Evolutionary Algorithms for the Rout Optimization for Cities in Minas Gerais

Oswaldo Rodrigues de Barros Neto, Milton Pereira Bravo Neto,
Marcelle Christine Aquino Silva, Gabriela Nunes Lopes
Universidade Federal de Minas Gerais, Belo Horizonte, Brasil
E-mails: osbarros@ufmg.br, milbravo@ufmg.br, marcelleaquino@ufmg, gabrielanuneslopes@ufmg.br

***Resumo*—The Traveling Salesman Problem (TSP) is one of the most well-known challenges in combinatorial optimization, extensively studied due to its theoretical and practical relevance. This work addresses a real-world instance of the TSP, applied to the 853 cities of Minas Gerais, using evolutionary algorithms and heuristics to find near-optimal solutions. Three main methods were implemented: Tabu Search, Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO), along with auxiliary algorithms such as Kruskal's Algorithm and the Bellmore and Nemhauser Heuristic. The results demonstrated the efficiency of Tabu Search in finding a significantly lower-cost solution (26,482.811 km) compared to the other algorithms. This study also proposes suggestions for improving these approaches, highlighting the potential of evolutionary techniques in solving large-scale combinatorial problems.**

***Palavras-chave*—Traveling Salesman Problem (TSP), Tabu Search, Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Kruskal's Algorithm, Bellmore and Nemhauser Heuristic, Evolutionary Computation, Combinatorial Optimization.**

I. Introduction

The Traveling Salesman Problem (TSP) is a classic routing challenge that consists of finding the shortest route to visit all nodes in a graph exactly once, returning to the starting node at the end of the journey.

This problem has numerous practical applications in different contexts and scenarios, and it is closely related to many real-world issues, which motivates its study. Additionally, the TSP is a challenging problem whose exact solution is difficult to obtain, being classified as NP-hard according to Karp (1975) [1].

Throughout this article, a comprehensive analysis of the problem is presented, considering its large magnitude and the inherent difficulties of implementation. This includes everything from obtaining the real-world data used, through data manipulation, to the execution of the selected algorithms.

The evolutionary algorithms applied to address the problem were Tabu Search, Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO). Each of these methods required specific adaptations and distinct strategies to handle the particularities of the problem.

This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

In this work, a practical variant of the TSP was selected, aiming to determine the minimum distance required to visit the 853 municipalities of the state of Minas Gerais, Brazil.

Moreover, auxiliary heuristics and algorithms were employed to compare the obtained results and assist in the search for the best solution. Among them, Kruskal's Algorithm for obtaining the minimum spanning tree and the Bellmore and Nemhauser Heuristic are noteworthy.

II. Methodology

This section presents the methodology of the Traveling Salesman Problem (TSP) applied to the 853 cities of Minas Gerais, Brazil. The problem data were obtained through the Google Maps Platform API, and the cities were modeled as a fully connected graph, with distances between each pair of municipalities stored in a symmetric matrix.

To explore and evaluate diverse optimization strategies, the study employs both classical and bio-inspired algorithms. Kruskal's algorithm was used to construct a Minimum Spanning Tree (MST) for initial analysis of the connectivity structure. The Bellmore and Nemhauser heuristic served as a baseline greedy approach. More sophisticated techniques, including Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Tabu Search, were implemented to handle the combinatorial complexity of the TSP and seek high-quality approximate solutions. Each algorithm was adapted and parameterized to cope with the large solution space while balancing exploration and exploitation.

A. Database Used

To obtain the problem data, the API from the *Google Maps Platform*, a cloud service provided by Google [2], was used. This service allows routing between two or more locations, considering various factors. In this case, a route for cars was considered, disregarding traffic conditions and prioritizing paved roads.

The problem was modeled as a fully connected graph, where the 853 municipalities were represented as nodes, and simply passing through a city without explicitly visiting it, was not considered a visit. The resulting distance matrix was organized as a symmetric matrix of dimensions (853×853). The main diagonal, representing the distance from a city to itself, was populated with sufficiently large values to avoid erroneous positive influences on the results. This strategy ensures that the algorithms naturally avoid such edges. To optimize API usage, only the upper triangular part of the matrix was initially filled and then mirrored to the lower part, reducing the number of queries to the service and, consequently, the associated costs.

Modeling the graph as fully connected brought the problem closer to reality, where traveling between cities may involve passing through others without visiting them. This approach also simplified data handling, as all edges between graph vertices were considered viable. On the other hand, it significantly expanded the search space due to the completeness of the distance matrix. Nevertheless, the simplicity in data handling provided a solid foundation for the execution of the algorithms and the analysis of the results.

B. Kruskal's Algorithm for Minimum Spanning Tree (MST)

Kruskal's algorithm, used for constructing the Minimum Spanning Tree (MST), is an example of a greedy algorithm. A greedy algorithm follows a strategy that makes local decisions at each step, always choosing the best possible solution at that moment without considering future implications. In the case of Kruskal's algorithm, it selects the edges of the graph in order, starting with the one with the lowest cost, and adds them to the spanning tree as long as they do not form cycles.

The process continues until the number of selected edges equals the number of vertices minus one, ensuring that all vertices are connected. This approach is efficient because it exploits the problem structure to minimize the total cost of the tree while maintaining implementation simplicity. Due to its greedy nature, Kruskal's algorithm is widely used in graph and optimization problems.

The MST implementation using Kruskal's algorithm was carried out utilizing the *minimum_spanning_tree* function from the Python *scipy* library, with the distance matrix as its input.

C. Bellmore and Nemhauser Heuristic

The Bellmore and Nemhauser (B&N) Heuristic, described in [1], consists of choosing the nearest unvisited city from the last visited city — that is, always taking the minimum step — provided the city has not yet been visited. It can be described as shown in the pseudocode in Algorithm 1.

D. Particle Swarm Optimization Algorithm

This algorithm, also known as PSO (Particle Swarm Optimization), seeks to replicate the behavior of certain animal populations in nature, such as schools of fish swimming, flocks of birds flying, or swarms of bees. A key characteristic of this algorithm is the exchange of information among the particles in the population, meaning they cooperate with each other rather than solely compete, as in other algorithms [3].

Algorithm 1 Bellmore and Nemhauser Heuristic

```
1:  $n\_city \leftarrow$ Number of cities
2:  $start\_city \leftarrow$ Random starting city
3:  $unvisited[\ldots] \leftarrow$ All cities except $start\_city$
4:  $visited[\ldots] \leftarrow start\_city$
5:  while $length(visited) < n\_city$ do
6:      $city \leftarrow last(visited)$
7:      $next\_city \leftarrow$ Nearest city to $city$ in $unvisited$
8:      Append $next\_city$ to $visited$
9:      Remove $next\_city$ from $unvisited$
10: end while
11: Return: $visited$
```

In PSO, the behavior over the course of execution is governed by (1):

$$v_i = w \cdot v_i + c_1 \cdot \xi \cdot (pbest_i - x_i) + c_2 \cdot \xi \cdot (gbest - x_i) \quad (1)$$

$$x_i = x_i + v_i$$

Where the terms in (1) are:

- v_i: particle's movement velocity that determines its new position.
- w: inertia weight, which controls how much the particle continues in its current direction.
- c_1: cognitive weight, which controls the influence of $pbest$, the best value found by this particle.
- c_2: social learning weight, which controls the influence of $gbest$, the best value found among all particles.
- ξ: a random value between 0 and 1, introducing stochasticity to the particles' behavior.

It is also important to define the population size. A small population has low diversity and a higher chance of stagnating in a local minimum, while a larger population mitigates this issue but increases execution time linearly.

However, this PSO formulation does not directly fit the TSP and needs to be adapted.

The problem is viewed as selecting a permutation of the 853 cities of Minas Gerais that results in the shortest route. That is, a path vector C of dimension 853, where cities are ordered according to the visitation sequence. The edge between $C[i]$ and $C[i+1]$, connecting city i to city $i+1$, has its distance retrieved from the distance matrix D at coordinates $D[i][i+1]$. The total distance is computed by summing these distances, including the return from $C[852]$ to $C[0]$.

To adapt PSO to this problem, velocity is interpreted as position swaps within the path C. Three types of swaps are defined:

- Swap within itself: two random positions in particle i are chosen, and the cities in those positions are swapped.

- Swap with $pbest_i$: a random city from the particle's best-known path ($pbest_i$) is selected, and its position is replicated in the current particle by swapping accordingly.
- Swap with $gbest$: the same procedure as with $pbest_i$, but using $gbest$.

Thus, moving toward $pbest_i$ or $gbest$ can be seen as making the path increasingly resemble those solutions.

The number of swaps performed is proportional to the directional contribution to the resulting velocity. These contributions are detailed in (1) and can be specifically described by (2) and (3):

$$v_{pbest} = c_1 \cdot rnd \cdot (pbest_i - x_i) \tag{2}$$

$$v_{gbest} = c_2 \cdot rnd \cdot (gbest - x_i) \tag{3}$$

$$v_i = w \cdot v_i + v_{pbest_i} + v_{gbest} \tag{4}$$

The number of swaps i_{pbest} and i_{gbest} is limited by $max\_changes$, which is passed as a parameter to the algorithm:

$$i_{pbest} = abs(int(100 \cdot (1 - (v_i - v_{pbest})/(v_i)))) \tag{5}$$

$$i_{gbest} = abs(int(100 \cdot (1 - (v_i - v_{gbest})/(v_i)))) \tag{6}$$

Thus, when v_{pbest} or v_{gbest} are null, no swaps are performed in those directions.

The swap logic for $pbest$ is shown in the pseudocode in Algorithm 2, which is similarly applied for $gbest$.

Algorithm 2 Particle Swaps

```
1: for $i = 1, 2, \ldots, i_{pbest}$ do
2:     $city \leftarrow$ Random city
3:     $m \leftarrow$ Position of $city$ in $pbest$
4:     $k \leftarrow$ Position of $city$ in the current particle
5:     Swap positions $m$ and $k$ in the particle
6: end for
```

Additionally, a mechanism was inserted into the algorithm to avoid stagnation in a local minimum, which would occur when all particles become very similar to $gbest$. When this happens, the velocity v_i decreases and tends to zero since, with $0 < w < 1$ and $V_{gbest} = V_{pbest} = 0$, (4) simplifies to $v_i = w \cdot v_i$, causing v_i to eventually approach zero.

Thus, whenever $v_i < 0.0001$, a random swap is performed within the particle itself, forcing a movement.

The complete implementation of the algorithm can be found in the repository [4], where the file "pso_tsp.py" starts with random particles, and "pso_tsp_BeN.py" uses the B&N heuristic to define the initial particles of the population.

E. Ant Colony Optimization Algorithm (ACO)

The Ant Colony Optimization Algorithm (ACO) was inspired by the behavior of real ants in search of food. These ants deposit pheromones along the paths they travel, creating trails that can be followed by others, with the intensity of the pheromone guiding the choice of shorter paths over time [5].

The first formal proposal of ACO was made by Dorigo [6], and since then, it has been applied to various combinatorial problems, such as the Traveling Salesman Problem (TSP), vehicle routing problems, and network optimization. Its effectiveness lies in the cooperation between agents and in balancing the exploration of new solutions with the intensification around the best solutions found.

In the context of the TSP applied to the state of Minas Gerais, ACO was adapted to handle the high dimensionality of the problem (853 cities), leveraging its ability to converge to near-optimal solutions.

1) Algorithm Modeling: The probability of an ant k choosing city j from city i is defined by (7):

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{j \in J} \tau_{ij}^\alpha \cdot \eta_{ij}^\beta}, \tag{7}$$

where:

- p_{ij}^k: probability of ant k choosing city j from city i;
- τ_{ij}: amount of pheromone on the edge between cities i and j;
- η_{ij}: heuristic representing the "desirability" of moving from i to j, defined as $1/d_{ij}$, where d_{ij} is the distance between the cities;
- α: parameter controlling the influence of the pheromone;
- β: parameter controlling the influence of the heuristic.

The pheromone update is governed by (8):

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k, \tag{8}$$

where:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & \text{if edge } (i,j) \text{ belongs to } S_k, \\ 0, & \text{otherwise.} \end{cases}$$

with:

- ρ: pheromone evaporation rate;
- Q: constant defining the amount of pheromone deposited;
- L_k: length of the path found by ant k;
- S_k: set of edges that make up path k.

2) Parameters and Justifications: The parameters chosen for ACO were:

- $\alpha = 1.0$: Provides a balanced influence of pheromone, allowing previously successful trails to be considered without monopolizing the choices.

- $\beta = 2.0$: Prioritizes the "desirability" of edges based on distances, which is essential in routing problems like the TSP.
- $\rho = 0.1$: A low evaporation rate to preserve relevant historical information over several iterations.
- $Q = 1000$: Controls the amount of pheromone deposited, calibrated to balance the contribution of each partial solution.
- *Number of ants (m) = 853*: Chosen to match the number of cities, ensuring broad exploration of the distance matrix.
- *Number of iterations = 2000*: A high value to allow convergence, considering the large number of cities.

These values were defined considering the problem scale and the need for a balance between exploration and intensification.

3) Implementation and Results: The ACO was implemented in Python, using the symmetric distance matrix of the 853 cities. The main function computes the transition probabilities, constructs the paths for all ants, and updates the pheromones at each iteration. Further details, including the code logic, can be found in the repository [4], specifically in the file "ACO.py". The results showed the effectiveness of ACO in generating approximate solutions to the problem, with the best paths recorded in files for later analysis.

4) Algorithm Pseudocode: The pseudocode for ACO can be described as follows:

Algorithm 3 Ant Colony Optimization (ACO) Algorithm

```
1: for $iteration = 1, 2, \ldots, max\_iterations$ do
2:     Initialize the paths for all ants.
3:     for each ant $k$ do
4:         Set the initial city $i$.
5:         while there are unvisited cities do
6:             Calculate $p_{ij}^k$ for all unvisited cities $j$.
7:             Choose the next city $j$ based on $p_{ij}^k$.
8:             Update ant $k$'s path.
9:         end while
10:        Complete the cycle by returning to the initial city.
11:    end for
12:    Update pheromones on the edges based on the traveled paths.
13:    Apply pheromone evaporation $(1 - \rho)$.
14: end for
15: Return the best path found.
```

F. Tabu Search Algorithm

Tabu Search is a heuristic optimization algorithm introduced by Fred Glover in 1986 [7]. This algorithm is widely used to solve combinatorial optimization problems, such as the Traveling Salesman Problem (TSP) and other NP-hard problems, based on a memory structure called the tabu list — a short-term memory that records the recent moves of the algorithm.

The term "tabu," originating from Polynesian language, refers to something sacred, special, forbidden, or dangerous, and in philosophy, anthropology, and sociology, it is associated with prohibitions or restrictions.

In the context of Tabu Search, this concept is applied to avoid unproductive paths, such as getting trapped in local optima. Just like in social contexts where prohibitions can be lifted when necessary, in Tabu Search the "tabus" can be removed under certain circumstances. The main connection with the traditional use of the term comes from the idea of evolutionary memory, where forbidden elements can have their status changed over time depending on the circumstances [8].

The purpose of the tabu list is to prevent the algorithm from reverting to previously visited solutions, thus avoiding cycles and enabling the exploration of new areas of the solution space.

In general terms, this metaheuristic explores the solution space through successive moves between neighboring solutions. This approach aims to find optimal or near-optimal solutions by minimizing an objective function.

The objective function for the TSP context can be represented by the following formula:

$$\sum_{i=1}^{n-1} d(s_i, s_{i+1}) + d(s_n, s_1) \tag{9}$$

where:

- $f(s)$: the total distance traveled.
- $d(s_i, s_{i+1})$: the distance between city s_i and the next city s_{i+1} in the path.
- s_n: the last city and s_1 is the starting city, closing the cycle.

1) Algorithm and Main Functions: The Tabu Search implementation combined techniques for generating optimized initial solutions, neighborhood exploration, and dynamic tabu list updates to efficiently solve the problem. The inclusion of strategies such as aspiration criteria and tabu list size adjustments enhanced the algorithm's ability to find solutions close to the global optimum, promoting a balance between exploration and intensification of the search space.

Main functions:

- $objective\_function$: Calculates the total distance of a path.
- $generate\_initial\_solution$: Generates an initial solution using the Nearest Neighbor Algorithm (B&N heuristic).
- $generate\_neighborhood\_2opt$: Generates neighborhoods using the 2-opt optimization technique.
- $tabu\_search$: Implements the core logic of the Tabu Search, including initializing the tabu list, exploring neighborhoods, evaluating solutions, and updating the list.

For generating the initial solution, the function $generate\_initial\_solution$ uses the Nearest Neighbor Algorithm (another name for the B&N heuristic), selecting a random starting city. This strategy provides a reasonably good initial solution, serving as a starting point for the optimized search.

The neighborhood is generated using the local optimization technique known as 2-opt, which inverts the order of a segment of the path, creating new solutions from small modifications and allowing the exploration of neighboring solutions that may improve the current solution effectively.

The main function, $tabu\_search$, fully implements the Tabu Search logic. Initially, the current solution is set as the best solution found so far, and the tabu list is initialized. The main search loop involves generating neighborhoods, evaluating the generated solutions, and selecting the best solution, provided it is not in the tabu list or it satisfies the aspiration criteria, which allow accepting tabu solutions if they represent a substantial improvement over the current solution.

When a better solution than the best known so far is found, it replaces the current best solution, and the local optima counter is reset. Otherwise, the counter is incremented, and the tabu list size may be adjusted to promote diversification. The tabu list is periodically updated, and the results are recorded in a file.

G. Tools and Implementation

The algorithms were implemented using the Python programming language. Execution was carried out in a standard computational environment.

III. Results and Discussion

A. Kruskal's Algorithm for Minimum Spanning Tree

The limit provided by the minimum spanning tree was used for comparison with the results obtained.

By defining a Hamiltonian cycle Φ in a graph $G = (N, A)$, and removing one of the vertices from Φ, a path in G is obtained. If T_{GM} is the minimum spanning tree of the subgraph resulting from the removal of the vertex, then it serves as a lower bound for the resulting path. Relaxing this consideration to the graph G and obtaining a minimum spanning tree for it also provides a lower bound [1].

However, it should be noted that this tree represents an *illegal tour* for the TSP [1] and serves only as a comparative value for minimization purposes.

To generate the minimum spanning tree, Kruskal's algorithm was applied, resulting in:

$$\textbf{21758153 } \text{meters or } \textbf{21758.1 } \text{kilometers.} \tag{10}$$

Thus, no solution or distance found can be smaller than this defined lower limit.

B. Bellmore and Nemhauser Heuristic

This greedy heuristic was of great importance for evaluating the performance of the evolutionary algorithms implemented. Since we do not have guarantees of optimality in these algorithms, it was necessary to establish known bounds to properly assess and compare the results.

The B&N heuristic was executed starting from all 853 cities of Minas Gerais. Among the results, the one with the shortest distance was chosen to serve as our upper bound for evaluating the results, ensuring that we would aim to achieve a result at least better than this.

The value found was:

$$\textbf{31197415 } \text{meters or } \textbf{31197.4 } \text{kilometers.} \tag{11}$$

C. Particle Swarm Optimization Algorithm

The PSO algorithm was initially executed starting from completely random particles (file "pso_tsp.py"), using the following parameters:

- $num\_particles = 50$
- $max\_iterations = 1000000$
- $w = 0.6$
- $c_1 = 2$
- $c_2 = 2$
- $max\_swaps = 20$

The evolution of the results can be seen in Figure 1.

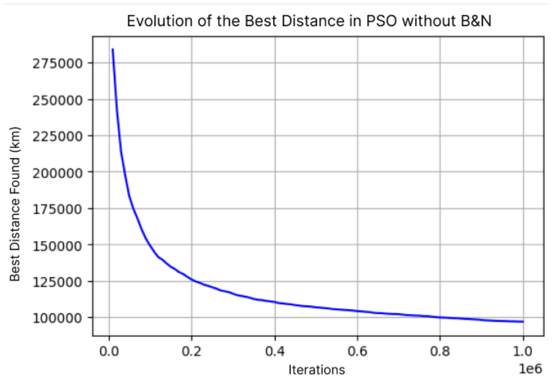

Fig. 1. Graph showing the evolution of the best result for the PSO algorithm, without using B&N, over 1,000,000 iterations.

It can be seen that, despite 1,000,000 iterations, the algorithm reached a minimum value of 97102.2 km, which is still far from the target value of 31197.4 km established in (11).

Moreover, the graph shows a sharp decline in the variation of the best result as it approaches lower values. This behavior is expected, as the closer one gets to the minimum, the slower the progress tends to be.

With a sufficiently large number of iterations, or with a specific strategy to address stagnation when variation decreases, better results could have been achieved.

The second strategy to pursue better results with the PSO algorithm was to start from particles constructed based on the B&N heuristic, although the choice of the starting city remained random (file "pso_tsp_BeN.py").

The execution parameters were the same as those of the previous execution.

The evolution of the results can be seen in Figure 2.

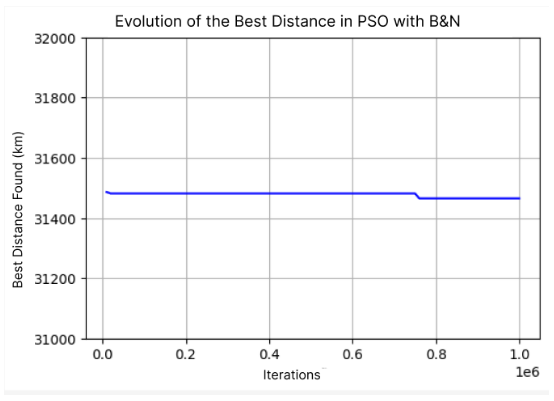

Fig. 2. Graph showing the evolution of the best result for the PSO algorithm, using B&N, over 1,000,000 iterations.

In this case, since the starting points already had much smaller values, the evolution of the results proceeded extremely slowly. The best value found was 31465.0 km, which was still greater than the value established in (11).

D. Analysis of the ACO Algorithm Results

Figure 3 shows the evolution of the best result obtained by the Ant Colony Optimization (ACO) algorithm over 2000 iterations. It can be observed that, initially, there is a sharp reduction in the total distance found, highlighting the algorithm's efficiency in quickly exploring promising solutions during the early cycles. In the first 250 iterations, ACO exhibits an intense exploration phase, during which the ants test different routes and reinforce the best edges with pheromone.

After this initial phase, an apparent stabilization in the improvement of results is noted, indicating a transition to an intensification phase. At this stage, exploration diminishes, and the algorithm focuses on refining the solutions around the identified local minimum. This stabilization is reflected in the flattening of the graph curve, suggesting a gradual reduction in improvements as the number of iterations increases.

Despite the 2000 iterations, the algorithm converged to a final value of **31.489,632 km**, slightly above the upper bound established by the Bellmore and Nemhauser heuristic, defined in (11) as **31.197,4 km**. This result demonstrates that ACO was capable of finding a high-quality solution, but stagnation in a local minimum prevented the algorithm from reaching the ideal value. This limitation can be attributed to the excessive concentration of pheromone on certain edges, which reduced the diversity of solutions during the final execution phase.

The observed behavior is characteristic of ACO, whose initial efficiency is due to the balance between exploration (guided by the heuristic η_{ij}) and intensification (pheromone reinforcement τ_{ij} on the most promising edges). However, for large-scale problems such as the TSP applied to the 853 cities of Minas Gerais, additional strategies may be necessary to avoid premature convergence. Mechanisms such as periodic pheromone reinitialization or hybridization with other evolutionary algorithms could help overcome these challenges.

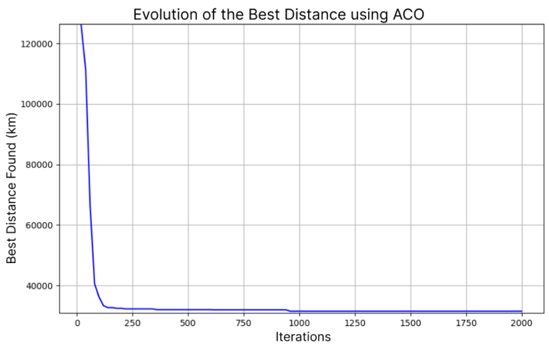

Fig. 3. Evolution of the best distance found by the ACO algorithm over 2000 iterations.

It can be seen that, despite the 2000 iterations, the algorithm reached a minimum value of **31.489,632 km**, slightly higher than the upper bound of **31.197,4 km** to be surpassed, defined in (11). This performance highlights the robustness of ACO in finding high-quality solutions for complex combinatorial problems, although it also emphasizes the need for strategies to mitigate stagnation in local minima.

E. Tabu Search Algorithm

The Tabu Search algorithm was first executed starting from a random initial point, and subsequently, the initial solution was improved using the Nearest Neighbor algorithm (file "busca_tabu.py"), with the following parameters:

- Initial Tabu List Size: $initial\_tabu\_size = 50$
- Maximum Number of Iterations: $max\_iterations = 1000$
- Maximum Tabu List Size: $max\_tabu\_size = 100$
- Local Optimum Period: $local\_optimum\_period = 200$

The evolution of the results can be seen in Figure 4.

The graph shows the evolution of the best solution cost throughout the Tabu Search iterations. Upon analyzing the results, it is possible to observe a rapid initial reduction in the solution cost, indicating that the algorithm is efficient at escaping suboptimal solutions and quickly finding a better solution. After around 200 iterations, the curve stabilizes around **26.528,34 km**,

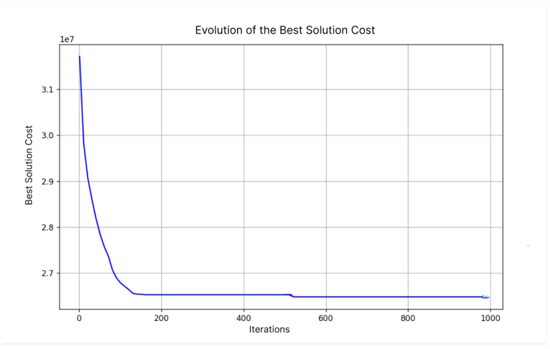

Fig. 4. Graph showing the evolution of the best result for the Tabu Search algorithm over 1000 iterations.

suggesting that Tabu Search found a relatively low-cost solution and maintained it throughout the subsequent iterations. Although the curve stabilizes, small cost variations are still observed, reflecting continued exploration of the solution space in search of even better solutions, without significant changes.

The analysis of the results reveals that the algorithm demonstrated efficiency by finding a solution with a significantly lower cost than the initial one in a short amount of time, effectively balancing the exploration of new solutions with intensification around the best solutions found. Furthermore, the stability of the cost throughout the iterations reflects the robustness of the algorithm in maintaining high-quality solutions. After 1000 iterations, the best result found was **26.482.811** meters, or **26.482,8** kilometers.

To further improve the results, it would be advantageous to incorporate additional diversification mechanisms, allowing a broader exploration of the solution space and the potential identification of even better solutions. Adjustments to the aspiration criteria could also be a useful strategy.

F. Summary of Results

The results obtained with the different algorithms implemented highlight the diversity of strategies and their respective efficiencies within the context of the Traveling Salesman Problem applied to the 853 cities of Minas Gerais.

Initially, the lower bound was established by Kruskal's algorithm for the Minimum Spanning Tree, with a value of **21,758.153 km** (10). On the other hand, the Bellmore and Nemhauser heuristic provided an upper bound of **31,197.415 km** (11), which was used as a reference to compare the evolutionary algorithms.

Among the evolutionary algorithms, Tabu Search stood out for its efficiency in escaping local optima and finding a solution with a cost of **26,482.811 km** after 1000 iterations (4). The algorithm's behavior demonstrates its robustness in exploring and intensifying solutions, balancing these two aspects well. The solution found was significantly lower than the defined upper bound, indicating strong overall performance.

The Particle Swarm Optimization (PSO) algorithm, despite adaptations and strategies, showed convergence limitations. When executed with randomly generated particles, the algorithm reached **97,102.2 km**, well above the upper bound (1). With initialization based on the B&N heuristic, the performance improved to **31,465.0 km** (2), but still above the target value. This highlights that PSO has potential but requires further adjustments for high-dimensional problems like the TSP.

Meanwhile, the Ant Colony Optimization (ACO) algorithm reached a final value of **31,489.632 km** after 2000 iterations (3). Although it demonstrated initial efficiency, the stagnation observed in later iterations points to the need for additional strategies to avoid local minima and enhance solution space exploration.

Overall, the algorithms' performances can be summarized as follows:

- Lower Bound (Minimum Spanning Tree - Kruskal): **21,758.1 km**.
- Upper Bound (Bellmore and Nemhauser Heuristic): **31,197.4 km**.
- Best result from Tabu Search: **26,482.8 km**.
- Best result from PSO: **31,465.0 km** (using B&N).
- Best result from ACO: **31,489.6 km**.

The results emphasize the efficiency of Tabu Search as the most promising strategy among the evaluated methods, obtaining a solution significantly closer to the expected optimum. On the other hand, both PSO and ACO showed limitations but also demonstrated potential for future improvements with parameter adjustments and diversification strategies.

Additionally, the use of well-defined bounds, both upper and lower, was crucial for evaluating algorithm performance and validating the solutions obtained. This reinforces the importance of combining heuristics and evolutionary algorithms to tackle complex combinatorial problems like the TSP.

IV. Conclusion

This study explored solving the Traveling Salesman Problem for the state of Minas Gerais using evolutionary and heuristic algorithms. Three main methods were implemented and evaluated: Tabu Search, Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO), along with auxiliary heuristics such as Kruskal's Algorithm and the Bellmore and Nemhauser Heuristic.

The results obtained show that Tabu Search stood out as the most efficient strategy, achieving a cost of **26,482.811 km**, significantly lower than the upper bound provided by the B&N heuristic (**31,197.415 km**). Although promising, ACO and PSO exhibited limitations related to stagnation in local minima and slow convergence, particularly for high-dimensional problems like the one addressed.

The analysis of the results highlights the importance of well-defined bounds to evaluate algorithm effectiveness and the impact of parameter choices. Despite the observed limitations, the study underscores the potential of evolutionary approaches for tackling highly complex combinatorial problems.

Finally, this work contributes to the literature by adapting evolutionary algorithms to a real and large-scale instance of the TSP, providing a solid foundation for future research and practical applications in logistics and route optimization.

Future directions include algorithm hybridization, enhanced diversification, parallelization, and exploring other metaheuristics to improve efficiency and solution quality.

ACKNOWLEDGMENTS

This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. We would like to thank the Federal University of Minas Gerais (UFMG) for institutional support and Professor Gabriela Nunes Lopes for her guidance during the Evolutionary Computation course. The knowledge acquired throughout the classes was essential for the development and deepening of the ideas presented in this work.

REFERÊNCIAS

[1] M. C. Goldbarg and H. P. L. Luna, *Otimização combinatória e programação linear: Modelos e Algoritmos*, 2nd ed. Rua Sete de Setembro, 111 – 16º andar 20050-006 – Centro – Rio de Janeiro – RJ – Brasil: Elsevier Editora Ltda, 2005.

[2] G. M. Platform, "Route api." 2025, accessed in May 2025. [Online]. Available: https://developers.google.com/maps/documentation/routes/reference/rest/v2/TopLevel/computeRoutes

[3] G. N. Lopes, "Aula pso," Apresentação de slides, Disciplina de Computação Evolucionária, Universidade Federal de Minas Gerais, 2024, accessed in May 2025.

[4] M. Aquino, M. Bravo, and O. Barros, "Repositório," 2025, accessed in May 2025. [Online]. Available: https://github.com/milbravo/compEv

[5] G. N. Lopes, "Aula aco," Apresentação de slides, Disciplina de Computação Evolucionária, Universidade Federal de Minas Gerais, 2024, accessed in May 2025.

[6] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 26, no. 1, pp. 29–41, 1996.

[7] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[8] A. Gomes, "Uma introdução à busca tabu," Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, SP, Brasil, 2009, accessed in May 2025. [Online]. Available: https://www.ime.usp.br/~gold/cursos/2009/mac5758/AndreBuscaTabu.pdf