Strategies for Problem Solving: the Capacitated Vehicle Routing Problem Case Study

António Rodrigues (up200400437@fe.up.pt)

***Abstract*—The objective of this work is to gain hands-on understanding of several heuristics and metaheuristics, essencial tools for problem solving with application in a multitude of research areas in science and engineering. As a case study, we consider the Capacitated Vehicle Routing Problem (CVRP) – an important subject in the areas of Operations Research related with physical distribution and logistics – applying constructive heuristics, local search procedures and metaheuristics and comparing the provided solutions. This work describes the implementation of both parallel and sequential versions of the Clarke and Wright Savings (CWS) algorithm (a clearly constructive heuristic), several local search heuristics (2-opt and 1-interchange), as well as custom versions of the Simulated Annealing (SA) algorithm and Genetic Algorithm (GA) as examples of metaheuristics. The provided implementations are tested against several datasets.**

I. Introduction

The Vehicle Routing Problem (VRP) plays an important part in the areas of Operations Research related with physical distribution and logistics. The VRP is associated with real-world applications concerned with the distribution of supplies between depots and demand points or stations, using vehicles as transport for those same supplies. Among the variations of the VRP is the Capacitated VRP (CVRP), which will be the object of study for the remainder of this paper.

A. Definition

Consider N demand points or stations P_i, with $i = 1, 2, ..., N$, in a given region, each requesting a quantity/ demand σ_i of supplies to be delivered to it. The supplies are kept at a depot D, which also keeps a fleet of V transporting vehicles. Vehicles have equal maximum carrying capacities C. Each vehicle must start and finish its route at the depot D. The problem is to get a group of delivery routes from the depot D to the N stations, so that the total distance covered by the whole fleet of vehicles is minimized. One assumes the demands σ_i are less than the maximum capacity of each vehicle, and the whole quantity σ_i at a station P_i must be satisfied by a single vehicle.

Notice that the VRP reduces to a Traveling Salesman Problem (TSP) when the vehicle capacity constraint is relaxed: if the objective is to minimize total distance, it is a 1-TSP and a n-TSP if a given number of vehicles V is specified. Following this reasoning, the TSP can be looked at as a special case of the VRP.

B. Constructive Heuristics for the CVRP

The Clarke Wright Savings (CWS) algorithm, as presented in Section 5.2.1 of [1], is a widely known approach to tackle CVRP. The key idea behind the CWS algorithm is the concept of 'savings'. In order to better understand this concept — and keeping the definition of the CVRP in mind — let us consider a depot D and N stations P_i, with $i = 1, 2, ..., N$. Now consider an initial solution to the CVRP[1] consisting of N routes (serviced by N vehicles), dispatching one vehicle to each one of the N stations. For each one of the routes to a station P_i, with $i = 1, 2, ..., N$, the length is $2 \times d(D, P_i)$[2], i.e $d(D, P_i) + d(P_i, D)$. For all the routes, the overall cost (i.e. length) becomes $2 \sum_{i=1}^{N} d(D, P_i)$.

If one now dispatches a single vehicle to satisfy two stations P_i and P_j, on a single route, the total distance traveled is reduced by the amount s_{ij}, i.e. the 'saving' one gets by servicing two stations at once vs. servicing only one station:

$$\begin{aligned} s_{ij} &= 2d(D, P_i) + 2d(D, P_j) - (d(D, P_i) + d(P_i, P_j) + d(P_j, D)) \\ &= d(D, P_i) + d(P_j, D) - d(P_i, P_j) \end{aligned}$$

The larger the 'saving', the better it is to combine the link (P_i, P_j) in a single CVRP route. The Clarke and Wright Savings algorithm follows this concept. The algorithm is composed by the following steps, as described in [1], section 5.2.1:

- **STEP 1:** Compute the savings s_{ij} for each combination of stations P_i, P_j, with $i, j = 1, ..., N \wedge i \neq j$. Create N vehicle routes (D, P_i, D) for $i = 1, ..., N$. Order the savings s_{ij} in a descending order of magnitude.
- **STEP 2 (Parallel Version):** Start from the top of the savings list. Given a saving s_{ij}, determine whether there exist two routes, one containing link (D, P_j) and the other containing link (P_i, D), that can feasibly be merged. If so, combine these two routes by deleting (D, P_j) and (P_i, D) and introducing (P_i, P_j).
- **STEP 2 (Sequential Version):** Consider in turn each route $(D, P_i, ..., P_j, D)$. Determine the first saving s_{ki} or s_{jl} that can feasibly be used to merge the current route with another route containing link (P_k, D) or containing link (D, P_l). Implement the merge and repeat this operation to the current route. If no feasible merge exists,

[1] In this context, a solution S to the CVRP is represented as a group of V routes, each one serviced by one of the V available vehicles, which pass through all demand points P when taken together. More details on Section III-A1.

[2] We use the total distance between stations, of a route or of a complete solution S, $d(.)$ as the evaluation function for the CVRP. Check Section III-A2 for details.

consider the next route and reapply the same operations. Stop when no route merge is feasible.

By analyzing the description of the CWS algorithm given above, one can classify it as a clearly constructive heuristic, due to its principle of building 'complex' routes by merging more simple routes — initially in the form (D, P_i, D) — via links (P_i, P_j), associated with a special quantity designated by 'savings'.

C. Metaheuristics for the CVRP

Metaheuristics are high-level strategies designed to explore the solution space in search of near-optimal solutions, which normally combine different lower-level heuristics such as the CWS algorithm described in Section I-B. In contrast to traditional heuristics, metaheuristics often accept solutions which are worse than a current solution S_c during a run, allowing for a more extensive search and the escape from local optima.

In this paper we present a custom implementation of two metaheuristics, adapted for the CVRP: Simulated Annealing and Genetic Algorithms.

1) Simulated Annealing: Simulated Annealing (SA) is an example of a probabilistic metaheuristic, which has been applied to the CVRP by several authors [1]. SA is based on the following rule: given a neighbor S_n of a current solution S_c, if better that S_c, accept it as the new S_c; otherwise, either probabilistically accept this new weaker S_n or continue to search in the current neighborhood. The probability of accepting weaker S_n depends on a parameter T — a temperature — which decreases, i.e. 'cools down', as the algorithm advances, making such acceptances less likely to occur.

The version of the Simulated Annealing (SA) algorithm implemented in this work does not significantly differ from its usual form, as presented in Figure 5.3 in [2][3] (values of L and M are defined a priori):

$k \leftarrow 0$
$l \leftarrow 0$
initialize T
generate a solution S_c via the CWS algorithm
$S_b \leftarrow S_c$
$S_{\text{unchanged}} \leftarrow 0$
repeat
  repeat
    generate new S_n, in the neighborhood of S_c
    if $d(S_n) < d(S_c)$ **then**
      $S_c \leftarrow S_n$
      $S_{\text{unchanged}} \leftarrow 0$
      if $d(S_n) < d(S_b)$ **then**
        $S_b \leftarrow S_n$
      end if
    else if random[0,1) $< e^{\frac{d(S_c)-d(S_n)}{T}}$ **then**
      $S_c \leftarrow S_n$
      $S_{\text{unchanged}} \leftarrow 0$
    else
      $S_{\text{unchanged}} \leftarrow S_{\text{unchanged}} + 1$
    end if
    $l \leftarrow l + 1$
  until $l < L$
  $T \leftarrow T \times (1 - \alpha)$
  $k \leftarrow k + 1$
until $S_{\text{unchanged}} < M$

2) Genetic Algorithms: Genetic Algorithms (GA) try to mimic the process of evolution verified in nature by modeling the natural processes of competition, reproduction and mutation, over a population of individuals W. In this case, the population W, at a given time (or generation) t is composed by $|W|$[4] different solutions S_n to the CVRP. This is one of the main differences of GAs when compared to other metaheuristics, the maintenance of $|W|$ solutions at a given iteration of the procedure, instead of just one as in SA.

The evolutionary approach implemented here follows a typical sequence of steps, as presented in Figure 6.5 in [2]:

$t \leftarrow 0$
initialize $W(t)$
$S_b \leftarrow$ best of $W(t)$
while $t < G$ **do**
  $t \leftarrow t + 1$
  select $W'(t)$ from $W(t-1)$
  generate offspring set $O(t)$ from $W'(t)$
  apply mutations over $O(t)$,
    with probability $M \in [0.0, 1.0]$
  join $W'(t)$ and $O'(t)$ into a new population $W(t)$
  if best of $W(t) < d(S_b)$ **then**
    $S_b \leftarrow$ best of $W(t)$
    save t
  end if
end while

In the listing shown above, t is a generation index; $W(t)$ represents the population of solutions (of size $|W|$, given as a parameter to the algorithm) at generation t; S_b corresponds to the best solution found by the algorithm as it executes; G represents the number of generations to be created (also the stopping condition of the algorithm), defined a priori; $W'(t)$ represents the set of solutions after the application of a Random Tournament Selection strategy (explained in detail in Section III-K); $O(t)$ is the set of 'offspring' solutions generated by the elements of $W'(t)$, using a variant of Partially-Mapped Crossover (PMX) (see Section III-K); $O'(t)$ is resulting set of solutions after applying random mutations over the set $O(t)$, with a probability $M \in [0.0, 1.0]$, also defined a priori (see Section III-M).

II. Objectives

The main objectives of this work are the following:

[3]Note that the SA structure shown in Figure 5.3 in [2] assumes a maximization problem while we assume minimization.

[4]We will henceforth use the notation $|W|$ to represent the size of a population of solutions W.

1) To understand the Capacitated Vehicle Routing Problem (CVRP) and identify appropriate constructive heuristics to tackle the problem.

2) Implement the parallel and sequential versions of the Clarke and Wright Savings (CWS) algorithm, one of the most widely known heuristics (in fact a constructive heuristic) for tackling the CVRP.

3) Plan and implement a local search routine to enhance the sub-optimal output of the CWS constructive heuristic.

4) Plan and implement two metaheuristics for the CVRP: (1) a Simulated Annealing (SA) approach and (2) a Genetic Algorithm.

5) Run the proposed implementations against several datasets, documenting the obtained results. For each dataset, compare and discuss the results of (1) parallel version of the CWS, (2) sequential version of the CWS, (3) eventual enhancements provided by the implemented local search routine, (4) behavior of the metaheuristics according to different parameters. These different results should be compared against the optimal solution, provided in the list of datasets to be applied [3].

III. Methodology

The following subsections provide insights on the methodology one followed in order to achieve the objectives listed in Section II.

A. Solution Representation and Evaluation Function for the CVRP

All strategies and algorithms proposed in Section I to tackle the CVRP were implemented in C/C++, following descriptions distributed over several sources [1], [2], [4]–[6]. The following sub-sections provide some details about each implementation.

1) Solution Representation: Regarding general data structures for representing important entities and quantities used in the CVRP, the following are of particular importance:

- `Station`: Represents a demand point P which should be visited by a vehicle. An instance of `Station` has information about (1) the demand point's index (as given in the dataset), (2) demand point's (x, y) coordinates, (3) the demand value of the point, (4) the route it (currently) belongs to and (5) pointers to the previous and next demand points in the route it belongs to.
- `Route`: Represents a single vehicle's route. Contains information about (1) an identifying index for the route, (2) the current cost (i.e. length) of the route, (3) the current amount of goods supplied by the route, (4) the number of demand points currently in the route and (5) pointers to the first and last `Station` in the route.

The entities listed above are important for the solution representations used in this problem. Since solutions in the CVRP are usually represented as sequences of demand points closed by the depot (i.e. routes R), in this case we treat a solution S as set of linked lists of `Station` instances:

$$R_1 : D \rightarrow P_1 \rightarrow P_{23} \rightarrow P_7 \rightarrow ... \rightarrow D$$

$$R_2 : D \rightarrow P_3 \rightarrow P_{12} \rightarrow P_9 \rightarrow ... \rightarrow D$$

$$...$$

A graphical representation of a solution S for a CVRP instance with 38 stations is given in Figure 1

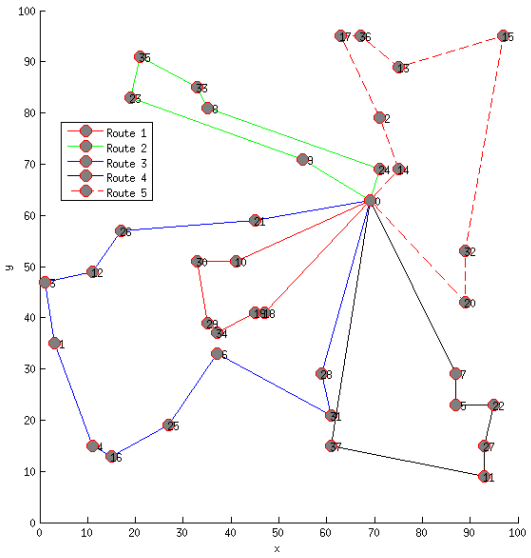


Fig. 1: Graphical depiction of a solution S for a CVRP instance with 38 stations.

Each linked list is described by an associated `Route` instance. A special aggregating data structure `CVRP` is used to represent a graph for the CVRP, i.e. in practice a `CVRP` instance is our solution representation S. This entity holds a list of `Route` instances, each one containing several `Station` instances, linked between each other according to their order within a route.

2) Evaluation Function: The CVRP aims at minimizing the total distance traveled by the different vehicles when following their respective routes. Therefore, we use the total distance between `Station` instances within a `Route`, for all the `Route` instances in a solution S, as the evaluation function $d(S)$. Given a `Route` R, we sometimes use $d(R)$ for referring to the distance traveled by a vehicle within that single route. Also, as previously used in Section I-B, we may use $d(D, P_i)$ to represent the distance between two stations (in this case the depot D and a generic station P_i).

B. Clarke and Wright Savings (CWS) Algorithm

After the initial literature review phase, documented in Section I, the Clarke and Wright Savings (CWS) algorithm, as presented in Section 5.2.1 in [1], was identified as an appropriate heuristic for tackling CVRP. It presents itself as a clearly constructive heuristic, due to its principle of building

'complex' routes by merging more simple routes — initially in the form (D, P_i, D) — via links (P_i, P_j), associated with a special quantity designated by 'savings'.

The CWS presents two versions, (1) a parallel version and (2) a sequential version, further explained in Section III-D.

C. Construction of the Savings List

One of the key features of the CWS algorithm is the use of the 'saving' entities, already discussed in previous sections. In addition to general data structures for representing a CVRP problem (described in Section III-A), the CWS algorithm implementation defines the representation of a saving quantity, `Saving`, which contains information about (1) the indexes of demand points (P_i, P_j) (i.e. instances of `Station`) to which the saving is associated and (2) the actual savings value.

The description provided in [1] suggests the calculation of every s_{ij} for all P_i, P_j with $i, j = 1, ..., N$, only imposing $i \neq j$ as a constraint. For N nodes, this would result in a list of size $N(N-1)$. This number can be further reduced by realizing the following:

- The savings associated with the depot can be discarded, which reduces the size by a parcel of $2(N-1)$.
- $s_{ij} = s_{ji}$, which reduces the size of the list to half. This translates into a reduction of memory usage, however, requires an extra computation on the `merge` routines, as mentioned in Section III-D.

Taking the considerations above into account, we get a final size for the savings list of $\frac{1}{2}((N-1)(N-2))$. This is easy to imagine in matrix form, as seen in 1. Imagine a $N \times N$ matrix, with the rows and columns as the node indexes i and j, so that each cell is a (i, j) combination. The elements marked by an **x** are those which are eliminated by the simplifications mentioned above, while those marked with **o** are kept. This is important in computational terms, since both parallel and sequential versions of the CWS algorithm cycle through the entire savings list, with the possibility of repetitions for the sequential case.

$$\begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{x} & \mathbf{o} & \mathbf{o} \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{o} \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \end{bmatrix} \tag{1}$$

In addition, Step 1 of the CWS algorithm indicates that the savings list should be ordered in descending order of magnitude. To do so, we use a custom implementation of the Quicksort algorithm [7].

D. Versions of the CWS

The simplifications performed while constructing the savings list have implications on the `merge()` routines, for both versions of the CWS algorithm, specifically the evaluation of the conditions shown below:

1: **if** $(P_{i+1}$ is $D) \wedge (P_{j-1}$ is $D)$ **then**
2: Join route R_k ($R_k \ni P_i$) and R_l ($R_l \ni P_j$), via link (P_i, P_j)
3: **else if** $(P_{i-1}$ is $D) \wedge (P_{j+1}$ is $D)$ **then**
4: Join route R_k ($R_k \ni P_i$) and R_l ($R_l \ni P_j$), via link (P_j, P_i)
5: **else**
6: At least one of (P_i, P_j) is interior to R_k or R_l. Abort.
7: **end if**

In short, the conditions shown above compensate the elimination of the symmetric pairs of points (P_i, P_j) and (P_j, P_i), by evaluating a merge between two routes via the link (P_j, P_i) if trying it via the link (P_i, P_j) does not work.

In addition to the conditions given above, the procedures taken upon the establishment of a link between two previously unconnected routes is similar among the two versions. Specifically, considering route R_l is merged to route R_k (i.e. route R_l ceases to exist) all `Station` entities previously part of a route R_l are updated to belong to route R_k, the pointers of the linking `Station` entities are updated (as these no longer point to the depot) and the information held by the `Route` entity of route R_l is appropriately merged with that of route R_k.

The parallel and sequential version of the CWS algorithm mostly differ in the way the various route merging operations are applied, not on the nature of the single `merge()` operation. For the parallel version, the algorithm cycles through the savings list once, performing merges between several routes along the way, i.e. it does not 'stick' to merging other routes to a particular route, given a pair of points (P_i, P_j) it attempts to merge any two routes that include the edges P_i and P_j.

For the sequential version, the algorithm 'sticks' to a route R_k, cycles through the savings list and evaluates if a link (P_i, P_j) is suitable for merging a route R_l with R_k (i.e. only links (P_i, P_j) in which either P_i or P_j are in the edges of R_k are considered suitable). If that is the case, that link (P_i, P_j) is discarded from the list, and the algorithm returns to the top of the savings list, since the new merging operation may have enabled a previous pair (P_w, P_z). This is repeated until the algorithm reaches the end of the savings list. In its turn, the whole procedure is repeated for each remaining route in the algorithm's route list[5].

E. Constraints in CWS Route Mergings

Whenever a merge operation between two routes R_i and R_j is attempted — in both the parallel and sequential cases — the implemented version of the CWS algorithm takes the following evaluations into account (let $c(R)$ represent the quantity of goods delivered within a route R):

- Total amount of goods supplied by the two routes must be less than a vehicle's capacity C, i.e. $c(R_i) + c(R_j) \leq C$. If such a condition is not verified, the merge operations is not allowed.

[5]In our case, the process is only applied to a new route in the list if the associated `Route` entity is active.

- Demand points on a link (P_i, P_j), associated with a saving quantity s_{ij} under consideration for a route merge, must not be interior to a route.
- Demand points on a link (P_i, P_j), associated with a saving quantity s_{ij} under consideration for a route merge, must not belong to the same route.

If the previously listed conditions are met, the merge operation is allowed to go ahead, the `Station` (P_i, P_j) instances, at the appropriate edges of each route R_i and R_j, are linked in order to form a new route. The value of the new route, $d(R_n)$ is calculated according to $d(R_n) = d(R_i) + d(R_j) - s_{ij}$.

Although no constraints related to the number of routes V are imposed in the `merge()` functions, the CWS algorithm always determines the optimal number of routes[6] for the sequential version, while it sometimes reports a sub-optimal number for the parallel version, as verified in the results given in Section IV.

F. 2-Opt Local Search Heuristic

Given the results of the constructive heuristic obtained after a run of the CWS algorithm, we run a 2-opt local search heuristic over the results of each route R_i. This procedure is based on that described in [4], [5]. Its objective is to remove crossings of links in a route, while preserving the orientation of the routes, which reduces the distance of the route (see Section 6.2 in [2]).

In this case, we try to find a better solution R_i', i.e. one for which the evaluation function $d(R_i') < d(R_i)$, within the neighborhood of R_i, i.e. the routes one can reach by swapping two nodes in R_i.

The 2-opt local search routine was implemented according to the specification given below, which shows an exchange between all combination of nodes at positions k and m, within all routes calculated by the CWS algorithm, R (let $R[k]$ and $R[m]$ represent the `Station` instances at indexes k and m within a `Route` R, and $d(R[k], R[m])$ represent the euclidean distance between stations $R[k]$ and $R[m]$):

```
1:  for all final CWS routes R do
2:      best_distance ← d(R_i)
3:      for (k ← 1; k < size(R_i) −1; k++) do
4:          for (m ← (m + 1); m < size(R_i); m++) do
5:              new_distance ←
                    d(R_i)
                    + d(R_i[k − 1], R_i[m])
                    + d(R_i[k], R_i[m + 1])
                    - d(R_i[k − 1], R_i[k])
                    - d(R_i[m], R_i[m + 1])
6:              if new_distance < best_distance then
7:                  best_distance ← new_distance
8:                  swap(R_i[k], R_i[m])
9:              end if
10:         end for
11:     end for
12: end for
```

[6]By 'optimal' one refers to the best value given by each test instance.

After an initial evaluation of the potential gains of an exchange between nodes k and m, if a lower cost $d(R_i') < d(R_i)$ is detected, the actual `swap()` procedure is applied. Given a route arrangement, such as the one shown in Figure 2(a), the `swap(W,Y)` procedure changes it into Figure 2(b).

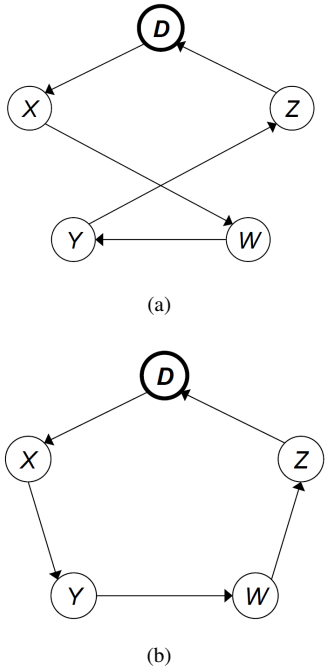


Fig. 2: Graphical depiction of the `swap()` procedure, in this case between nodes W and Y of a given route composed by four nodes (D represents a depot).

G. Simulated Annealing (SA)

The version of the Simulated Annealing (SA) algorithm implemented here follows the specification given in Section I-C. Its main 'deviations' from other versions of the algorithm are (1) the mechanism for neighborhood generation (details in Section III-H) and (2) the method for setting an initial temperature T_s (see Section III-I).

The SA algorithm was implemented in a separate C/C++ module which uses the basic CVRP data structures given in Section III-A and uses the CWS module to generate initial solutions. As a result of this implementation, a new heuristic — 1-interchange local search (see Section III-H — was added to the basic CVRP module.

H. 1-Interchange Local Search Heuristic

The neighborhood generation mechanism used in line 9 of the SA algorithm described in Section I-C is based on a λ-interchange method introduced by I. Osman [1], [4], [6]. Regarding the work presented here, the adoption of such heuristic represents an evolution compared to the simpler 2-opt local search heuristic described in Section III-F: now, instead of restricting the interchanges between two stations P_i and P_j to a single route R, a neighbor S_n of a solution S is generated by inserting/exchanging one station into/between

different routes R_i and R_j. Such 1-interchange mechanism includes two different operations:

- **(0,1) or (1,0):** Extracting a station P_i from a route R_i and inserting it in a route R_j.
- **(1,1):** Swapping stations P_i and P_j between their initial routes R_i and R_j.

In order to remove crossings of links in the routes R_i and R_j — which may happen due to the re-configuration caused by the 1-interchange procedure — the same 2-opt local search method described in Section III-F is applied to both routes at the end of the 'core' 1-interchange process.

I. Initial Temperature Calculation

The results of SA algorithms depend on the tuning of multiple parameters: (1) initial temperature T_s, (2) cooling ratio α, (3) termination condition and (4) halting criterion. We follow a method for determining T_s which is similar to that used in the hybrid Tabu Search - Simulated Annealing algorithm proposed by I. Osman in [6].

We define the acceptance probability A of a new solution S_n (derived from a current solution S_c via a 1-interchange procedure) as (see line 16 of the algorithm defined in Section I-C):

$$A(S_n) = \begin{cases} e^{\frac{-(d(S_n)-d(S_c))}{T_c}} & \text{if } d(S_n) \geq d(S_c) \\ 1 & \text{if } d(S_n) < d(S_c) \end{cases}$$

where $d(.)$ is the evaluation function and T_c is the current temperature. Considering an initial solution S_i, we determine the initial temperature T_s by calculating T for which $A(\min(\Delta_d)) = 0.5$, where $\Delta_d = d(S_n) - d(S_i)$.

Given an initial solution S_i, its entire neighborhood obtained via a 1-interchange local search heuristic is analyzed and the changes in the evaluation function Δ_d are recorded for each feasible interchange[7]. We then apply the following expression to determine T_s:

$$T_s = \frac{\min(\Delta_d)}{\ln(0.5)}$$

J. Genetic Algorithm (GA)

The version of the Genetic Algorithm (GA) algorithm implemented in this work follows the specifications given in Section I-C2. Its main specificities are (1) the mechanism for population selection (see Section III-K); (2) the crossover method for generating new solutions (see Section III-L); (3) the mutation process applied to new solutions generated by the crossover process (see Section III-M).

As with previous cases, the GA algorithm was implemented in a separate C/C++ module which uses the basic CVRP data structures described in Section III-A. The following input parameters must be given to our GA implementation:

- Solutions provided by the CWS (both parallel and sequential) and SA algorithms.
- Number of generations to produce G.
- Population size $|W|$.
- Mutation probability for the crossover process $M \in [0.0, 1.0]$

In our case, we include the solutions provided by the parallel and sequential versions of the CWS algorithm, along with that provided via SA, in the initial population $W(t_0)$. The remaining $|W| - 3$ elements of the initial population $W(t_0)$ are randomly generated, taking into account the capacity C and route number V constraints implicit in the CWS and SA solutions.

K. Random Tournament Selection

The population selection strategy used in our GA instance (step 6 on the listing shown in Section I-C2) is based on simple Random Tournament Selection, with a sample size of two, i.e. one pair of solutions (S_i, S_j) is picked at random from $W(t-1)$ and one chooses that S with lowest evaluation function value $d(S)$. In addition, our strategy automatically elects the best 10% of $W(t-1)$ to $W'(t)$, the set of 'fittest' solutions (those with lower values of $d(.)$), which are excused from participating in the tournament.

The process is summarized in pseudo-code in the following listing ($|W|$ is the population size, given as argument to the algorithm):

```
1: sort W(t-1)
2: add top 10% of W(t-1) to W'(t),
       exclude them from W(t-1)
3: while size of W'(t) < |W|/2 do
4:     repeat
5:         select random pair (S1, S2) from W(t-1)
6:     until S1! = S2
7:     if d(S1) ≤ d(S2) then
8:         add S1 to W'(t), exclude it from W(t-1)
9:     else if d(S1) > d(S2) then
10:        add S2 to W'(t), exclude it from W(t-1)
11:    end if
12: end while
```

L. Crossover Approach

Partial-Mapped Crossover (PMX) and Order Crossover (OX) [1] are examples of crossover operators, specially devised for the Traveling Salesman Problem (TSP). Besides the special characteristics imposed by the TSP to crossover operations, the CVRP adds additional constraints, specifically the division of a typical TSP tour in several routes R and capacity of routes C. As proposed by [1] in Section 6.5.3, we tackle the multiple route constraint by extending the CVRP into a TSP-like representation, with multiple copies of the depot in-between routes. E.g. for a CVRP with 8 stations and 3 routes (with D representing the depot) such as

$$D \rightarrow 1 \rightarrow 5 \rightarrow 8 \rightarrow D$$

[7] I.e. all changes that result in routes not exceeding the vehicle capacity constraint.

$$D \rightarrow 3 \rightarrow 2 \rightarrow D$$

$$D \rightarrow 7 \rightarrow 6 \rightarrow 4 \rightarrow D$$

one would get the following TSP-like representation:

$$D \rightarrow 1 \rightarrow 5 \rightarrow 8 \rightarrow D \rightarrow 3 \rightarrow 2 \rightarrow D \rightarrow 7 \rightarrow 6 \rightarrow 4 \rightarrow D$$

In addition, we apply a PMX-based crossover operation similar to the Best Route Better Adjustment (BRBAX) recombination mechanism presented in [8]. The following listing summarizes the offspring generation procedure (which includes the BRBAX-based crossover operation) in pseudo-code (let Q_R be the quantity of goods supplied in a route R; V the number of routes of the solutions contained in the set $W'(t)$; the remaining symbols follow the conventions applied in the paper so far):

```
1:  while size of O(t) < |W|/2 do
2:      repeat
3:          select random pair (S1, S2) from W'(t)
4:      until S1 != S2
5:      initialize 'offspring' solution So
6:      sort routes of S1 according to C - QR,
            in ascending order
7:      add best V/2 routes of S1 to So
8:      for all routes R in S2 do
9:          for all stations P in route Ri do
10:             if Pi not in So then
11:                 add Pi to So, in order,
                        with D delimiting routes of S2
12:             end if
13:         end for
14:     end for
15:     if number of routes in So > V then
16:         condense So into V routes
17:     end if
18:     if CVRP constraints are met by So then
19:         add So to O(t)
20:     else
21:         discard So
22:     end if
23: end while
```

While the route condensation method is not described here, it is partially commented in the method `FromTSPRepresentation()`, in file `Evolutionary.cpp`.

M. Mutations

The 'offspring' solutions of the set $O(t)$, generated via the crossover process described in Section III-K, can be slightly altered by a mutation procedure which occurs with probability $M \in [0.0, 1.0]$, defined as an input parameter.

The mutation process is simple, consisting in the application of two simple heuristics, already applied in the previous CWS and SA implementations:

- A 1-interchange procedure (see Section III-H) between two random routes of a solution $S_o \in O(t)$.
- A 2-opt local search method (see Section III-F) on a random route of a solution $S_o \in O(t)$.

IV. Results

The following subsections present computational results obtained from the implementations of the CWS algorithm, 2-opt local search and SA procedures described over Sections I and III.

The tests were performed on a 64-bit machine equipped with a Intel(R) Core(TM) i5-3320M CPU (2 cores), with a clock rate of 2.60 GHz, 5.7 GB of RAM.

A. CWS and 2-Opt Local Search Combination

Tables I to III provide a list of the results obtained by the proposed implementation(s), against several datasets available in [3]. For each dataset, we provide results for (1) the parallel version of the CWS, (2) the implemented sequential version of the CWS and (3) the enhancements (if any) provided by the implemented local search routine. Table I provides a comparison of the total cost of the different N routes, while Table II compares the computation time along the same categories as those present in Table I. Table III compares the number of generated routes (for both parallel and sequential versions of the CWS algorithm) with the optimal values provided in the datasets [3]

Dataset	CWS	2-Opt	Optimal
A-n32-k5	865[1] 987[2]	863[1] 968[2]	784
A-n34-k5	826 990	809 955	778
A-n38-k5	816 939	785 908	730
A-n39-k5	925 1033	919 1003	822
A-n54-k7	1247 1402	1230 1372	1167
A-n60-k9	1429 1871	1422 1765	1354
F-n135-k7	1269 1658	1252 1531	1162
M-n151-k12	1209 1432	1198 1404	1053
M-n200-k17	1504 1865	1500 1797	1373

[1] Parallel version of the CWS algorithm.
[2] Sequential version of the CWS algorithm.

TABLE I: Results obtained by the proposed implementation(s) (in terms of total route costs), against several datasets available in [3].

The results given above confirm the dominance of the parallel version over the sequential one, reported in Section

Dataset	CWS	2-Opt
A-n32-k5	0.098[1] 0.120[2]	0.007[1] 0.011[2]
A-n34-k5	0.233 0.441	0.022 0.022
A-n38-k5	0.321 0.592	0.024 0.026
A-n39-k5	0.412 0.692	0.028 0.028
A-n54-k7	0.647 1.583	0.019 0.038
A-n60-k9	0.818 2.115	0.033 0.038
F-n135-k7	10.805 16.313	0.082 0.091
M-n151-k12	15.779 23.087	0.066 0.062
M-n151-k12	26.462 43.850	0.081 0.082

[1] Parallel version of the CWS algorithm.
[2] Sequential version of the CWS algorithm.

TABLE II: Results obtained by the proposed implementation(s) (in terms of average computation time, in milliseconds), against several datasets available in [3].

5.2.1 of [1]. The computation times seem to also follow this tendency, which is at least intuitive given the nature of the two versions: the parallel version merges multiple routes for one passage over the savings list, while the sequential version 'sticks' to a route and evaluates each entry in the savings list against that route only.

On the other hand, the sequential strategy yields better results concerning the number of generated routes, as verified in Table III, specifically for datasets `A-n38-k5` and `F-n135-k7`. Since the parallel version builds multiple routes in parallel, when attempting merges along entries near the end of the savings list, it may reject a large number of merging operations due to the violation of capacity constraints (or due to inclusion of the 'savings' nodes in interior positions to routes), resulting in additional 'unmergeable' routes.

The 2-opt local search procedure produced enhancements on the values (i.e. total cost of N routes) of solutions initially provided by the CWS constructive heuristic, for all datasets and for both versions of the CWS algorithm. In addition, the simple 2-opt local search procedure generally produces better enhancements (i.e. difference between (1) the value of solution provided by the constructive heuristic and (2) the value of the solution provided by the local search procedure) when applied over the sequential version solutions.

The closest values to the optimal solutions are obtained for datasets `A-n34-k5` and `A-n38-k5`, after a 2-opt local search on the solution provided by the parallel version of the CWS algorithm (solution value differences of 31 and 55 units, respectively). Listings 1 and 2 provide the differences between

Dataset	# Routes (P)	# Routes (S)	# Routes (O)
A-n32-k5	5	5	5
A-n34-k5	5	5	5
A-n38-k5	6	5	5
A-n39-k5	5	5	5
A-n54-k7	7	7	7
A-n60-k9	9	9	9
F-n135-k7	8	7	7
M-n151-k12	12	12	12
M-n200-k17	17	17	17

TABLE III: Results obtained by the proposed implementation(s) (in terms of number of generated routes), against several datasets available in [3].

the optimal routes and those found by the proposed CWS constructive heuristic & 2-opt local search combination, for the dataset `A-n38-k5`.

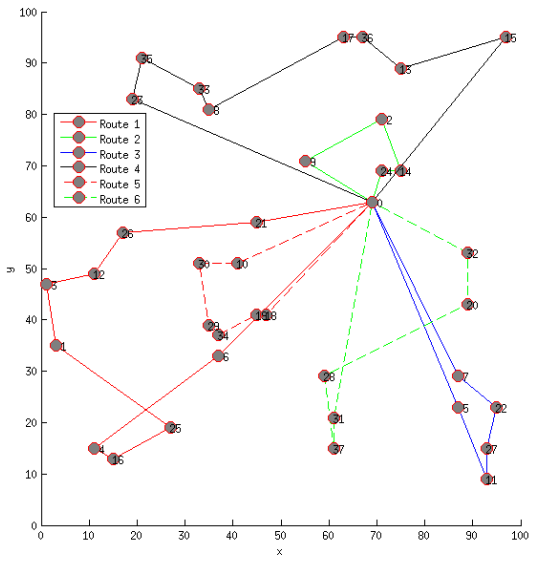
One can identify similarities between some of the routes, particularly between Route 4 in Listing 1 and Route 1 in Listing 2 or between Route 1 and Route 3. This may indicate that, in addition to a local search routine that swaps elements within individual routes, one may benefit from attempting swaps between different routes, such as the λ-interchange mechanism suggested in [6], and which is applied in the SA proposed in Section I-C. The graphical representations of the same routes, given in Figure 3 show how the 2-opt local search procedure removed crossings between links, specifically Routes 1 and 6, along with an enhancement of Route 4, achieved via (1) the swap between nodes 23 and 8 (arcs $(0, 23) \rightarrow (0, 8)$ and $(8, 17) \rightarrow (23, 17)$) and (2) a swap between nodes 33 and 23 (arcs $(8, 33) \rightarrow (8, 23)$ and $(23, 17) \rightarrow (33, 17)$).

Listing 1: Optimal routes and solution value for the A-n38-k5 dataset. Values provided by Augerat et al. [3]

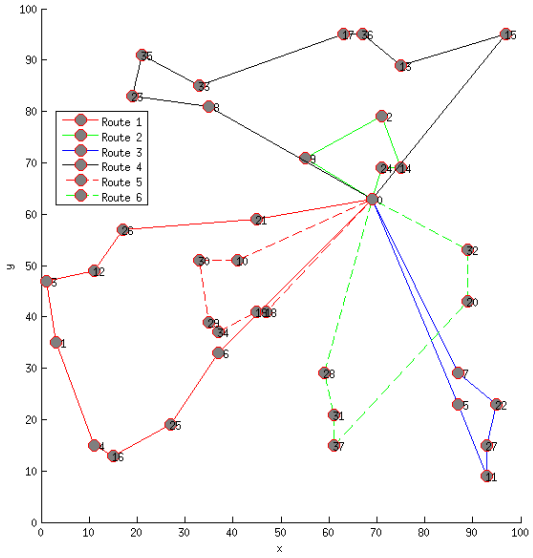
```
Route #1: 37 11 27 22 5 7
Route #2: 10 30 29 34 19 18
Route #3: 20 32 15 13 36 17 2 14
Route #4: 28 31 6 25 16 4 1 3 12 26 21
Route #5: 24 33 35 23 8 9
cost 730
```

Listing 2: Routes and solution value obtained after running the proposed CWS constructive heuristic & 2-opt local search combination, against the dataset A-n38-k5.

```
Route #1: 6 25 16 4 1 3 12 26 21
Route #2: 24 14 2 9
Route #3: 5 11 27 22 7
Route #4: 8 23 35 33 17 36 13 15
Route #5: 10 30 29 34 19 18
Route #6: 28 31 37 20 32
cost 785
```



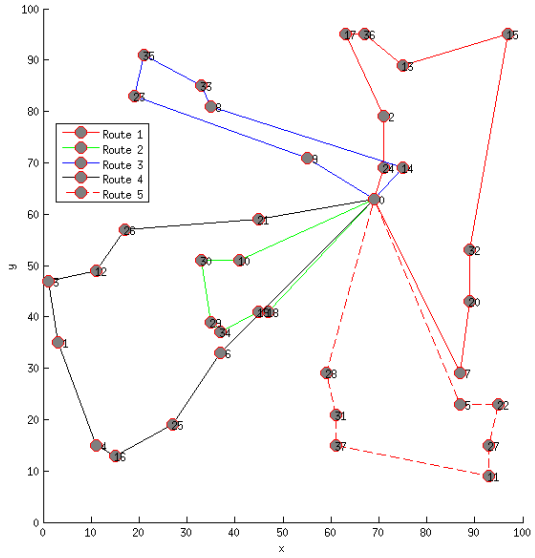
(a)



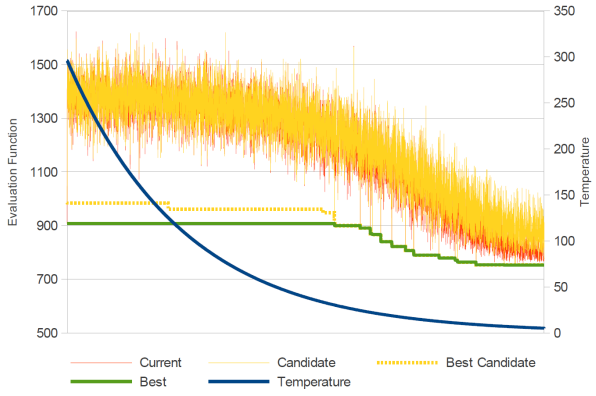
(b)

Fig. 3: Graphical depiction of the routes generated by the proposed solution (dataset `A-n38-k5`), (a) after a CWS run (parallel version) and (b) after the 2-opt local search procedure.

- $T_s = 295.75$
- $\alpha = 0.005$
- $L = 35000$
- Stopping condition: 300 iterations without change in S_c.



(a)



(b)

Fig. 4: Graphical depiction of the routes generated by the proposed SA implementation (dataset `A-n38-k5`), starting with the solution from the sequential version of the CWS algorithm (a); depiction of the evolution of the SA procedure, with $T_s = 295.75$, $\alpha = 0.005$, $L = 35000$, for a total number of iterations of 27.62 M (b).

B. Simulated Annealing

Following the focus on the example given by dataset `A-n38-k5`, Figure 4 shows the graphical results of a SA run, starting with the solution given by the sequential version of the CWS algorithm. Besides the improvement versus both the sequential and parallel versions of the CWS algorithm ($e(.) = 753$ vs. [785, 908]), the joint inspection of Listings 1 and 3 shows how closer the SA solution is to the optimal. These results were obtained with the following set of parameters for the SA algorithm:

Listing 3: Routes and solution value obtained after running the proposed SA implementation metaheuristic, against the dataset A-n38-k5.

```
Route #1: 24 2 17 36 13 15 32 20 7
Route #2: 10 30 29 34 19 18
Route #3: 9 23 35 33 8 14
Route #4: 6 25 16 4 1 3 12 26 21
Route #5: 5 22 27 11 37 31 28
cost 753
```

Tables IV and V provide a list of the results obtained by the proposed SA implementation, against several of the same

datasets used in Section IV-A, with $L = 35000$ (with some noted exceptions) and the stop criterion being the occurrence of more than 300 iteration without change in S_c. The results confirm that SA techniques are costly in terms of time, while achieving improvements for all the datasets, for both versions of the CWS algorithm and the effect of changing some of the algorithm parameters (the initial temperature T_s is automatically calculated using the method shown in III-I, however an additional test with $10 \times T_s$ was performed for each of the datasets chosen for evaluation).

Dataset	T_s	α	S_i	S_b
A-n38-k5	295.75	0.005	908[1]	753
		0.05		753
	2957.50	0.005	908[1]	753
		0.05		753
M-n151-k12	158.70	0.005	1404[1]	1142
		0.05		1142
	1586.96	0.005	1404[1]	1134
		0.05		1128
	4530.06	0.005	1198[2]	1135
M-n200-k17	389.53	0.005	1797[1]	1485
		0.05		1442
	3895.28	0.005	1797[1]	1461
		0.005		1433[3]
		0.05		1433
		0.05		1491[3]
	4270.38	0.005	1500[2]	1470

[1] Initial solution value after the sequential version of the CWS algorithm.
[2] Initial solution value after the parallel version of the CWS algorithm.
[3] $L = 50000$.

TABLE IV: Results obtained by the proposed SA implementation, against several datasets available in [3], and according to different values for S_i, T_s and α. $L = 35000$ and stop criterion equal to 300 iterations without change in S_c, otherwise noted.

Table IV shows that for the `A-n38-k5`, with a rather small number of stations and routes, the same result of the evaluation function is obtained for all variations of SA parameters. Such a result is probably caused by the use of the same initial solution already optimized by the CWS algorithm and the rather limited 1-interchange local search heuristic used for neighbor generation (a 2-interchange mechanism would probably generate better results, i.e. better 'escapes' from local optima).

For the `M-n151-k12` and `M-n200-k17` datasets, the best results are obtained for the largest tested $\alpha = 0.05$ (even though one achieves the same best result for the `M-n200-k17` — $e(.) = 1433$ — with $\alpha = 0.005$ and $L = 50000$) which seems somewhat counterintuitive. Due to the stochastic nature of the SA approach, such a result is plausible. Nevertheless, it was also noted that most of the improvements for the larger datasets occurred during the final phases of the algorithm, i.e. in the 'colder', 'hill-climber' phase of the algorithm, which seems to indicate the SA implementation keeps getting stuck in the same local optimum. Although the probabilistic nature of the SA algorithm should avoid such a situation, this is maybe caused by the use of solutions generated by the CWS algorithm (in the case of the `M-n151-k12` and `M-n200-k17` datasets those provided by the parallel version are also used) and the use of a 'poor' 1-interchange neighbor generation mechanism.

Dataset	T_s	α	S_i	T_{exec}	N_{iter}
A-n38-k5	295.75	0.005	908[1]	154.69	27.62 M
		0.05		17.17	3.01 M
	2957.50	0.005	908[1]	226.80	43.75 M
		0.05		28.72	5.22 M
M-n151-k12	158.70	0.005	1404[1]	614.72	33.71 M
		0.05		70.99	3.75 M
	1586.96	0.005	1404[1]	758.51	48.44 M
		0.05		85.52	5.25 M
	4530.06	0.005	1198[2]	878.32	57.33 M
M-n200-k17	389.53	0.005	1797[1]	744.66	38.36 M
		0.05		86.92	4.24 M
	3895.28	0.005	1797[1]	918.86	55.37 M
		0.005		1332.25[3]	79.55 M
		0.05		106.06	5.92 M
		0.05		131.77[3]	7.9 M
	4270.38	0.005	1500[2]	912.66	55.20 M

[1] Initial solution value after the sequential version of the CWS algorithm.
[2] Initial solution value after the parallel version of the CWS algorithm.
[3] $L = 50000$.

TABLE V: Execution times and iteration numbers obtained by running the proposed SA implementation against several datasets available in [3], and according to different values for S_i, T_s and α. $L = 35000$ and stop criterion equal to 300 iterations without change in S_c, otherwise noted.

C. Genetic Algortithm

Table VI summarizes the results obtained with our GA implementation, for datasets `A-n38-k5`, `A-n60-k5` and `M-n200-k17`. As mentioned in Section III-J, the initial populations contain the solutions provided by the CWS algorithm (both parallel and sequential versions) as well as that generated via SA, with the following parameters:

- $T_s = 2957.50$ (`A-n38-k5`), $T_s = 3318.20$ (`A-n60-k5`), $T_s = 3895.28$ (`M-n200-k17`).
- $\alpha = 0.05$
- $L = 25000$
- Stopping condition: 300 iterations without change in S_c.

The number of generations was set to $G = 10000$ for all experiments summarized in Table VI, and several values for the population size N, and mutation probability M were used.

Overall, the results show that our GA implementation does not yield significant benefits when compared with the solutions provided by SA. For most cases, the solution generated by the latter method prevails through the G generations. With the lowest value of M, one also verifies the populations $P(t)$ tend

Dataset	N	M	S_i	S_b	G_i	T_{exec}
A-n38-k5	30	0.25	**752**	752	0	14.29
		0.50	753	753	0	14.12
		0.75	753	753	0	14.30
	60	0.25	**752**	752	0	27.33
		0.50	754	754	0	27.36
		0.75	753	753	0	28.47
	120	0.25	753	**752**	1931	56.77
		0.50	753	753	0	57.11
		0.75	**752**	752	0	57.75
A-n60-k5	30	0.25	**1381**	1381	0	15.88
		0.50	1387	1387	0	15.92
		0.75	1389	1389	0	16.27
	60	0.25	1389	1389	0	23.67
		0.50	1391	1391	0	26.06
		0.75	1389	1389	0	27.86
	120	0.25	1391	1391	0	51.66
		0.50	1388	1385	1175	52.94
		0.75	**1381**	1381	0	60.39
M-n200-k17	30	0.25	1446	1442	3080	40.92
		0.50	1460	1456	2400	42.84
		0.75	1451	1447	3168	43.26
	60	0.25	1439	1439	0	93.38
		0.50	1443	1443	0	98.24
		0.75	1462	1451	5263	102.48
	120	0.25	1452	1452	0	188.37
		0.50	1435	**1429**	6776	177.16
		0.75	1462	1460	9448	236.99

TABLE VI: Results obtained by the proposed GA implementation, against several datasets available in [3], with $G = 10000$ and variable N, M. The initial population includes the solutions provided by the previously implemented CWS algorithm (both parallel and sequential versions) and SA method. Includes execution times, T_{exec}, in seconds.

to homogeneity, with solution S_b becoming the most common in the successive generations t, as $t \rightarrow G$. Nevertheless, for the larger dataset, `M-n200-k17`, our GA implementation was able to improve the initially provided solutions in most of the runs (probably due to 'poor' SA solutions caused by a low α, which provided some room for improvement) and even found an overall best solution for this dataset, with value $e(.) = 1429$, with parameters $N = 120$, $M = 0.50$ at generation $G_i = 6776$.

Listing 4: Routes and solution value obtained after running the proposed GA implementation, against the dataset A-n38-k5.

```
Route #1: 10 30 29 34 19 18
Route #2: 9 23 35 33 8 24
Route #3: 28 31 6 25 16 4 1 3 12 26 21
Route #4: 37 11 27 22 5 7
Route #5: 20 32 15 13 36 17 2 14
cost 752
```

During the GA test runs, we have also found overall best solutions for the `A-n38-k5` dataset, depicted in Figure 5 and Listing 4 (for $N = 120$, $M = 0.25$, $G_i = 1931$, even though the same value had been obtained via SA on previous runs) and

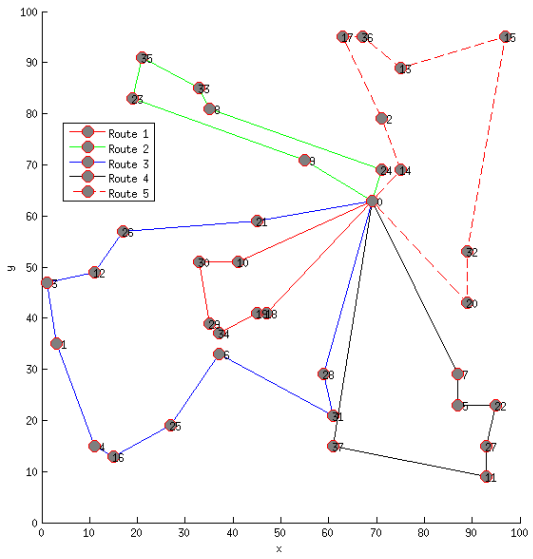


Fig. 5: Graphical depiction of the routes generated by the proposed GA implementation (dataset `A-n38-k5`), with parameters $N = 120$ and $M = 0.25$ at generation $G_i = 1931$.

the `A-n60-k5` dataset, with $e(.) = 1381$ (solution generated via SA).

Following the rather disapointing results shown in Table VI, we have also ran our GA implementation starting with an 'unbiased' initial population, i.e. without the inclusion of the solutions provided by previously implemented methods, only with randomly generated solutions. The results are summarized in Table VII, for $G = 100000$ and $N = 120$, which show that the GA implementation is able to produce improvements to the initial population, although never as competitive as the CWS or SA methods.

Dataset	N	M	S_i	S_b	G_i	T_{exec}
A-n38-k5	120	0.25	1677	788	1301	38.68
		0.50	1692	773	8687	46.51
		0.75	1809	834	96415	51.10
A-n60-k5	120	0.25	3200	1497	22855	51.03
		0.50	3195	1476	22841	60.67
		0.75	3187	1458	3202	63.25
M-n200-k17	120	0.25	6418	1558	88945	194.92
		0.50	6585	1456	77971	206.74
		0.75	6540	1789	49304	228.10

TABLE VII: Results obtained by the proposed GA implementation, against several datasets available in [3], with $G = 100000$, $N = 120$ and an initial population entirely composed by randomly generated solutions. Includes execution times, T_{exec}, in seconds.

V. Conclusions

This work documents the study of different heuristics and metaheuristics, essential pieces of problem solving toolboxes, applied in many areas of science and engineering [9]–[11].

Here we focus on a particular case study – the Capacitated Vehicle Routing Problem (CVRP) – and provide custom implementations of several heuristics and metaheuristics, specially oriented to tackle it. We describe the implementation of both parallel and sequential versions of the Clarke and Wright Savings (CWS) algorithm, several local search heuristics (2-opt and 1-interchange), as well as custom versions of the Simulated Annealing (SA) algorithm and Genetic Algorithm (GA) as examples of metaheuristics. The C/C++ code used in the implementation is available online [8] for further inspection and was tested against several datasets, available in [3].

None of the implementations reached the best values stated in the datasets. The CWS algorithm proved a strong constructive heuristic, particularly when combined with a 2-opt local search procedure, due to the latter's capacity in removing crossings between links in 'tangled' routes. The results confirm the dominance of the parallel version over the sequential one.

The SA method implemented here clearly benefits from a 1-interchange local search procedure, always producing enhancements to the solutions provided by the CWS algorithm. Nevertheless, the use of initial solutions generated by the CWS algorithm and the use of the same 1-interchange neighbor generation mechanism seems to force our SA procedure to be stuck in local optima, an insight which seems to be supported by the fact that similar final solutions are reached even when applying different starting parameters.

Similarly to the SA's case, our GA implementation seems to be rather poor as it does not provide results as competitive as those generated by the CWS and SA algorithms, both when the initial population is entirely composed by randomly generated solutions or when it includes solutions generated by the CWS and SA methods.

References

[1] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics (SIAM), Bologna, Italy, 2002.

[2] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2nd edition, 2004.

[3] Ted Ralphs. Branch Cut and Price Applications : Vehicle Routing. Available as http://www.coin-or.org/SYMPHONY/branchandcut/VRP/index.htm, October 2013.

[4] Sam R Thangiahl, Jean-yves Potvin, and Tong Sun. Heuristic Approaches to Vehicle Routing with Backhauls and Time Windows. *Computers & Operations Research*, 23(11):1043–1057, 1996.

[5] Harilaos N Psaraftis. k-Interchange Procedures for Local Search in a Precedence-Constrained Routing Problem. *European Journal of Operational Research*, 13(4):391–402, 1983.

[6] Ibrahim Hassan Osman. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem. *Annals of Operations Research*, 41(4):421–451, December 1993.

[7] C. A. R. Hoare. Algorithm 63: Partition. *Commun. ACM*, 4(7):321–, July 1961.

[8] Carlos Bermudez, Patricia Graglia, Natalia Stark, Carolina Salto, and Hugo Alfonso. A Comparison of Recombination Operators for Capacitated Vehicle Routing Problem. *Inteligencia Artificial*, 14(46):34–44, March 2010.

[9] S Sakamoto, T Oda, E Kulla, M Ikeda, L Barolli, and F Xhafa. Performance Analysis of WMNs Using Simulated Annealing Algorithm for Different Temperature Values. In *Complex, Intelligent, and Software Intensive Systems (CISIS), 2013 Seventh International Conference on*, pages 164–168, 2013.

[10] Jari Kytöjoki, Teemu Nuortio, Olli Bräysy, and Michel Gendreau. An Efficient Variable Neighborhood Search Heuristic for Very Large Scale Vehicle Routing Problems. *Computers & Operations Research*, 34(9):2743–2757, September 2007.

[11] Jonathan Becker, Jason Lohn, and Derek Linden. Evaluation of Genetic Algorithms in Mitigating Wireless Interference in Situ at 2.4 GHz. In *Modeling & Optimization in Mobile, Ad Hoc & Wireless Networks (WiOpt), 2013 11th International Symposium on*, pages 54–61. IEEE, 2013.

[8] Available in http://paginas.fe.up.pt/~up200400437/cvrp.zip