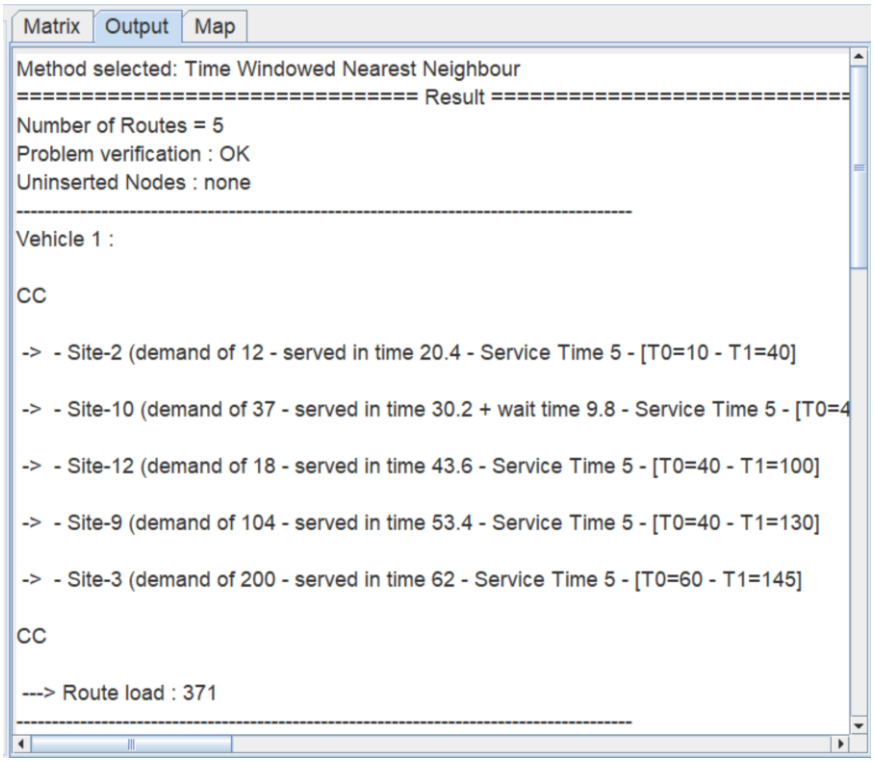
The application is developed in the programming language Java. The design of graphical user interface is done using Eclipse for Java Developer 2018-12.

The following figures depict the main panel of the application, where the user is allowed to:

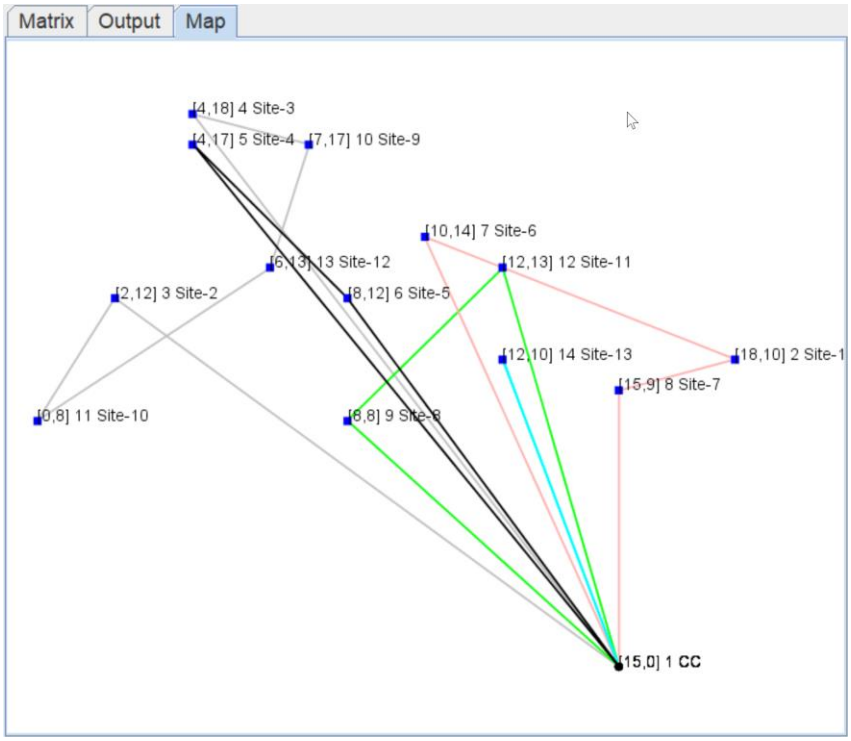
- create a new Vehicle Routing Problem by scratch;
- save the current problem as named project;
- load a saved project;
- select the method to execute;
- run the code concerning the method selected and view the results.



Then the user, moving to the panel Output, can display the result:

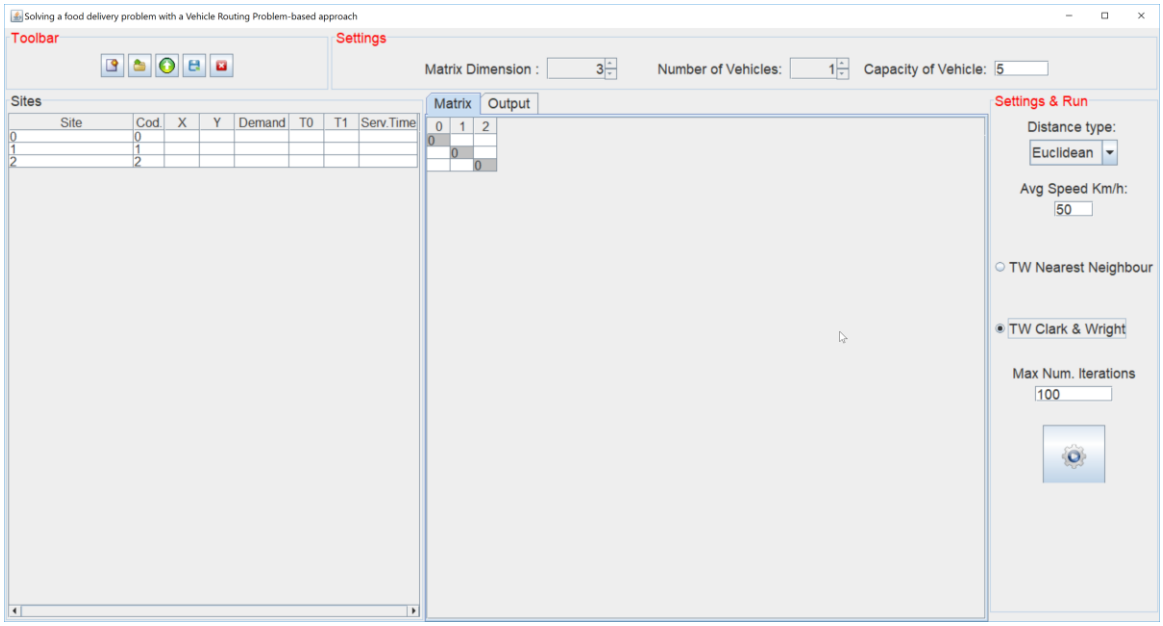


And, furthermore, moving to the panel Map the user can display the graphical views of computed routes:



8.5 OPERATIONAL INSTRUCTION

Running the application, the main panel shall be displayed:



The main panel is split in several frames:

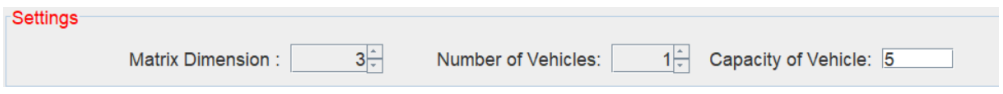
- Toolbar
- Settings
- Sites panel
- Matrix panel
- Settings & Run

8.6 TOOLBAR

This frame foreseen the following push buttons:

1. **New project** will reset the panels sites and Matrix to the initial stage;
2. **Open project** will allow to select, from the file system, a project previously saved; the panels Setting, Sites and Matrix shall be populated with the data specified in the opened project.
3. **Upload Solomon's/Homberger's VRPTW benchmark file**; will allow to select, from the file system, a standard benchmark file; the panels Setting and Sites shall be populated with the data specified in the uploaded file while the panel Matrix shall be populated only if the size of the matrix is <= 100;
4. **Save project**, the data filled in the panels Setting, Sites ans Matrix shall be saved on the file system;
5. **Exit**, exit from application.

8.7 SETTINGS



This frame foreseen the following push buttons:

1. Matrix Dimension : 3, in case of manual insertion of data, the user, acting on the spinner, is able to select the dimension of the wished size of the matrix;
2. Number of Vehicles: 1, this widget allow the user, acting on the spinner, to set the number of vehicles being used in the problem;
3. Capacity of Vehicle: 5, this widget allow the user, acting on the spinner, to set the capacity of each vehicles being used in the problem.

8.8 SITES PANEL

Sites

Site	Cod.	X	Y	Demand	T0	T1	Service Time
0	0						
1	1						
2	2						

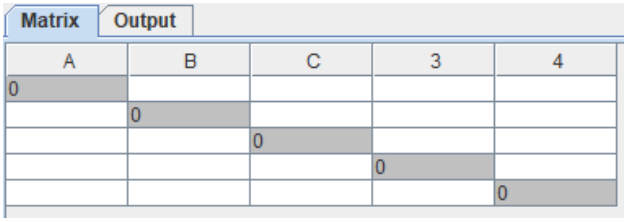
This panel foreseen the following fields:

1. **Site**, name/description of the site, e.g Depot, Customer1, Customer2 etc.; this field is updatable by the user;

   Note: the software assumes that the Depot/Whareouse is defined at first position of the table, i.e. Cod = 0.

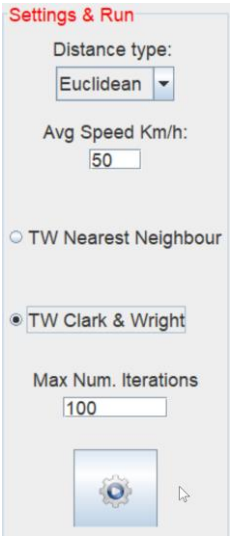
2. **Cod**., internal code assigned automatically to the site; this field is NOT updatable by the user;
3. **X**, assuming a cartesian X,Y asxis base, this field value states the X coordinate (position) of the site on this plan;
4. **Y**, the Y coordinate (position) of the site on the plan;
5. **Demand**, amount of the number of items requested in the site;
6. **T0**, begin time of time window range in which the customer shall be served;
7. **T1**, end time of time window range in which the customer shall be served;
8. **Service Time**, number of minutes need to unload the demanded number of items in the site.

8.9 MATRIX PANEL

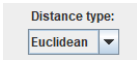


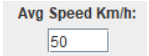
This panel will show the distance between sites. These values are computed automatically once the run method function is performed, i.e. the user should not insert data in this panel.

8.10 SETTIGNS & RUN

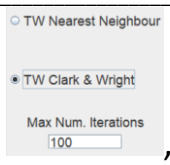


This frame foreseen the following push widgets:

- 

, drop box list which allows the user to select type of distance to be adopted in execution of method; as a default, the distances are considered Euclidean;
- 

, it allows the user to set the average speed of vehicles to be adopted in execution of method; the default is set to 50 Km/h;

- 

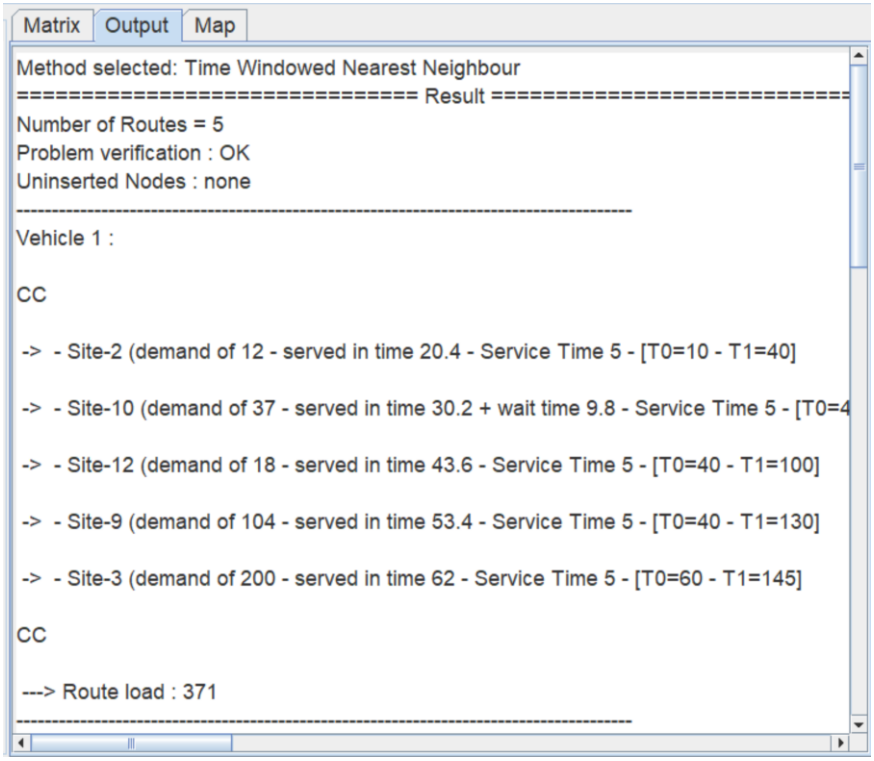
, radio button which allows the user to select the method to be executed; if Clark&Wright method the user is allowed to set the Max num of iteration to be performed as well.

8.11 START & OUTPUTS

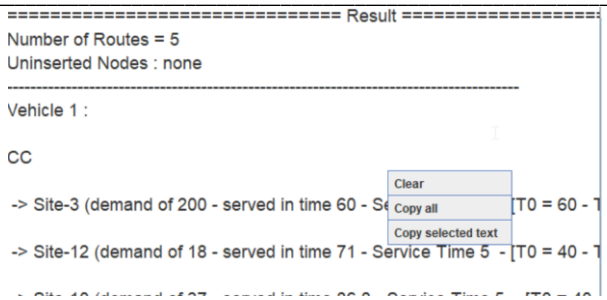
To start the process, the user has to click on  push button of Settings & Run panel. After the cumputation has teminate, the result shall be displayed in Output and Map tabs.

8.12 OUTPUT TAB

The text area item shall list every vehicle and its route cumputed about the best solution found.

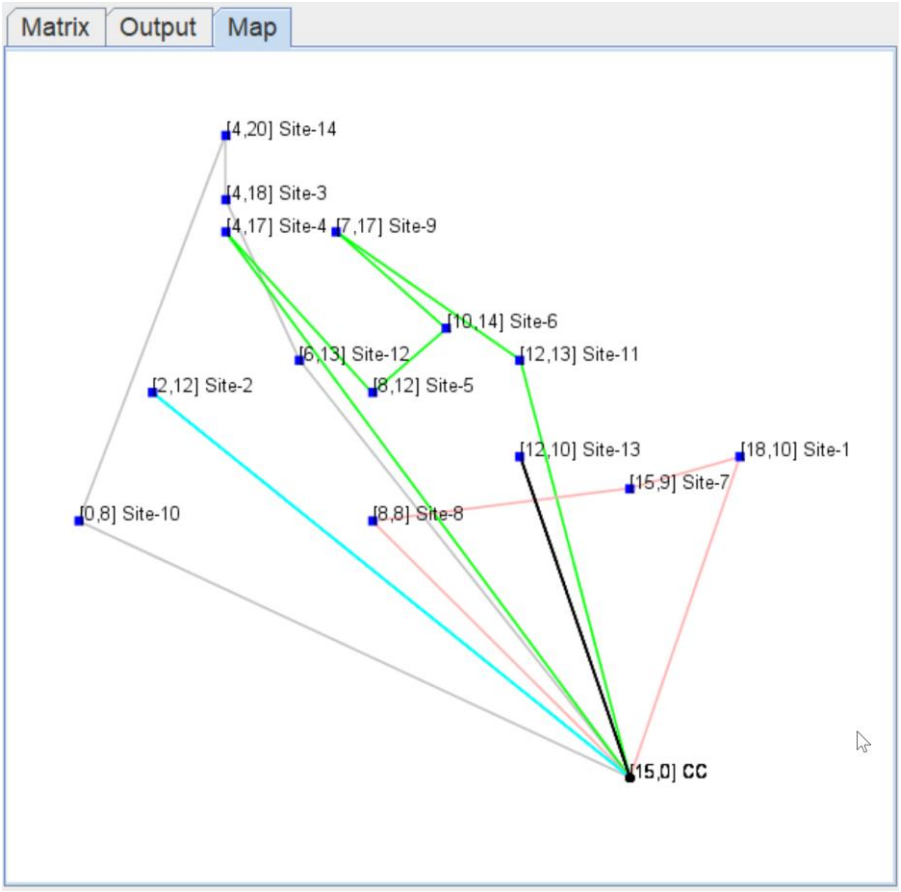


The text in the output tab can be selected, copied by mean a dedicated context mouse right click menu.



8.13 MAP TAB

The graph area shall display the routes (in different color) cumputed compounding the best solution found.



The produced graph can be:

- zoomed in/out by the mean of mouse wheel;
- moved in every direction by dragging the mouse.

9 REFERENCES

[1] G. B. Dantzig, J. H. Ramser, *The truck dispatching problem,* Management Science, Vol. 6, No. 1, pp. 80-91, 1959

[2] N. Christofides, S. Eilon, *An algorithm for the vehicle dispatching problem,* Operational Research, Vol. 20, No. 3, pp. 309-318, 1969

[3] N. R. Achuthan, L. Caccetta, *Integer linear programming formulation for a vehicle routing problem,* European Journal of Operational Research, Vol. 52, No. 1, pp. 86-89, 1991

[4] M. Abdul-Niby, L. Caccetta, M. Alameen, *An Improved Clarke and Wright Algorithm to Solve the Capacitated Vehicle Routing Problem,* Engineering, Technology & Applied Science Research, Vol. 3, No. 2, 413-415

[5] Bernard Korte, Jens Vygen, *Ottimizzazione Combinatoria, Teoria e Algoritmi,* Bernhard Korte, 1985

[6] G. Zotteri, P. Brandimarte*, Logistica di Distribuzione, CLUT 2004*

[7] M. Solomon, *Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints*, INFORMS, Vol. 35, No. 2, pp. 254-265, Mar. 1987

[8] Pichpibul, T., Kawtummachai, R., 2012. *An improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem*. ScienceAsia 38, 3, 307–318

[9] J. F. Cordeau, G. Laporte, M. Savelsberg, D. Vigo*, Handbook in OR & MS*, Canada Research Chair in Logistics and Transportation, Vol. 14, 2007

[10] J. Càceres Cruz, D. Riera, R. Buil, A. Juan, *Applying a Savings Algorithm for Solving a Rich Vehicle Routing Problem in an Urban Real Context*, Lecture Notes in Management Science, Vol. 5, 84-92, 2013

[11] Bräysy O. and Gendreau M. (2001b). *Genetic Algorithms for the Vehicle Routing Problem with Time Windows*. Internal Report STF42 A01021, SINTEF Applied Mathematics, Department of Optimization, Norway.

[12] Laporte, G. (2007). *What you should know about the vehicle routing problem*. Naval Research Logistics 54, 811-819.

[13] Kernighan, B. W. and S. Lin (1973). *An effective heuristic algorithm for the traveling-salesman problem*. Operations Research 21, 498-516.

[14] Glover, F. and M. Laguna (1997). *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers.

[15] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan, Ann Arbor.

[16] Nasser A. El-Sherbeny, *Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods,* Journal of King Saud University, 22, 123-131

For the best-known solution in each Solomon's file:

[17]http://www.bernabe.dorronsoro.es/vrp/index.html?/Problem_Instances/CVRPTWInstances.html