www.editada.org

International Journal of Combinatorial Optimization Problems and Informatics, 16(3), May-Aug 2025, 82-91. ISSN: 2007-1558. https://doi.org/10.61467/2007.1558.2025.v16i3.1143

Implementation of a Hybrid Tabu Search Algorithm with Local Search for Solving the Capacitated Vehicle Routing Problem

José Alberto Hernández-Aguilar [1], Víctor Pacheco-Valencia [1], Martín H. Cruz-Rosales [1], Julio César Ponce-Gallegos [2] and Carlos Condado-Huerta [1]

[1] FCAeI, Autonomous University of the State of Morelos, Av. Universidad 1001, 62209 Cuernavaca, Morelos, Mexico.
jose_hernandez@uaem.mx, mcr@uaem.mx, vhpacval@gmail.com

[2] Autonomous University of Aguascalientes, Av. Universidad 940, Ciudad Universitaria, 20100 Aguascalientes, Mexico.
julio.ponce@edu.uaa.mx

Abstract. The Capacitated Vehicle Routing Problem (CVRP) involves generating a route for each vehicle such that the sum of customer demands does not exceed the vehicle's capacity. Each vehicle must depart from the depot, visit its assigned customers exactly once and then return to the depot, delivering a solution in which the total distance travelled by all vehicles is minimised. In this research, randomisation and the Swap local-search algorithm are employed to create an initial solution with a high-quality neighbourhood. Subsequently, Tabu Search is applied to explore the solution space and obtain an improved result. The proposed algorithms yield feasible solutions with an approximation error of less than 10 %, and in some instances they achieve the best-known solution.

Keywords: CVRP, Tabu Search, Swap Local Search Algorithm.

Article Info
Received May 30, 2025
Accepted Jul 1, 2025

1 Introduction

The Capacitated Vehicle Routing Problem (CVRP) consists of constructing tours for vehicles with limited capacity, which start and end at a location known as the depot, to transfer goods from the depot to various customers, each with a demand. One salesman must fully satisfy the demand in a single visit.

The Vehicle Routing Problem (VRP) presents many variations due to environmental restrictions. CVRP was selected from all the variations of the VRP. CVRP is classified as a combinatorial optimization problem belonging to the class of NP-hard problems; that is, the optimal solution is obtained using algorithms that use non-polynomial time, and it cannot be obtained in a reasonable time. The solutions have different methods to solve the CVRP, divided into exact, heuristic, and metaheuristic methods. In the exact methods, dynamic programming and backtracking search, among others, are more noticeable. In the heuristic methods, constructive algorithms and insertion algorithms, among others, stand out, and in the metaheuristic methods, Simulated Annealing (SA), Tabu Search (TS), and Swarms stand out. We can also find hybridizations of metaheuristics. The problem can be approached with different objectives. Initially, it was solved by assigning clients to vehicles based solely on demand and vehicle capacity without considering the route. Once the subsets were determined, the route for each vehicle was calculated, and the total cost of the routes was summed. A new objective was then considered, which focused on grouping the closest nodes while ensuring that demand did not exceed vehicle capacity. This research proposes solving a set of CVRPs with different numbers of nodes and vehicle capacities, using three steps: 1) obtain an initial solution with the method of randomness and nearest neighbor, 2) Local Search, and 3) the TS algorithm according to (Gómez Atuesta & Rangel Carvajal, 2011).

Many optimisation problems need to be solved. Most of these problems have practical implications in science, engineering, economics and business, and are highly complex and challenging (González-Hernández et al., 2019).

Problems with similar restrictions are analysed to understand the CVRP. They can be used as a starting point; for example, the Travelling Salesman Problem (TSP) (Flood, 1956) attacks an integral part of the VRP: the TSP focuses on creating a route in

which all the customers are visited only once, and the route must start and end at the same customer. The shortest path must be considered to make this journey, provided that the route obtained at the end yields an optimal solution as reported in the literature. Over the years, different studies have applied various methodologies to solve the problem, from exact, heuristic and metaheuristic methods. Among these, implicit enumeration methods, Branch & Bound, cutting-plane or dynamic programming techniques, constructive algorithms, simulated stitching, Tabu Search (TS) and Ant Colony (AC), among others, stand out. After this problem, the Multiple Travelling Salesman Problem (MTSP) variant was introduced (Bektas, 2006). This problem incorporates a warehouse, unlike the TSP, and multiple salespeople. The objective is to construct exactly one route for each salesperson so that each customer is visited once by one of the salespeople. The route will start and end at the warehouse and can contain, at most, a limited number of customers.

Thus, our research problem is to find a solution to the CVRP (Lei, Laporte and Guo, 2011), which consists of creating routes for vehicles that will store or distribute the demand of a set of customers. Unlike the TSP and the MTSP, a new restriction has been added: sellers have become vehicles with a limited capacity, and the customers have a demand that must be collected or delivered by a single vehicle. The other variants remain the same as the MTSP.

The CVRP is essential from both theoretical and practical points of view because every company needs to acquire or deliver merchandise to many customers with limited units. In addition, consideration must be given to the time it will take to deliver to each customer, the maintenance of each vehicle and the route it must take, bearing in mind that the vehicles must return to their point of origin.

This research is a continuation of the work of (Conrado et al., 2025) and seeks to clarify the methodology used, explaining in depth the stages and the algorithms used. The justification for this project lies in the challenges faced by companies, large or small, in managing the delivery or collection of goods. Planning efficient vehicle routes that ensure each client is visited once within a specific time frame is crucial for reducing operational costs, such as fuel consumption and service time. The objective is to design an algorithm based on the TS metaheuristic to explore and evaluate feasible solutions for the CVRP, aiming to achieve a solution close to the optimal.

Hypothesis (H1) states: if a metaheuristic algorithm efficiently generates and evaluates feasible solutions for the CVRP, it delivers a solution with a cost close to the optimal one. On the contrary, the null hypothesis (H0) states that if the algorithm is not a practical alternative, it does not deliver a solution with a cost close to the optimal one.

The algorithm will be evaluated with 40 instances with the following characteristics: 10 units, 200 clients as maximum, considering small and medium instances, and 350 clients for large instances of the CVRPLIB library (Reinhelt, 2014).

The document is organized as follows: The related work is presented in the second section, and the most relevant work and a timeline are provided. Section three presents the problem description. Section four presents the proposed methodology, describing experimental procedures in depth. Section five shows the results obtained compared with those reported in the literature. Finally, conclusions and future work are presented.

2 Related work

CVRP has been studied extensively throughout history, and various methods have been implemented to obtain a solution.

Danzig and Ramser (1959) analyzed a fuel distribution problem, proposed the CVRP, and explored how fuel could be distributed to different gas stations, considering the capacity of the vehicles and the demand at each station. Their solution ensures that the company delivers on time and improves the filling times.

Laporte et al. (2000) mention several heuristics that were used to achieve acceptable results for the VRP, commenting that there are heuristics that analyze problems where the number of trucks may or may not be a stopping parameter or where a maximum cost per trip can be defined and that it can also be divided into two phases to reach the solution (first part, obtain m subsets of the original set and then carry out the construction of the route by applying an algorithm for TSP).

Naddef and Rinaldi (2002) proposed the branch-and-cut algorithm. They mentioned that tests were performed on six instances, of which 2 with 135 nodes obtained favorable results for the best-known solution (BKS).

Di Lorenzo et al. (2022) proposed the TS metaheuristic for solving VRP. Due to computational time issues, it was noted that the algorithm takes longer with more than 50 nodes. It was tested with 27 instances, of which 8 obtained the same results as the BKS, in 11 instances, they were close to the BKS, and in 8 instances, they obtained better results than the BKS.

Di Lorenzo et al. (2022) also proposed an exact algorithm (branch and cut). Twelve instances were analyzed, and a maximum solution time of 1 hour was considered. This excellent result allows instances of up to 35 nodes to be tested.

Gendreau et al. (2008) proposed using the TS metaheuristic. Fifty-eight instances were considered, with a solution time of 24 hours per instance, demonstrating that optimal results were obtained in 33 instances.

Fuellerer et al. (2009) proposed the AC metaheuristic. It was tested with 36 instances, executing ten times for each one. The results obtained with TS were compared, demonstrating that the AC is better since it only managed to win against TS in 2 instances.

Strodl et al. (2010) proposed variable neighborhood search and an exact algorithm (branch and link); it was tested with 36 instances, demonstrating that a time limit must be placed to obtain favorable results, and that the algorithm does not enter a cycle.

Leung et al. (2010) proposed SA and packing and tested 36 instances with an average of 110 minutes per solution, improving the AC algorithm.

Chen, Huang, and Dong (2010) proposed a variable neighborhood descent algorithm that uses hybrid metaheuristics. Accuracy, speed, simplicity, and flexibility are the four criteria for evaluating a metaheuristic. This algorithm combines the strengths of iterated local search and variable neighborhood descent.

Gómez-Atuesta and Rangel-Carvajal (2011) proposed the TS and SA algorithms; using ten instances, they could be within 5% of the difference against the best, of which two were equal.

Mohammed et al. (2012) proposed the Genetic Algorithm (GA) and cellular-GA. They applied the GA to give a solution with a lower traversal cost. This algorithm is implemented in very large instances, helping to reduce processing time.

Mari, Mahmudy, and Santoso (2018) proposed basic SA and enhanced SA. They mention that more iterations will increase the computational time. After five iterations, a better result is given for a CVRP, and enhanced SA gives a better solution to the CVRP.

Herdianti, Gunawan, and Komsiyah (2021) mentioned that the CVRP problem expands exponentially, but it is possible to make a tour with five nodes manually. Additionally, the particle swarm optimization method (PSO) algorithm cannot solve the VRP with many locations; the pigeon-inspired optimization (PIO) algorithm requires the position of the pigeon and the speed to be defined, and each interaction is updated. When comparing algorithms, the PIO algorithm gave a solution with a lower cost, with an approximate difference of 2.5% compared to the solution of the PSO algorithm.

Moussa (2021) mentions that the K-means algorithm was used to find subsets of the set of nodes. The algorithm assigns nodes using a centroid, modifies this centroid until all the nodes are associated, and then generates m centroids once the m subsets are obtained; the Dijkstra algorithm performs the tours of the m subsets.

Pacheco-Valencia et al. (2022, 2023) proposed a 3-phase heuristic algorithm. In phase 1, cities are partitioned into m disjoint subsets. In phase 2, feasible tours are constructed for each subset, and phase 3 improves these tours using a hill-climbing method.

3 Problem description

Figure 1 shows a CVRP instance, P-n19-k2, which includes 19 vertices: the depot, labeled as d, and 18 customers with their demands, each identified by a pair of numbers separated by a comma. The instance involves two vehicles, each with a capacity of $Q = 160$.

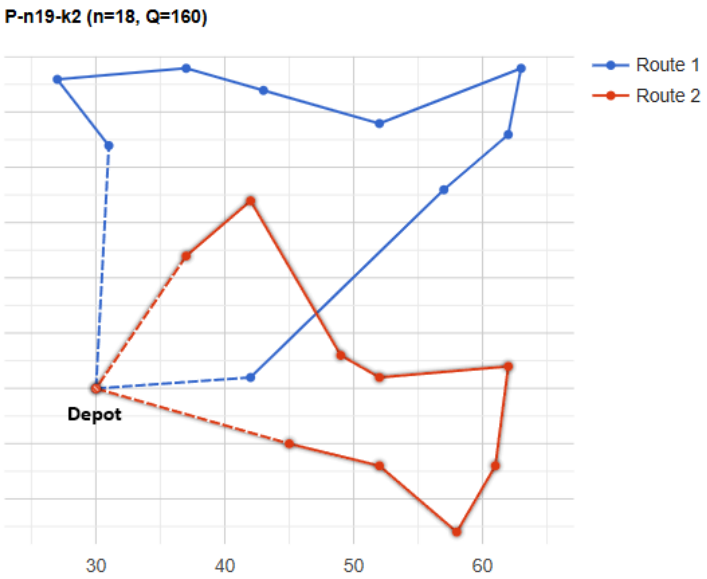


Fig. 1. CVRP instance P-n19-k2 (CVRPLIB, 2025)

The CVRP is modeled as a complete, undirected, weighted graph. $G = (V, E)$, as shown in Figure 1. The vertex set V comprises a depot d and n customers, each with a demand d_i ($d_d = 0$). Each edge $(i, j) \in E$ has a weight $w(i, j)$, representing the Euclidean distance between nodes i and j. A fleet of m vehicles, each with a capacity Q, is available to serve all customers.

A solution S is a set of m tours, T^j, for $j = 1, \ldots, m$ where each tour with cardinality $|T^j|$ starts and ends at the depot and includes a sequence of customers (i.e. $T^j = (d, t_1^j, t_2^j, \ldots, t_{|T^j|}^j, d)$).

The cost of a tour T^j is the sum of distances between consecutive vertices, including the depot, as expressed in Equation 1:

$$C(T^j) = w(d, t_1^j) + \sum_{i=1}^{|T^j|-1} w(t_i^j, t_{i+1}^j) + w\left(t_{|T^j|}^j, d\right) \quad (1)$$

The total cost of a solution S is the sum of the costs of all tours, as defined in Equation 2:

$$C(S) = \sum_{j=1}^{m} C(T^j) \quad (2)$$

The aim is to find a feasible solution S^* that minimizes the total cost, $C(S^*) = \min_S C(S)$, while satisfying the following constraints:

- All tours start and end at the depot.
- Each customer is visited exactly once by a single vehicle.
- The total demand served on a tour does not exceed the vehicle's capacity Q.
- Every vehicle must serve at least one customer.

4 Methodology

Fig. 2. Flowchart of the proposed methodology

The flowchart in Figure 2 illustrates the methodology employed. The process starts by reading a CVRP instance. The algorithm *Initial_Solution*() then generates a feasible initial solution, ensuring that the accumulated demand on each tour does not exceed the vehicle capacity. This initial solution is subsequently improved twice using the local search algorithm, *Local_Search_By_Swapping*(S), which performs vertex swaps between routes to obtain a better solution. The improved solution is passed to the Tabu Search algorithm, which executes insertion moves between routes while respecting vehicle capacity constraints. After all possible moves are explored, the algorithm delivers the final solution, aiming to approach the BKS or stay within a 5% difference from it.

```
Algorithm Initial_Solution( )
S ← ∅      //S is initially empty
while S = ∅ do:
|  j ← 1
|  while j ≤ m do:      //for each tour S^j, where j = 1,…,m
|  |  T^j ← (d)
|  |  while Σ_{i∈T^j} d_i ≤ Q do:
|  |  |  Select a vertex i randomly, with no repetition, from V \ {d}
|  |  |  T^j ← T^j ∪ {i}      //vertex i is added to tour T^j
|  |  T^j ← T^j ∪ {d}      //depot is added to the tour T^j
|  |  S ← S ∪ {T^j}      //tour T^j is added to the solution S
|  |  j ← j + 1
|  if S does not include all the vertices of V then:
|  |  S ← ∅
return S
```

Algorithm *Initial_Solution*() constructs an initial solution S by constructing m tours, where each tour starts and ends at the depot d, and the vertices are randomly selected from the set $V \setminus d$ to be added to the tours. The algorithm begins by initializing an empty solution S and setting $j = 1$, representing the current tour index. For each tour T^j, the depot d is added as the starting vertex. Then, the algorithm enters a loop where it randomly selects, without repetition, a vertex i from the set $V \setminus d$, i.e., all vertices except the depot, and appends it to the current tour T^j. This process continues as long as the total demand for the vertices in tour $\sum_{i \in T^j} d_i$ does not exceed the vehicle capacity Q. Once this condition is met, the depot d is added as the endpoint of the tour, and the tour T^j is included in the solution S. The procedure then moves to the next tour by incrementing j with one and repeating the process until all m tours are constructed. If any vertex remains uncovered after constructing all m tours, the process restarts by resetting S to empty. This ensures that the final solution comprises valid tours that collectively cover all vertices in the instance while adhering to capacity constraints. Finally, the solution S, containing all the m tours, is returned.

The initial solution, S, is improved using the algorithm *Local_Search_By_Swapping*(S) in two executions by iteratively swapping pairs of vertices within a tour or between two tours. For each vertex t_l^j in the tour T^j, where $j = 1, \dots, m$ and $l = 1, \dots, |T^j|$, another vertex $t_{l'}^{j'}$ is considered for swapping, with the condition that $j' \geq j$. If the vertices belong to the same tour ($j' = j$), then $l' > l$. If the swap reduces $C(S)$ while maintaining the feasibility of S, the vertices t_l^j and $t_{l'}^{j'}$ are swapped. This process is repeated for all pairs of vertices in the solution.

```
Algorithm Local_Search_By_Swapping(S)
j ← 1
l ← 1
while j ≤ m do:      // for each tour T^j, where j = 1,...,m
|  while l ≤ |T^j| do:      // for each vertex t_l^j ∈ T^j, where l = 1,...,|T^j|
|  |  if l ≠ |T^j| then:      // sets values for indices j' and l'
|  |  |  j' ← j
|  |  |  l' ← l + 1
|  |  else:
|  |  |  j' ← j + 1
|  |  |  l' ← 1
|  |  while j' ≤ m do:      // for each tour T^j'
|  |  |  while l' ≤ |T^j'| do:      // for each vertex t_l'^j' ∈ T^j'
|  |  |  |  // Calculate the gain or cost for swapping vertices
|  |  |  |  Δ ← −w(t_{l−1}^j, t_l^j) − w(t_l^j, t_{l+1}^j) − w(t_{l'−1}^{j'}, t_{l'}^{j'}) − w(t_{l'}^{j'}, t_{l'+1}^{j'})
|  |  |  |  Δ ← Δ + w(t_{l−1}^j, t_{l'}^{j'}) + w(t_{l'}^{j'}, t_{l+1}^j) + w(t_{l'−1}^{j'}, t_l^j) + w(t_l^j, t_{l'+1}^{j'})
|  |  |  |  if Δ < 0 ∧ Σ_{i∈T^j} d_i − d_{t_l^j} + d_{t_{l'}^{j'}} ≤ Q ∧ Σ_{i∈T^j'} d_i − d_{t_{l'}^{j'}} + d_{t_l^j} ≤ Q then:
|  |  |  |  |  // If the swap reduces C(S) while maintaining the feasibility of S
|  |  |  |  |  aux ← t_l^j      // Swap the vertices t_l^j and t_l'^j' in S
|  |  |  |  |  t_l^j ← t_{l'}^{j'}
|  |  |  |  |  t_{l'}^{j'} ← aux
|  |  |  |  |  l' ← l' + 1
|  |  |  |  j' ← j' + 1
|  |  |  l ← l + 1
|  |  j ← j + 1
return S
```

Algorithm Simple_Tabu_Search(S, $iters$, $tabuTenure$) solving the CVRP. It starts with a solution S and aims to improve it over $iters$ iterations.

Let S_{cur} be the current solution at each iteration, initially equal to a given solution S; S_{best}, the best solution found up to iteration $iter$, also initialized to S, and L is the tabu list, initially equal to an empty set, which is used to avoid the use of relocation moves (described later) that are included in it, unless they lead to an improvement in S_{best}.

The relocation strategy involves evaluating whether relocating one of the vertices each an edge $(i,j) \in S_{cur}$ iteratively, first considering i and then j, starting with the highest-cost edges and progressing to lower-cost edges. In each iteration, the first feasible relocation move found is selected, provided it satisfies one of the following conditions: *i*) the move is not in L and generates a solution with a lower cost than $C(S_{cur})$, or *ii*) the move is in L and generates a solution with a lower cost than $C(S_{best})$ (aspiration criterion). If no cost-reducing move is available after evaluating the relocation of all vertices in $C(S_{cur})$, the relocation move that increases the cost the least is performed.

In each iteration $iter$, a set of edges E_S is constructed, containing the edges of the current solution S_{cur}, ordered in non-increasing order by their weights. The variables M_{im} and C_{im} are declared to store the movement that minimally increases the cost of S_{cur} and the resulting cost of that relocation movement, respectively. These variables are initialized as an empty set and a very large value, and are used in cases where no relocation movements reduce the cost of S_{cur}. A binary variable $flag$, is also declared to indicate whether a valid movement has been performed ($flag = True$) during the iteration.

The evaluation of possible movements is carried out by exploring the edges in E_S using the index i. For each edge, the relocation of its vertices, assigned to the variable ν, is analyzed by traversing the vertices in $E_{S_i}$with the index j (i.e., $\nu \leftarrow E_{S_{ji}}$). Then, a new set of edges E'_S is constructed, containing those connected to ν, ordered in non-decreasing order by their weights.

Algorithm: Simple_Tabu_Search(S, $iters$, $tabuTenure$)

```
S_cur ← S        // current solution
S_best ← S       // best solution found
L ← ∅        // Tabu list
iter ← 0       // number of iterations executed
while iter ≤ iters do:      // as long as no "iters" iterations have been executed
    E_S ← { (i,j) |(i,j) ∈ S_cur; i,j ∈ V }
    Sort E_S in non-increasing order by weights      // Initially |E_S| = n + m edges
    M_im ← ∅       // variable to store minimum insertion move
    C_im ← ∞       // cost if M_im is realized
    flag ← False       // True if movement was done in the current iteration
    i ← 1
    while flag = False ∧ i ≤ |E_S| do:      // index i traverses each edge in E_S
        j ← 1
        while flag = False ∧ j ≤ 2 do:
            v ← E_Sji       //v: Vertex in position j in edge E_Si
            E'_S ← { (v,l) | (v,l) ∈ E; v,l ∈ V; v ≠ l }
            Sort E'_S in non-decreasing order by their weights //Initially |E'_S| = n − 1 edges
            k ← 1
            while flag = False ∧ k ≤ |E'_S| do:      //index k traverses each edge in E'_S
                (v,ω) ← E'_Sk       //(v,ω): Edge in position k in E'_S
                S^α, p ← Locate the tour S^α and position p of vertex v within S_cur
                S^β, q ← Locate the tour S^β and position q of vertex ω within S_cur
                if ∑_{ι∈S^β} d_ι + d_v ≤ Q then:
                    M_1 ← (1, t^α_{p−1}, v, t^α_{p+1}, t^β_{q−1}, ω)       //movement_1 (M_im1 = 1)
                    M_2 ← (2, t^α_{p−1}, v, t^α_{p+1}, ω, t^β_{q+1})       //movement_2 (M_im1 = 2)
                    ΔG ← −w(t^α_{p−1}, v) − w(v, t^α_{p+1}) + w(t^α_{p−1}, t^α_{p+1})      //gain for removing v from S^α
                    ΔC_1 ← w(t^β_{q−1}, v) + w(v, ω) − w(t^β_{q−1}, ω)      //cost for movement_1
                    ΔC_2 ← w(ω, v) + w(v, t^β_{q+1}) − w(ω, s^β_{q+1})      //cost for movement_2
                    if ΔG + ΔC_1 < 0 ∨ ΔG + ΔC_2 < 0 then:      //if M_1 or M_2 improve S_cur
                        if (M_1 ∉ L ∧ ΔC_1 ≤ ΔC_2) ∨ (M_1 ∈ L ∧ C(S_cur) + ΔG + ΔC_1 < C(S_best)) then:
                            T^α ← T^α \ {v}
                            T^β ← Insert v into S^β between t^β_{q−1} and ω
                            L ← L ∪ M_1      // If |L| > tabuTenure, remove the oldest item in L
                            flag ← True
                        else if (M_2 ∉ L ∧ ΔC_2 < ΔC_1) ∨ (M_2 ∈ L ∧ C(S_cur) + ΔG + ΔC_2 < C(S_best)) then:
                            T^α ← T^α \ {v}
                            T^β ← Insert v into T^β between ω and t^β_{q+1}
                            L ← L ∪ M_2      // If |L| > tabuTenure, remove the oldest item in L
                            flag ← True
                        else:      //if none of the moves, M_1 or M_2, improve S_cur
                            if ΔC_1 ≤ ΔC_2 ∧ ΔG + ΔC_1 < C_im then:
                                M_im ← M_1
                                C_im ← ΔG + ΔC_1
                            else if ΔC_2 < ΔC_1 ∧ ΔG + ΔC_2 < C_im then:
                                M_im ← M_2
                                C_im ← ΔG + ΔC_2
                k ← k + 1
            j ← j + 1
        i ← i + 1
    if flag = False ∧ C_im ≠ ∞ then:
        if M_im1 = 1 then:
            (1, t^α_{p−1}, v, t^α_{p+1}, t^β_{q−1}, ω) ← M_im
            S^α ← S^α \ {v}
            S^β ← Insert v into S^β between t^β_{q−1} and ω
        else:
            (2, t^α_{p−1}, v, t^α_{p+1}, ω, t^β_{q+1}) ← M_im
            S^α ← S^α \ {v}
        L ← L ∪ M_im      // If |L| > tabuTenure, remove the oldest item in L
    if C(S_cur) < C(S_best) then:
        S_best ← S_cur
    iter ← iter + 1
return S_best
```

The edges in E'_S are traversed using the index k, evaluating every possible relocation movement for v. During each evaluation, the possible relocations of v to new positions in different routes are determined, ensuring that the vehicle's capacity Q is not exceeded. For each potential movement, M_1 and M_2, the gain, ΔG, from removing v from its current position p within the route S^α is calculated, as well as the costs, ΔC_1 and ΔC_2, associated with its insertion into the new positions: between vertices t^β_{q-1} and t^β_q, and t^β_q and t^β_{q+1}, respectively, in route S^β. If any movement is not in the tabu list, L, and improves the cost of S_{cur}, its execution, it is considered; otherwise, if the move is in L and the cost obtained for that move is less than the cost of S_{best}, then the aspiration criterion is used to perform that move. If no movement improves S_{cur}, the movement that least increases the cost is stored in M_{im}, and its associated cost is stored in C_{im}.

If, after exploring all possible moves, none is found that improves S_{cur}, then the movement stored in M_{im} is performed. Finally, if S_{cur} results in a cost lower than the best solution found so far, S_{best}, the latter is updated with S_{cur}.

Once all iterations are completed, the algorithm returns S_{best} as the best solution found.

5 Results and discussion

The CVRP data instances were obtained from the CVRP Library (Reinhelt, 2014). Each instance has a file name; its last digits refer to the number of nodes considered. The internal structure of this file has the file name, name of the author, type (CVRP), dimension (number of nodes), capacity, coordinates of the nodes section, demand section, and depot section. The nodes section has the ID number of the node, X, and Y coordinates. The demand section has the number of the node and its demand. The depot section has depot identifiers. This information was read and stored in a data node structure formed with ID, X, $Y$and demand.

The hybrid algorithm developed in this project was coded in Python 3.9 and executed on a DELL Inspiron 15 3000 laptop with the following specifications: Intel(R) Core(TM) i3-1005G1 CPU at 1.20GHz, 8GB RAM, running Windows 11 Home Single Language.

Table 1 shows the results obtained using seven Augerat instances taken from the CVRP library and solved by our proposal. The first column lists the instance name, the second column shows the BKS for each instance, the third column provides the cost obtained using the Random-Insertion (RI) plus Tabu Search (TS), and the fourth column displays the percentage difference between the RI + TS algorithm and its BKS. The fifth column presents the cost obtained with RI+ TS + Intensification Stage, and the sixth column shows the percentage difference between the results of this previous approach and its BKS. The average % difference is shown at the end of Table 1, 14.52 for RI + TS, and 5.45 for the proposed methodology.

Table 1. Comparison BKS versus Random-Insertion(RI) + TS, versus RI+TS+ Intensification

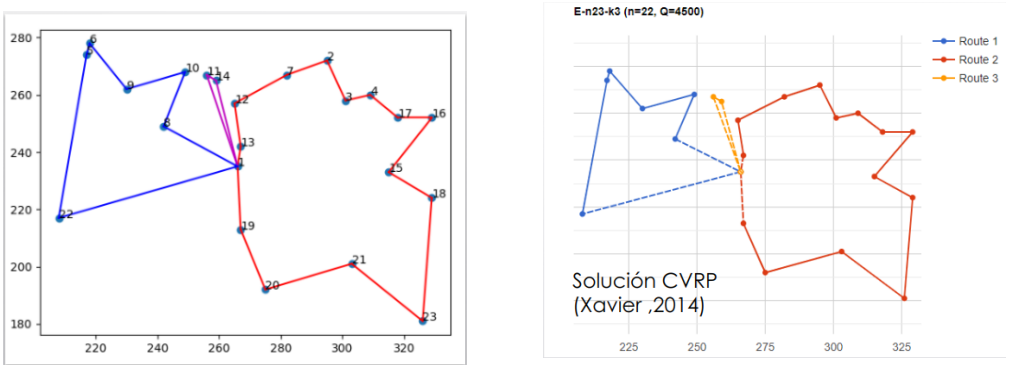
Instance	BKS	RI + TS	% difference	RI +TS + Intensification	% Difference
A-n32-k5	784.000	839.691	7.103	792.612	1.098
A-n33-k5	661.000	721.092	9.091	685.956	3.775
A-n33-k6	742.000	889.327	19.855	746.026	0.543
A-n34-k5	778.000	868.493	11.631	834.986	7.325
A-n36-k5	799.000	859.872	7.619	854.577	6.956
A-n37-k5	669.000	852.354	27.407	725.681	8.472
A-n37-k6	949.000	1128.775	18.944	1043.943	10.005
		Average %	14.52	Average %	**5.45**

Table 2 shows the results obtained using the instances taken from the CVRP library and solved by Gómez-Atuesta and Rangel-Carvajal (2011). The first column lists the instance name, the second column shows the BKS for the instance, the third column provides the cost obtained using the proposed hybrid algorithm, and the fourth column displays the percentage difference between the proposed hybrid algorithm and the BKS. The fifth column presents the cost reported by Gómez-Atuesta and Rangel-Carvajal (2011), and finally, the sixth column shows the percentage difference between the results of these authors and the BKS.

Table 2. Comparison versus Gomez-Atuesta and Rangel-Carvajal (2011)

Instance	BKS	Hybrid Algorithm	% difference	Gomez et al. 2011	% difference
E-n16-k3	262	278.726	6.384	284.23	8.485
E-n23-k3	569	568.562	-0.077	568.56	-0.077
E-n33-k4	845	868.644	2.798	844.29	-0.084
P-n50-k10	696	776.565	11.573	732.50	5.244
P-n22-k2	216	221.427	2.513	217.85	0.856
P-n55-k10	694	730.93	5.321	720.76	3.856
M-n200-k17	1373	1559.536	13.586	1579.09	15.009

Table 2 shows that our hybrid approach obtained two better solutions than the one provided by (Gómez-Atuesta & Rangel-Carvajal, 2011) and obtained the BKS for E-n23-k3. This solution is shown in Fig. 6a. Conversely, the proposed solution provided solutions close to BKS, for instance, E-n33-k4 and P-n22-k3.



(a) (b)

Fig. 3. The Best-Known Solution (BKS) for instance E-n23-k3, a) proposed, b) reported in the literature

As shown in Figure 3, the proposed solution (a), for instance, E-n23-k3, is equal to the BKS (b) reported in the literature.

6 Conclusions

The proposed hybrid Tabu Search algorithm addresses the problem in stages, allowing feasible results to be obtained. In some cases, the deviation from the best-known solution is below 5 % and even zero, but in others it exceeds 10 %. Based on the results obtained, we conclude that H1 is true and the objective was met.

For future work, we propose combining the swap and insertion algorithms within Tabu Search. We intend to test the algorithm on real-world problems; in particular, we are interested in applying the methodology to smart cities, such as smart waste collection, product delivery by a fleet of electric vehicles and multi-layer transportation systems. Our aim is to provide real-time mobile apps to track vehicles and their routes and to estimate arrival times in these scenarios.

References

Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega, 34*(3), 209–219.

CVRPLIB. (2025). *Instance: P-n19-k2*. Recuperado 27 de abril de 2025, de http://vrp.galgos.inf.puc-rio.br/index.php/en/plotted-instances?data=P-n19-k2

Chen, P., Huang, H. K., & Dong, X. Y. (2010). Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. *Expert Systems with Applications, 37*(2), 1620–1627.

Condado-Huerta, C., Hernández-Aguilar, J. A., Cruz-Rosales, M. H., Pacheco-Valencia, V., & Ponce-Gallegos, J. C. (2025). Implementation of a hybrid tabu search algorithm for solving the capacitated vehicle routing problem. En L. Martínez-Villaseñor, G. Ochoa-Ruiz, M. Montes Rivera, M. L. Barrón-Estrada, & H. G. Acosta-Mesa (Eds.), *Advances in Computational Intelligence: MICAI 2024 International Workshops* (Lecture Notes in Computer Science, vol. 15464). Springer. https://doi.org/10.1007/978-3-031-83879-8_19

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science, 6*(1), 80–91.

Di Lorenzo, D., Bianconcini, T., Taccari, L., Gualtieri, M., Raiconi, P., & Lori, A. (2022, agosto). Ten years of Routist: Vehicle routing lessons learned from practice. En *International Conference on Optimization and Decision Science* (pp. 265–276). Springer Nature Switzerland.

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research, 4*(1), 61–75.

Fuellerer, G., Doerner, K. F., Hartl, R. F., & Iori, M. (2009). Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers & Operations Research, 36*(3), 655–673.

Gendreau, M., Iori, M., Laporte, G., & Martello, S. (2008). A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks: An International Journal, 51*(1), 4–18.

Gómez Atuesta, D. F., & Rangel Carvajal, C. E. (2011). Formular las metaheurísticas búsqueda tabú y recocido simulado para la solución del CVRP. Recuperado 27 de abril de 2025, de https://noesis.uis.edu.co/server/api/core/bitstreams/66177819-d48b-4699-ab89-87b5e9f38e45/content

González-Hernández, I. J., Granillo-Macías, R., Martínez-Flores, J. L., Sánchez-Partida, D., & Gibaja-Romero, D. E. (2019). Hybrid model to design an agro-food distribution network considering food quality. *International Journal of Industrial Engineering, 26*(4).

Herdianti, W., Gunawan, A. A., & Komsiyah, S. (2021). Distribution cost optimization using pigeon inspired optimization method with reverse learning mechanism. *Procedia Computer Science, 179*, 920–929.

Laporte, G., Gendreau, M., Potvin, J. Y., & Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research, 7*(4–5), 285–300.

Lei, H., Laporte, G., & Guo, B. (2011). The capacitated vehicle routing problem with stochastic demands and time windows. *Computers & Operations Research, 38*(12), 1775–1783.

Leung, S. C., Zheng, J., Zhang, D., & Zhou, X. (2010). Simulated annealing for the vehicle routing problem with two-dimensional loading constraints. *Flexible Services and Manufacturing Journal, 22*, 61–82.

Mari, F., Mahmudy, W. F., & Santoso, P. B. (2018). An improved simulated annealing for the capacitated vehicle routing problem (CVRP). *Jurnal Ilmiah Kursor, 9*(3).

Mohammed, M. A., Ahmad, M. S., & Mostafa, S. A. (2012, junio). Using genetic algorithm in implementing capacitated vehicle routing problem. En *2012 International Conference on Computer & Information Science (ICCIS)* (Vol. 1, pp. 257–262). IEEE.

Moussa, H. (2021). Using recursive KMeans and Dijkstra algorithm to solve CVRP [Preprint]. arXiv. Recuperado 27 de abril de 2025, de https://arxiv.org/abs/2102.00567

Naddef, D., & Rinaldi, G. (2002). Branch-and-cut algorithms for the capacitated VRP. En *The Vehicle Routing Problem* (pp. 53–84). Society for Industrial and Applied Mathematics.

Pacheco-Valencia, V. H., Vakhania, N., Hernández-Mira, F. Á., & Hernández-Aguilar, J. A. (2022). A multi-phase method for Euclidean traveling salesman problems. *Axioms, 11*(9), 439. https://doi.org/10.3390/axioms11090439

Pacheco-Valencia, V. H., Vakhania, N., & Hernández-Aguilar, J. A. (2023, julio). An algorithm to solve the Euclidean single-depot bounded multiple traveling salesman problem. En *2023 IEEE World Conference on Applied Intelligence and Computing (AIC)* (pp. 7–12). IEEE. https://doi.org/10.1109/AIC57670.2023.10263920

Reinhelt, G. (2014). TSPLIB: A library of sample instances for the TSP (and related problems) from various sources and of various types. Recuperado 27 de abril de 2025, de http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Strodl, J., Doerner, K. F., Tricoire, F., & Hartl, R. F. (2010). On index structures in hybrid metaheuristics for routing problems with hard feasibility checks: An application to the two-dimensional loading vehicle routing problem. En Hybrid Metaheuristics: 7th International Workshop, HM 2010, Vienna, Austria, October 1–2, 2010 (pp. 160–173). Springer Berlin Heidelberg.