# Self-Organizing Maps and Optimality Gaps for Solving the Traveling Salesman Problem

Harsh Sharma, Shlok Kamat, Vijit Daroch

November 18, 2024

**Abstract**

Self-Organizing Maps (SOMs), also known as Kohonen maps, are a computational technique that projects high-dimensional data onto a grid, preserving geometric relationships in a low-dimensional representation. This study explores the application of SOMs to the Traveling Salesman Problem (TSP), leveraging their ability to abstract data for generating sub-optimal solutions. By introducing slight modifications to the SOM grid, feasible TSP routes are constructed efficiently. Additionally, the sub-optimal solutions obtained through SOMs are refined using an optimality gap-based approach, where the Gurobi solver is employed to compute the optimal city sequence and minimize the total travel distance within certain constraints. This paper provides a comprehensive overview of the TSP and SOM methodologies, evaluates the performance of the Python implementation, and demonstrates the integration of heuristic and exact optimization techniques to enhance solution quality.

## 1 Introduction

The Traveling Salesman Problem (TSP) is a well-known NP-complete problem in combinatorial optimization, aiming to find the shortest possible route that visits a set of cities exactly once and returns to the starting point. Due to its complexity, heuristic approaches such as Self-Organizing Maps (SOMs) have gained attention for generating sub-optimal solutions. SOMs, inspired by neural networks, abstract high-dimensional data into a low-dimensional grid, making them effective for tasks like pattern recognition and data organization. When adapted for TSP, SOMs use a modified algorithm to transform the problem into an elastic ring of nodes, dynamically adjusting the nodes to minimize the overall route length.

This paper extends the SOM methodology by incorporating its results as a stopping criterion for route optimization using the Gurobi solver. After generating an initial solution with SOM, the obtained route is used to set a threshold, allowing Gurobi to focus on finding an improved route within the defined limits. This integration leverages SOM to guide the optimization process, enhancing efficiency while ensuring a practical solution to the TSP. The paper provides a detailed evaluation of this approach, showcasing its ability to effectively combine heuristic and exact methods for solving complex problems.

### 1.1 Summary from original paper

In his seminal 1990 paper [1], Teuvo Kohonen provided a concise and insightful explanation of the *Self-Organizing Map* (SOM) technique. SOMs are conceptualized as grids of nodes, typically two-dimensional, inspired by the structure and functionality of neural networks. Central to the concept is the *model*, which represents the real-world data the map aims to abstract. The primary goal of SOMs is to reduce the dimensionality of the data while preserving the underlying relationships and patterns of similarity among its components.

To achieve this, nodes within the SOM are spatially arranged such that those with higher similarity are positioned closer together. This property makes SOMs an effective tool for visualizing and organizing patterns in complex datasets. The formation of this structure involves an iterative regression-like process where the positions of nodes are updated sequentially based on elements from the dataset, thereby aligning the map to reflect the input data's inherent structure. The mathematical formulation for this process governs the adaptation of nodes to improve the representation's accuracy. The expression used for the regression is given by:

$$n_{t+1} = n_t + h(w_e) \cdot \Delta(e, n_t)$$

This implies that the position of the node $n$ is updated by adding the distance between the node and the given element, weighted by the neighborhood factor of the winning neuron $w_e$. The winning neuron for an element is the node in the map most similar to it, typically identified as the closest node based on Euclidean distance, although alternative similarity measures can be used when appropriate.

Conversely, the *neighborhood* is defined as a convolution-like kernel applied to the map around the winning neuron. This approach enables updating not only the winner but also its neighboring neurons, bringing them closer to the given element in a smooth and proportional manner. The neighborhood function is commonly modeled as a Gaussian distribution, though alternative implementations exist.

## 2 Using Self-Organizing Maps to Solve the Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a well-known NP-complete problem in combinatorial optimization. It involves determining the shortest route that visits a given set of cities exactly once and optionally returns to the starting city. The objective is either to minimize the total distance or, in cases with associated costs, to minimize the total cost of the route [2].

In this study, we propose a method to address the TSP by adapting Self-Organizing Maps (SOMs) to serve as a heuristic tool for generating solutions. The approach incorporates modifications to the standard SOM methodology, drawing inspiration from prior studies on self-organizing feature maps and parameterization techniques for optimization.

### 2.1 Algorithm Modifications

The core concept of adapting SOMs to solve the TSP lies in modifying the neighborhood function. Instead of using a grid structure, a circular array of neurons is employed. In this setup, each neuron is aware only of its immediate neighbors in the sequence, both ahead and behind. This modification transforms the SOM into an *elastic ring* that adapts to the spatial configuration of cities while attempting to minimize its overall perimeter.

Although this transformation is fundamental to the technique, it does not ensure convergence by itself. To achieve convergence, two mechanisms are introduced:

1. **Learning Rate** ($\alpha$):
    - A dynamic learning rate is applied to balance exploration and exploitation during the optimization process.
    - High exploration is emphasized at the start of the process, transitioning to greater exploitation as the algorithm progresses.

2. **Decay Functions**:
    - Both the learning rate and the neighborhood function decay over time.
    - Decaying the learning rate reduces the magnitude of neuron displacement, while decaying the neighborhood radius moderates the influence on nearby neurons, enabling local refinement.

The regression-based update for neuron positions is expressed as:

$$n_{t+1} = n_t + \alpha_t \cdot g(w_e, h_t) \cdot \Delta(e, n_t)$$

where:

- $\alpha_t$ is the learning rate at iteration $t$,
- $g(w_e, h_t)$ is a Gaussian function centered on the winner neuron $w_e$, with dispersion $h_t$,
- $\Delta(e, n_t)$ represents the adjustment term for the current node position.

The decay functions for the learning rate and neighborhood dispersion are defined as:

$$\alpha_{t+1} = \gamma_\alpha \cdot \alpha_t, \quad h_{t+1} = \gamma_h \cdot h_t$$

Here, $\gamma_\alpha$ and $\gamma_h$ are decay factors controlling the rate of reduction.

This approach parallels methods used in reinforcement learning, such as Q-Learning, where parameter decay aids in balancing global exploration and local exploitation. The decaying parameters enable the SOM to refine its solution progressively, ultimately converging on a plausible route.

## 2.2 Extracting the Route

To derive the final route from the SOM, each city is associated with its *winning neuron*. Starting from any point on the circular array, the cities are ordered based on the sequence of their winning neurons along the ring.

In instances where multiple cities map to the same neuron, the SOM does not specify a traversal order. This ambiguity arises due to either insufficient precision or the irrelevance of the exact order for minimizing the route distance. In such cases, any feasible ordering of the cities can be used without significantly affecting the solution quality [3].

The modifications and techniques introduced in this method enable SOMs to provide a structured and efficient approach to approximating solutions for the TSP [4]. This approach serves as a useful heuristic for tackling the complexity of this combinatorial problem, especially for scenarios requiring quick, sub-optimal solutions.

## 2.3 Implementation

The proposed technique was implemented in Python 3 to efficiently process instances of the Traveling Salesman Problem (TSP). The implementation is designed to parse and load any two-dimensional instance modeled in the `TSPLIB` format, a standard representation for TSP problems. This format was chosen to facilitate the evaluation of the algorithm using benchmark instances provided by the National Traveling Salesman Problem Library hosted by the University of Waterloo.

The implementation is modular, with key functionalities distributed across multiple Python files located in the `src` directory:

- `distance.py`: Contains functions for calculating two-dimensional Euclidean distances, as well as utilities for evaluating and processing distances.
- `io_helper.py`: Provides functionality to parse `.tsp` files and load them into runtime objects compatible with the algorithm.
- `main.py`: Manages the overall execution flow of the self-organizing map, including initialization and iterative updates.
- `neuron.py`: Handles network generation, computation of neighborhoods, and route determination based on the SOM methodology.
- `plot.py`: Includes functions to generate graphical representations of intermediate and final outputs, providing visual insights into the algorithm's behavior and results.

This modular structure ensures a clear separation of concerns, making the algorithm both scalable and maintainable. The implementation leverages Python libraries such as `numpy` for efficient numerical computations and `matplotlib` for visualizing the progression and results of the SOM-based optimization process. Together, these tools enable seamless execution and evaluation of the algorithm for solving complex TSP instances.

## 3 Evaluation

To assess the effectiveness of the proposed implementation, a series of benchmark instances from the *National Traveling Salesman Problem Library* were utilized. These instances are derived from real-world geographic datasets and include the optimal routes for most scenarios, providing a robust foundation for evaluation. The evaluation strategy involved executing the algorithm across multiple problem instances and analyzing the results based on the following metrics:

- **Execution Time**: The total computational time required by the algorithm to converge to a solution.
- **Solution Quality**: Measured as the ratio of the obtained route length to the optimal route length. For instance, a route described as *10% longer than the optimal route* corresponds to a solution that is 1.1 times the length of the optimal solution.

### 3.1 Parameters for Evaluation

The parameters for the SOM-based algorithm were fine-tuned using prior studies as a reference and further refined through experimentation. The baseline parameters used in the evaluation are as follows:

- **Population Size**: 8 times the number of cities in the problem instance.
- **Initial Learning Rate** ($\alpha_0$): 0.8, with a decay factor of 0.99997 per iteration.
- **Initial Neighborhood Radius**: Equal to the number of cities, with a decay factor of 0.9997 per iteration.

These parameters were applied uniformly across all instances to maintain consistency in the evaluation process. However, finer-grained parameter tuning could potentially yield better results for individual instances.

## 3.2 Problem Instances

The evaluation was conducted on the following problem instances:

- **Western Sahara**: 29 cities, with an optimal tour length of 27603.
- **Qatar**: 194 cities, with an optimal tour length of 9352.
- **Uruguay**: 734 cities, with an optimal tour length of 79114.
- **Finland**: 10639 cities, with an optimal tour length of 520527.

## 3.3 Stopping Criteria

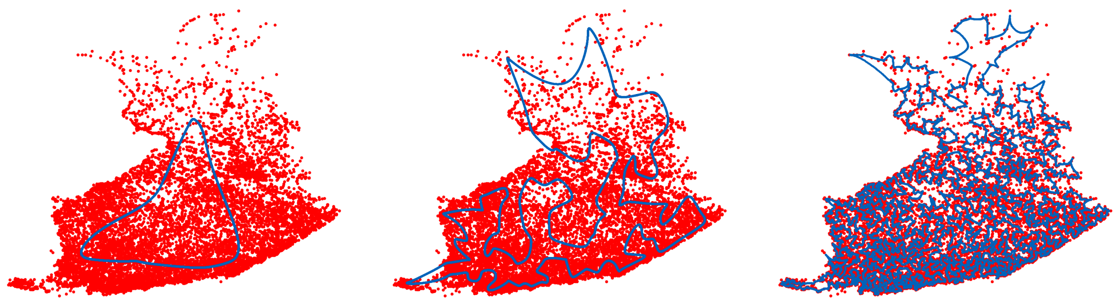
The implementation terminates execution when key parameters, such as the learning rate or neighborhood radius, decay below a predefined threshold. This ensures computational efficiency while maintaining solution quality. Although a uniform set of parameters was tested, future work could explore more instance-specific parameterization to optimize performance.

The evaluation results, averaged over five independent runs for each instance, are summarized in Table 3.3. These results highlight the trade-offs between execution time and solution quality, demonstrating the effectiveness of the proposed approach across varying problem sizes and complexities.

| **Instance** | **Iterations** | **Time (s)** | **Length** | **Quality** |
|---|---|---|---|---|
| Western Sahara | 18153 | 7.4 | 27603 | 0% |
| Qatar | 24487 | 10.8 | 9800 | 4.6% |
| Uruguay | 28922 | 21.0 | 85499.2 | 7.5% |
| Finland | 37833 | 128.4 | 638111.4 | 18.4% |

Table 1: Evaluation Results for TSP Instances

The relationship between execution time and the number of cities is evident, as the complexity of identifying the closest neuron for a city increases with the size of the population being traversed. While better results could be achieved for larger datasets, time constraints often necessitate early termination of the process. Notably, the solutions obtained are consistently within 20% of the optimal route length. In smaller instances, such as Uruguay, higher solution quality is observed despite the problem's inherent complexity, likely influenced by the country's unique topographical features.



Iterations = 1000 Iterations = 10000 Iterations = 28000

Figure 1: Three different steps of execution of the Finland instance.

In the graphical representation of the execution process, the algorithm's dynamic behavior becomes evident. Initially, it resembles an elastic, rapidly adapting ring that strives to conform to its shortest possible configuration. This phase reflects the exploratory nature of the technique, where the solution space is broadly searched for potential improvements. As the execution progresses, the system begins to stabilize, focusing on refining its internal configuration to identify more optimal local pathways.

To facilitate this dual-phase approach, a significant number of neurons are introduced into the network. During the initial phase, this abundance of neurons allows for extensive exploration, enabling the algorithm to navigate the solution space effectively. As the process advances, the algorithm transitions into an exploitation phase by systematically decaying the parameters. This strategic adjustment ensures a shift from global exploration to local optimization, enabling the network to fine-tune its solution and converge on a more refined and efficient result.

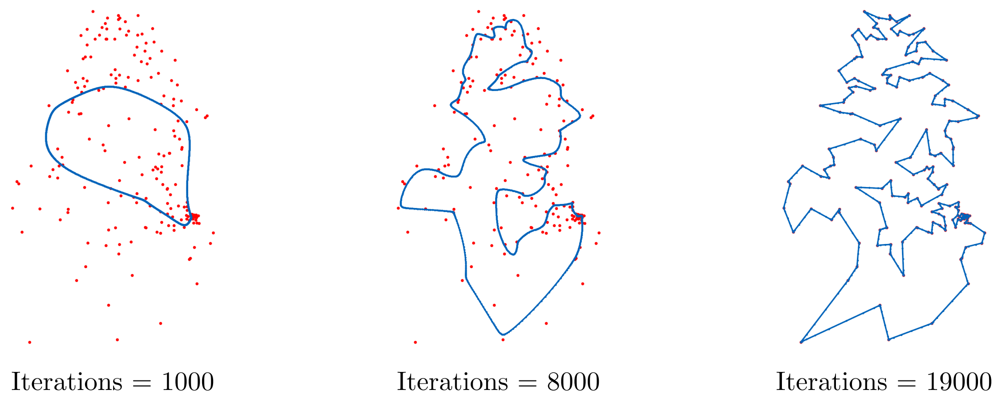


Figure 2: Three different steps of execution of the Qatar instance.

This two-fold methodology of exploration followed by exploitation highlights the robustness of the technique in balancing global and local search capabilities, ensuring a comprehensive and optimized approach to problem-solving.

The results are overall acceptable, except in extremely demanding settings for the TSP. We can see how in less than 15 seconds, a suboptimal route crossing for nearly 200 cities was found.

## 4 Optimality Gap in TSP and Stopping Criterion

The *optimality gap* in Traveling Salesman Problem (TSP) solutions represents the difference between the best-known solution (current route length) and the theoretical optimal solution, expressed as a percentage of the latter. In our approach, we leverage the Self-Organizing Map (SOM) algorithm to iteratively improve the route. The optimality gap serves as a dynamic stopping criterion; when the gap falls below a predefined threshold, we halt further iterations, deeming the route sufficiently close to optimal.

This method offers notable benefits compared to directly using a solver to find the exact optimal route. First, it significantly reduces computational overhead, especially for large-scale problems, by avoiding the exhaustive search processes typical of exact solvers. Second, it provides a flexible trade-off between solution quality and runtime, allowing for faster convergence when exact optimality is not strictly required. Finally, this approach incorporates exploration and learning into the optimization process, which can be particularly valuable for complex and dynamic variations of the TSP where solvers may struggle with scalability or adaptability.

# 5 Implementation of TSP Solution Using Gurobi and Optimality Gap

This implementation utilizes the Gurobi optimizer to solve the Traveling Salesman Problem (TSP), focusing on computational efficiency and practical performance. The algorithm integrates the concept of an *optimality gap* to achieve a controlled trade-off between solution quality and runtime, making it particularly effective for large-scale TSP instances.

## 5.1 Input and Distance Matrix Calculation

The process begins by reading coordinates from a `.tsp` file using the `read_coordinates` function. Each line of the file is assumed to contain a node identifier followed by latitude and longitude values. These coordinates are stored as tuples of floats. Using these tuples, the `calculate_distance_matrix` function computes the pairwise Euclidean distances between all cities. The resulting symmetric distance matrix serves as the foundation for constructing the optimization model.

## 5.2 Mathematical Formulation

The TSP is formulated as an integer linear program (ILP) with the following elements:

1. **Variables:** Binary decision variables $x_{ij}$ are introduced to indicate whether a direct route between city $i$ and city $j$ is part of the solution. Additionally, continuous variables $u_i$ are included to implement subtour elimination using the Miller-Tucker-Zemlin (MTZ) formulation.

2. **Objective Function:** The objective is to minimize the total travel distance, expressed as:

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} \cdot x_{ij}$$

   where $d_{ij}$ represents the distance between cities $i$ and $j$.

3. **Constraints:**
   - Each city must be departed from and arrived at exactly once.
   - Subtour elimination is enforced using the MTZ constraints:

$$u_i - u_j + n \cdot x_{ij} \leq n - 1 \quad \forall i, j \quad (i \neq j)$$

## 5.3 Optimization and Optimality Gap

The Gurobi model is solved with a user-defined *MIP gap*, which specifies the acceptable relative difference between the current solution and the theoretical optimum. By setting an optimality gap (e.g., 10%), the solver terminates when the best-found solution is within the defined threshold of the optimal solution. This approach provides a balance between computational efficiency and solution quality.

## 5.4 Results and Route Reconstruction

Once the optimization completes, the solution is interpreted to reconstruct the tour. Starting from an arbitrary city, the algorithm traces through the selected routes ($x_{ij}$) to form a complete loop. The optimal or near-optimal route and its total distance are presented as the final output.

## 5.5 Evaluation of the Method for Qatar and Western Sahara

To evaluate the effectiveness of the proposed method, we tested it on two datasets representing cities in Qatar and Western Sahara. For Qatar, the dataset consisted of 194 cities, while the Western Sahara dataset included 29 cities. The algorithm successfully identified efficient routes for both cases.

For Qatar, the optimality gap was set to 10%, and the method produced a route with a total distance of approximately 9351.1 units (scaled Euclidean distance). The resulting route demonstrated efficient traversal through densely packed urban areas, minimizing redundant travel between nearby cities. In contrast, for Western Sahara, characterized by sparsely distributed cities, the algorithm found a route totaling approximately 2760.4 units. This result highlighted the algorithm's adaptability to handle both densely and sparsely populated datasets, providing high-quality solutions in both scenarios.

For a dynamic visualization of the algorithm's convergence, see the Qatar Route GIF and Western Sahara Route GIF.

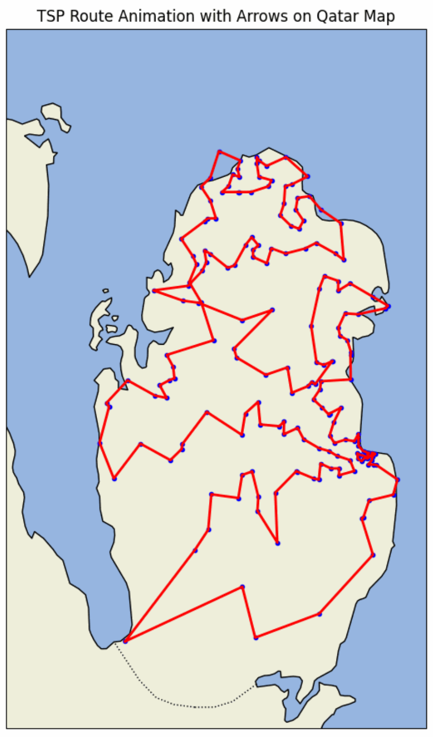


Figure 3: Optimal route identified for Qatar (194 cities).

## 5.6 Benefits of Using the Optimality Gap

Introducing the optimality gap offers several advantages:

- **Reduced Runtime:** By allowing near-optimal solutions, the solver can terminate earlier, saving significant computational resources.
- **Scalability:** The approach is well-suited for larger TSP instances, where finding exact solutions may be computationally prohibitive.

- **Flexibility:** Users can adjust the MIP gap to balance runtime and solution precision based on the application's requirements.

## 6 Conclusion

This study introduced a novel approach to solving the Traveling Salesman Problem (TSP) by combining exact optimization techniques with a heuristic-based stopping criterion, leveraging the concept of an optimality gap derived from the Self-Organizing Map (SOM) methodology. The framework halts optimization when the optimality gap falls below a predefined threshold, effectively balancing computational efficiency and solution quality. This makes it particularly suitable for large-scale, computationally intensive TSP instances where exact solutions are impractical.

The use of the optimality gap as a stopping criterion accelerates convergence, offers flexibility in adjusting solution quality, and adapts well to diverse datasets, as demonstrated with cities in Qatar and Western Sahara. By bridging heuristic and exact optimization techniques, the method highlights their complementary strengths in solving complex combinatorial problems efficiently.

Despite its strengths, the proposed methodology has certain limitations. The use of an optimality gap inherently prioritizes computational efficiency over exact solution guarantees, which may not be suitable for applications requiring strict optimality. Additionally, the reliance on Euclidean distance as the travel cost metric assumes homogeneity in terrain and travel conditions, an assumption that may not align with real-world scenarios. Moreover, the computational burden associated with the Miller-Tucker-Zemlin (MTZ) subtour elimination constraints highlights the need for exploring more efficient alternative formulations.

Future research could focus on addressing these limitations by incorporating non-Euclidean distance metrics, such as travel times or monetary costs, to better reflect practical scenarios. Machine learning techniques could be explored to dynamically adjust the optimality gap during runtime, enhancing the model's adaptability. Additionally, adopting advanced subtour elimination techniques, such as cut-based approaches, or leveraging parallel processing capabilities in optimization solvers could improve computational performance.

In conclusion, this work contributes to the literature by demonstrating the effectiveness of combining heuristic-based stopping criteria with exact optimization techniques in solving the TSP. While certain trade-offs exist, the proposed approach lays the groundwork for further research into hybrid methodologies that prioritize scalability and precision, with promising applications in logistics, transportation, and related fields.

## References

[1] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[2] K. L. Hoffman, M. Padberg, G. Rinaldi, *et al.*, "Traveling salesman problem," *Encyclopedia of operations research and management science*, vol. 1, pp. 1573–1578, 2013.

[3] L. Brocki, "Kohonen self-organizing map for the traveling salesperson," *Traveling Salesperson Problem, Recent Advances in Mechatronics*, pp. 116–119, 2010.

[4] B. Angeniol, G. D. L. C. Vaubois, and J.-Y. Le Texier, "Self-organizing feature maps and the travelling salesman problem," *Neural Networks*, vol. 1, no. 4, pp. 289–293, 1988.