Erasmus School of Economics

ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Master's Thesis

Econometrics & Management Science

with specialisation

Analytics and Operations Research in Logistics

A Comparative Analysis of artificial intelligence approaches on the Travelling Salesman Problem

---

Author:

Fotios Sardelis

506833

Supervisor:

Dr. (Hakan) MH Akyuz

Second assessor:

Dr. (Remy) R Spliet

July 2022

ERASMUS UNIVERSITEIT ROTTERDAM
ERASMUS SCHOOL OF ECONOMICS

Abstract

With the advent of neural networks in the previous decade, new and effective methods for addressing combinatorial optimization problems such as the Traveling Salesman Problem were devised. Despite the fact that neural networks transcend metaheuristics and evolutionary algorithms, it seems that these approaches have reached their limits, and their efficacy has been questioned.

Few comprehensive studies look into the use of neural networks to solve real-world optimization problems in industry. As a result, the purpose of this research is to address a gap in the literature by comparing neural network approaches to competitive methodologies.

A Kohonen Neural Network will be investigated, built, and compared to the K-means clustering approach, a machine learning tool for TSP and to an Elitist Ant System (EAS), a powerful metaheuristic. Performance metrics, solution quality, running times, and efficiency will all be determined through computational tests. The aforementioned approaches will be applied to various sorts of TSP instances in terms of size, format, and origin.

More practical applications of neural networks must undoubtedly be tackled in order to illustrate their potentiality. However, when compared to other strategies, the results show that the Kohonen Neural Network is a promising mechanism for solving the TSP. Due to its settings flexibility and performance in terms of the solution values provided, it is arguably the most efficient strategy.

Contents

1. Introduction

1.1 Background knowledge

The Traveling Salesman Problem is one of the most extensively studied combinatorial optimization problems in the literature of operation research. The TSP is associated with determining the shortest route between a set of cities. Due to the vast number of viable answers, this deceptively easy problem is difficult to solve. As a result, strategies that produce a decent sub-optimal solution in an acceptable amount of time are commonly utilized (Bert La Maire and Valeri M. Mladenov, 2012).

The rise of artificial neural networks in the past twenty years, offered new and efficient methodologies to solve complex combinatorial optimization problems such as the TSP. Excitement was unpredictable, as new ways were devised and their limits were discovered a few years later (Kate Smith-Miles, 1999). The first appearance of a neural network that solved the Traveling Salesman Problem comes back in 1985 with Hopfield & Tank. According to many researchers, Hopfield & Tank neural network was considered controversial and unreliable due to problematic parameter coordination and infeasible solutions. These doubts concerning the validity of the Hopfield & Tank approach, were published by Wilson and Pawley and seemed to refute the enthusiasm around it (G. V. Wilson and G. S. Pawley, 1988). A more promising approach, that will be benefited from this thesis study, comes from Kohonen's Self-Organizing Feature Map (T.Kohonen, 1982). The main advantage of this approach is that self-organizing or Kohonen neural networks have successfully solved not only the Traveling Salesman problem but also two - dimensional problems such as the vehicle routing problem (I. Vakhutinsky, B.L. Golden, 1994) and the shortest path problem (M. Takahashi, K. Kyuma, E. Funada, 1993), which can be considered variants of the TSP.

In this study the application of Kohonen's Neural Network on the Traveling Salesman Problem (Brocki and Koržinek, 2007) will be the benchmark for comparison. In terms of solution quality, the neural method is seldom compared with the best performing or most competitive alternative strategy in operations research literature. Usually, only comparisons between neural networks for TSP can be spotted (Bert La Maire and Valeri M. Mladenov, 2012) or between exact and machine learning techniques (Nassirou Lo Jeremiah Ishaya and Abdullahi Ibrahim, 2019). Thus, there is a need for such a comparative analysis, as proposed below, since to the best of our knowledge there is no similar comparison between a neural network with existing alternative methods for TSP.

According to J.N.Hooker (1995) the key direction for this computational experiment is to include heuristics, since neural networks are extremely competitive with metaheuristics.

Visiting the wide family of metaheuristics, the Ant Colony optimization based techniques seem promising and competitive. The first Ant Colony Optimization algorithm, the well known Ant System was introduced using the Traveling Salesman Problem as an example application (M.Dorigo, 1992). The Ant System provided

promising initial results but was considered to be less efficient and performant than the latest algorithms for TSP at that time (M.Dorigo, 1992). The Ant System is also notable because it inspired a number of improvements that contributed greatly to the performance of ACO techniques. One of these enhancements is the Elitist Ant System, which provides significant reinforcement to the arcs of the best route identified. (S.Parsons, 2005).

Combinatorial optimization problems, such as the TSP, have become an active field of application for machine learning. Thus, another motivation of this thesis study, is the comparison of the Kohonen Neural Network with a classic machine learning method. A widely used machine learning method belongs to the group of clustering analysis and more specifically is considered to be the classical K-means clustering algorithm. The reason that the K-means clustering algorithm is selected as a candidate for comparison is that it unexpectedly depicts more similarities than differences with a Kohonen Neural Network, due to their machine learning nature. Following the research already conducted (Bação Fernando, Victor Lobo, and M. Painho, 2005), it can be easily derived that clustering algorithms were highly affected by the amount of overlapping samples from different classes and outliers, while Kohonen Neural Network did not perform well, being enormously affected by the number of variables. Moreover, it is worth mentioning that in simulations where many different clusters structures were involved combined with large data sets, K-means methods had a really good performance compared to the performance of Kohonen Neural Network (Sueli A. Mingoti, Joab O. Lima, 2005). Additionally, there is not a known comparison between a K-means clustering algorithm with a Kohonen Neural Network, specifically for the TSP. Thus, it is interesting to explore the dependency of the TSP performance on clusters and the behavior of the Kohonen Neural Network in comparison with a K - means clustering algorithm on the data sets selected for this study.

1.2 Research Questions

The objective of this thesis study is to provide the reader with a comparative analysis of two machine learning methods and one metaheuristic that aim to solve the Traveling Salesman Problem. Apart from implementing and solving the Traveling Salesman Problem using the aforementioned methods, the purpose of this thesis study adds value to the operations research by exploring diverse and interesting fields for comparison. This comparative analysis will offer computational experiments, but also insights concerning the solution quality and the running times. In other words, it will explore the behavior of these methods in different data sets, in terms of size and form, that will be described in the Data section 5.

It is certain that more and more practical applications need to be solved using Neural Networks in order to demonstrate their capabilities. Until today, most of the research has been focused on the solution to the Traveling Salesman Problem and to a wide range of classic combinatorial problem using Neural Network but not to the evaluation of the Neural Networks' potential.

A more detailed description of the proposed methods will follow in the Methodology section 4.

Therefore, this thesis aims to answer the following main research question:
How do machine learning methods for the Traveling Salesman Problem perform in comparison to the well known metaheuristic ?

In other words, what kind of performance insights can be derived comparing a Kohonen's Neural Network on TSP instances to the metaheuristic of the Elitist Ant System and to the machine learning algorithm of K - means algorithm.

The following two sub-questions will be addressed to answer the main research question :

- How the performance of the proposed methods is measured?

  The answer of this sub-question refers to metrics and criteria on how the performance is measured, in terms of solution quality, accuracy, efficiency and running time on small and large data sets. Moreover, to the performance trade-off between the three proposed methods and how they can be improved is to be explored as well. Another aspect that will be extensively investigated is how an initialization technique affects the overall performance and how the selection of crucial parameters can result in better outcomes and faster convergence.

- Which is the most efficient method and most preferred approach depending on the characteristics of the instances ?

  Elaborating on this sub-question, it is crucial to spot the kind of insights can be derived from the computational experiments and the extent the traditional computationally bulky method of the metaheuristic, approaches the near - optimal solution of the machine learning methods.

By answering the research questions and conducting a thorough analysis, this thesis study aims to validate that the Kohonen Neural Network can cope with the complexity of the combinatorial problems. Hence, it will indicate that the Neural Network is one of the most effective approaches to solve TSP compared to other techniques, as expected.

2. Literature Review

In this section the literature review will be discussed in terms of background knowledge, methodology and results that have been obtained in previous research on the TSP or other combinatorial problems. Moreover, the relevance of the existing literature to the current research will be presented in the three subsections below. In section 2.1 Kohonen Neural Network approaches for the TSP are presented, while in section 2.2 the metaheuristic approach of Elitist Ant System together with its extensions is given. The last part in section 2.3 contains the K - means clustering based approaches.

2.1 Kohonen's Neural Network

As it is already mentioned, the first appearance of the Kohonen Neural Network and the introduction of the Self - Organizing Map (SOM) feature took place in 1982 by T.Kohonen. However, these early models and neural networks seem to be rather weak. Kohonen's breakthrough was the introduction of a system that comprises of two distinct subsystems. The neural network implements the winner function, whereas the other subsystem adjusts the local behavior of the neurons during the learning process. The first application area of the SOM was speech recognition, which marks the time when the SOM has entered the world of data analysis and data exploration (Kaski S. Oja, M. and T. Kohonen, 2003).

Taking a look at extensions and improvements, a large amount of variants of the basic SOM has been suggested. The main alternatives for the definition of the SOM models are the Generative Topographic Map (C.M. Bishop, M.Svensén and C.K.I. Williams, 1998) and the information-based computation of the SOM models (Van Hulle, 2000). Moreover, improvements on the adaptive structure of the SOM array have been introduced by Fritzke (1994), and alternative definition of the neighborhood relations are defined by Martinetz (1993). Latest publications concerning the Self-Organizing Map are included in Laaksonen and Honkela (2011) and Príncipe and Miikkulainen (2009).

The SOM algorithm, originally introduced by Kohonen, is an unsupervised learning algorithm, that creates a topological link between the inputs. This is the main characteristic that will explored by this thesis study and has been explored by many others found in the literature that approached the Traveling Salesman Problem via the SOM algorithm. Some of those key references that have been reviewed in the literature are coming almost twenty years back. One of these was provided by Aras N, Oommen BJ, and Altinel IK (1999), who incorporated statistics in the Kohonen Network setting it the most accurate method for the TSP among the neural solutions available in the literature. Another interesting approach comes from Somhom S, Modares A, Enkawa T. (1999) where a competition based neural network was established for the multiple traveling salesman problem using a mini-max objective function. J.C.Fort (1988) also provided an interesting application,containing many numerical examples and giving good sub-optimal tours. What

was special in this study, is that only Kohonen's algorithm was applied without an energy function nor any parameter selection criteria chosen.

2.2 Elitist Ant System

The Ant Colony Optimization, the second component of this comparative study, is a metaheuristic in which artificial ants collaborate to discover good answers to complex optimization problems. Ant Colony Optimization techniques are used to address both static and dynamic combinatorial optimization problems, such as the Traveling Salesman problem.

The Ant System, created by Marco Dorigo, was the first algorithm in the Ant Colony optimization framework (1992). A large number of diverse algorithms followed this system and tried to improve it in terms of performance. As it is already mentioned in the introduction chapter, the Ant System provided tangible initial results and was an inspiration for a large amount of extensions that improved its performance significantly. These extensions consider among others the Elitist Ant System, the rank - based Ant System and the Max - Min Ant System. The main difference between those, is how the pheromone levels are updated.

A thourough literature research yields that the Elitist Ant System was introduced by Dorigo, Maniezzo & Colorni in 1996, while the following extensions of Max - Min Ant System by Stutzle & Hoos in 1999 and the Rank-based Ant System by Bullnheimer, Hartl & Strauss in 1997. Relevant extensions of the Ant System can be found in the approximate non-deterministic tree search algorithm introduced by Maniezzo in 1999 and the Hyper-cube Ant System by Blum, Roli & Dorigo in 2001.

The adaptive elitist-ant system for addressing combinatorial optimization problems was proposed by Abuhamdah (2021), which is the most recent appearance of the Elitist Ant System enhancement. Karmakar, Mitra, Dey, Chakraborty, and Nayak, on the other side, provided an enhanced Elitist Ant System based on pheromone approach and dynamic candidate lists (2016). The common ground of these extensions and improvements is that they follow the narrative of Dorigo (1992) as base, so is worth exploring the details of the initial Ant System.

Dorigo presented three new Ant System variants after expanding on the original Ant System (Dorigo, 1992, Colorni, Dorigo, & Maniezzo,1996). Ant-density, ant-quantity, and ant-cycle, to name a few. The pheromone level was updated after all ants had built the tours in the ant-density and ant-quantity versions, whereas the pheromone level was updated after all ants had built the tours in the ant-cycle version. The amount of pheromone deposited by each ant was set to be the function of tour quality. When the Ant System is referred, the ant-cycle is considered since the two other variants were of inferior performance. Additionally, the performance of the Ant System when compared to other metaheuristics decreases dramatically as the size of the test-instance increases.

Therefore, computational results presented in Dorigo (1992) and Colorni, Dorigo, & Maniezzo (1996) suggest that the use of the elitist strategy or the Elitist Ant System with appropriate parameter selection values,

allows the Ant System to find better tours with a lower number of iterations. That is the reason, that the algorithm of this research will be followed and implemented in the suggested comparative analysis.

2.3 K - means clustering algorithm

Moving to the third component of this comparison, a K-means clustering algorithm is explored. The literature review on this approach is wide and vast.

According to J. MacQueen(1967) K-means is one of the most popular algorithms for non-hierarchical clustering problems. The algorithm's goal is to divide data items into k clusters. The data items in the k-means algorithm are grouped based on their distance from one another, with each data object belonging to just one cluster. The K-means method seeks out centroids with the shortest distance between them and the data items that belong to them.

Applications of this kind of machine learning algorithms to the Traveling Salesman Problem are numerous but it is important to mention the "most relevant" to this thesis study. One of them is presented by Karakoyun M.(2019), who uses a K-means clustering method combined with a Shuffled Frog Leaping Algorithm (SFLA). This approach consists of three parts: separation of the cities into k clusters, finding the shortest path for each cluster and merging the clusters. The results have shown that this algorithm gets better results as the number of clusters increases for problems that have a large number of cities.

Additionally, it can be easily observed that K-means clustering algorithm is usually used as a pre-processing step in order to solve the Traveling Salesman Problem. R.Nallusamy, K.Duraiswamy, R.Dhanalaksmi & P. Parthiban (2009) used an approach of clustering the number of given cities depending upon the number of salesmen and each cluster is allotted to a salesman. In this approach, multiple -TSP has been changed to TSP, which is easier to calculate than m-TSP. After clustering, each salesperson in his allocated cluster is assigned an optimal route. Majd Latah (2016) uses a similar method to propose a solution for the multiple TSP Problem using the K-Means and Crossover based Modified ACO Algorithm. Also, in this case K-means clustering algorithm is used in the pre-process since cities are grouped in clusters where each cluster represent a set of adjacent cities.

The most relevant research conducted in the literature close to this suggested study is presented by Ameera J.,Bara'a M. & Waed A. (2019) by utilizing a Firefly Algorithm (FA) and a k-means clustering algorithm. The three major steps include: clustering the nodes, finding the optimal path in each cluster, and reconnecting the clusters. The first stage divides the nodes into sub-problems using k-means clustering, while the second step uses FA to discover the best path in each cluster and eventually connects all clusters.

Generally, the most common approach of solution for the TSP is in the 2-opt optimization. 2-opt, a local search algorithm, was first proposed by Croes in 1958. The main idea behind it, is to take a route that crosses over itself and reorder it by comparing every possible combination of this mechanism. This technique checks if a swap of two edges yields a better result (removing a crossing) and can be applied to the travelling

salesman problem as well as many related problems. The reason why this is mentioned is totally relevant since it will be studied in the suggested thesis. Instead of correcting crossings, it will be more efficient to create as few as possible costly ones by grouping the cities. A modified version of the K-Means algorithm that clusters nearby cities, groups them, then groups the groups and corrects the sequence of visiting after which it connects all groups resembles with the aforementioned K-means clustering algorithms review.

To sum up, the comparative analyses in the literature are scarce and not applied specifically on the Traveling Salesman problem but it is worth mentioning that Kohonen's Neural Network have been compared with a K-means clustering algorithm on real life data with known cluster solutions. The performance of these algorithms is dependent to changes in the number of clusters and number of observations (Usha A. Kumar and Yuvnish Dhamija, 2010), while Sueli A. Mingoti, Joab O. Lima, (2005) have indicated that K-means clustering methods had a really good performance compared to a (SOM) Kohonen Neural Network. On the other hand, comparisons of a (SOM) Kohonen Neural Network with an Ant Colony Optimization metaheuristic do not exist.

3. Theoretical Background

The goal of the Traveling Salesman problem is to determine the shortest route that a salesman should take to travel through a list of locations and return to his starting point. A thorough discussion of the data sets that provide the list of cities and their distances follows in section 5.

In a TSP graph, vertices represent the cities and edges the routes. The distance of the route can be computed by the weights on the edges. It is a minimization problem in which each vertex is visited exactly once before starting and ending at the same one. In the case that no route between two cities exist, the graph can be complete without altering the optimal tour, by adding an arbitrary long enough edge.

It's worth noting that asymmetry and symmetry are two major distinctions in the TSP. The distance between two cities in the symmetric TSP is the same in both directions, resulting in an undirected graph. Paths may not exist in both directions in the asymmetric TSP, or their distances may differ, resulting in a directed graph. In this thesis, symmetric TSP instances is addressed.

3.1 Formulations

The TSP has been shown to be NP-hard, as mentioned by Karp in 1972 (Michael Jünger Gerhard Reinelt and Giovanni Rinaldi, 1995). The decision problem variant of TSP is NP-complete (Toby Walsh and Ian P. Gent, 1996). Even when the cities are on the plane with Euclidean distances, the problem remains NP-hard. By removing the requirement of visiting each city only once, the problem is still considered NP-hard. The reason for this, is the fact that there is an optimum tour that visits each city only once, satisfying the triangle inequality.

The TSP may be expressed as a mixed integer linear program (Papadimitriou, 1998). There are also alternative formulations, including the Miller–Tucker–Zemlin (MTZ) and Dantzig–Fulkerson–Johnson (DFJ). The DFJ formulation is more powerful, yet the MTZ formulation is still beneficial in some situations (Velednitsky, 2017).

The cities are labeled with the integers $i = 1, ..., n$, in both of these formulations, and the distance between cities i and j is taken to be $c_{ij} > 0$.

The main variables in the formulations are:

$$x_{ij} = \begin{cases} 1, \text{the path goes from city i to city j} \\ 0, \text{otherwise} \end{cases}$$

Which are binary ensuring that the below formulations are integer. In particular, the objective function is :

$$min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \quad (1)$$

Variables x_{ij} fluctuate over all subsets of edges, while their minimum is $x_{ij} = 0$. The requirement that each vertex has precisely one incoming and outgoing edge, is stated by equations (2) and (3), and is captured by both formulations.

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad for \quad j = 1, ..n \quad (2)$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \quad for \quad i = 1, ..., n \quad (3)$$

Equations (2) and (3) guarantee that the edges will form a tour and allow for several solutions, since there can not be only one tour that traverses through all vertices. Therefore, the difficulty of this problem lies in the way that MTZ and DFJ formulations convey this assumption.

Concerning the MTZ formulation, for each $i = 1, 2, ..., n$ a dummy variable u_i is utilized to indicate which cities are visited and in what sequence. Thus $u_i < u_j$ means that city i is visited before city j. When moving from the first city to city i it can be interesting to explore how balancing the number of edges with the values of u_i affects the formulation.

MTZ uses $(n-1)(n-2)$ linear constraints.

$$u_j + (n-2) \geq u_i + (n-1)x_{ij} \quad (4)$$

for all distinct $i, j \in \{2, ..., n\}$

The MTZ formulation of TSP is thus the following integer linear programming problem:

$$\min \sum_{i=1}^{n} \sum_{j\neq i, j=1}^{n} c_{ij}x_{ij} \quad (5)$$

$$x_{ij} \in \{0, 1\} \quad i, j = 1, \ldots, n \quad (6)$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad j = 1, \ldots, n \quad (7)$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \quad i = 1, \ldots, n \quad (8)$$

$$u_i - u_j + (n-1)x_{ij} \leq n-2 \quad 2 \leq i \neq j \leq n \quad (9)$$

$$1 \leq u_i \leq n-1 \quad 2 \leq i \leq n. \quad (10)$$

Equation (7) ensures that every city is reached from only another one, while equation (8) guarantees that every departure that takes place in a city reaches another exactly once. Constraints (9) and (10) make sure that there is just one tour that covers all cities, rather than two or more disconnected trips that merely cover all cities together. To demonstrate this, consider the following: (i) every possible solution has only one closed sequence of cities, and (ii) there exist values for the dummy variables u_i that fulfill the criteria for every single tour spanning all cities.

It suffices to establish that any viable solution comprises only one closed sequence of cities by demonstrating that every subtour goes through city 1. There can only be one such a tour due to equations (7) and (8).
It must now be demonstrated that there are values for the dummy variables u_i that meet the conditions for each and every tour that covers all cities. Define the trip as starting (and ending) at city 1 without losing its generality. Choose $u_i = t$ if city i is visited in step $t(i, t = 2, 3, ..., n)$. Then $u_i - u_j \leq n - 2$, since u_i can be no greater than n and u_j can be no less than 2; hence the constraints are satisfied whenever $x_{ij} = 0$. For $x_{ij} = 1$ we have: $u_i - u_j + (n-1)x_{ij} = (t) - (t+1) + n - 1 = n - 2$, satisfying the constraint.
For the Dantzig–Fulkerson–Johnson formulation the cities are labeled with the numbers $1, .., n$ and the below is defined as:

$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

Take $c_{ij} > 0$ to be the distance from city i to city j. Then TSP can be written as the following integer linear programming problem:

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \quad (12)$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \quad (13)$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \quad (14)$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1 \qquad \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2 \quad (15)$$

Constraint (15) guarantees that no acceptable subset Q forms a sub-tour. Therefore, the result is a single tour rather than a collection of smaller tours.

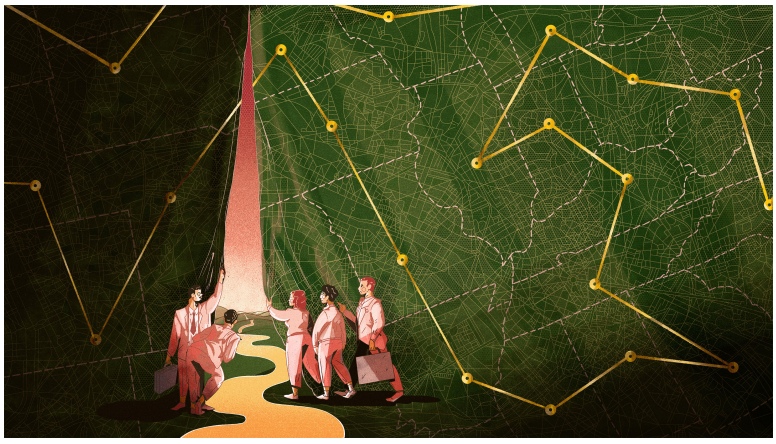


Figure 3.1: Cracking the Traveling Salesman Problem — Quanta Magazine,2020

4. Methodology

In this section, a deeper analysis on the methods that have been implemented for the TSP, is provided. Next, details about each of the steps, that have been taken to reach a solution for the TSP are presented. That includes the selection of different parameters, the form of the input data and all the improvements that established efficient solutions.

4.1 Kohonen's Neural Network

The starting point in the Kohonen's Neural Network approach concerns the data that are inserted in the algorithm and form a graph. While training the data, a network in the form of an adjacency matrix is built using data from a symmetric TSP instance.

4.1.1 Self-Organizing Map

The way that the neural network is created starting from the input data leads to the introduction of the concept of Self - Organizing Map (SOM). SOMs, have two modes of operation: training and mapping. Training employs an input data set to create a map space, which is a lower-dimensional representation of the data. Second, mapping uses the produced map to classify further input data.

The purpose of the training phase is to convert a p - dimensional input space (with p variables) into a two-dimensional map space. The map space is constructed by neurons that are structured in a two-dimensional hexagonal or rectangular framework. The number of neurons or nodes and their layout are pre - determined (Hollmen, 1996).

For every node in the map space, a weight vector specifies its position in the input space. The training process takes place by shifting these vectors towards the input data. That means, that a metric such as the Euclidean distance is lowered without affecting the way that a map space is formed. In the end of this phase, selecting the weight vector with the lowest metric in the map, contributes to the classification of additional input data.

A competition layer is used to form a neural network that uses competitive, unsupervised learning in this thesis project. This approach is based on the WTM (Winner Takes Most) algorithm, introduced by Teuvo Kohonen (Kohonen, 2001).

According to Brocki and Koržinek (2007), when an input vector (a pattern) is presented, a distance to each neuron's synaptic weights is determined. The neuron with the best weights associated with the current input vector is the winner. Correlation is calculated by taking the scalar product of the input vector and synaptic weights. Only the winning neuron adjusts its synaptic weights to the input pattern's point. Other neurons' synaptic weights remain unchanged. Adding to the foregoing, neurons that are activated seldom have a

better chance of winning. Because it modifies synaptic weights in one cycle, this WTM technique has better convergence properties than other strategies. This means that not only the winner but also its neighbors adjust, resulting in the fact that the closer a neuron is to the winner, the less its weights change.
As follows is a description of the learning process:

$$W_i \leftarrow W_i + \theta F(\lambda, d(i,j))(x - W_i) \quad (16)$$

where $i \in (0, ..., n)$, with n numbers of neurons, W_i represents all synaptic weights of the winning neuron, and θ is learning rate, while x stands for current input vector. $F(\lambda, d(i,j))$ defines the neighborhood. Thus, classic Self Organizing Map (SOM) can be created when function $F(\lambda, d(i,j)))$ is defined as:

$$F(\lambda, d(i,j)) = \begin{cases} 1, \text{for} \quad d(i,j) \leq \lambda \\ 0, \text{for} \quad d(i,j) > \lambda \end{cases} \quad (17)$$

where $d(i,j)$ is euclidean distance between the winning i^{th} neuron and some other j^{th} neuron in the map that belong to the winner's neighborhood. The neighborhood radius or the range of the neighborhood λ is depreciated after every iteration. The euclidean distance between the input vector and all neural weights must be determined to train Kohonen SOM. The winning neuron with the smallest distance to the input vector is picked, and its weights are slightly changed to match the input vector's direction. In the same way, the weights of nearby neurons are then adjusted. As a result, the learning rate θ and the neighborhood radius λ get smaller during this learning procedure. Iteration by iteration, the SOM is transferred from global to local organization causing θ and λ to become smaller every time, since the neighborhood shrivels. When it comes to the function that defines the neighborhood, it can be seen that at the start of the learning process, the neighborhood radius λ equals a maximum value λ_{max}. In the computational experiments section, the value that will yield a near-optimal outcome will be investigated. The widely held belief is that when the Gaussian Neighborhood Function (GNF) is utilized instead of the rectangular one, substantially superior learning outcomes can be obtained (Mokriš & Forgáč, 2004).
The GNF can be described as follows :

$$F(\lambda, d(i,j)) = exp(-\frac{d(i,j)^2}{2\lambda^2}) \quad (18)$$

The Gaussian function appears to create challenges in calculations and significant time consumption due to its complexity, arising from squaring, division, and exponential operations. This was the motivation to verify whether the Triangular Neighborhood Function (TNF) might be utilized in place of the GNF (Dlugosz, Kolasa, & Pedrycz, 2010).
This function is defined as:

$$F(\lambda, d(i,j)) = \begin{cases} -\alpha(\theta_0)(\lambda - d(i,j)) + c, \text{if } \ d(i,j) \leq \lambda \\ 0, \text{if } \ d(i,j) > \lambda \end{cases} \tag{19}$$

In the above equation (19), the $\alpha()$ function indicates the steepness of TNF, θ_0 the learning rate of the winner neuron and parameter c expresses the bias. After each training period, these parameters converge towards zero.

It's worth noting that the technique and methodology used in this thesis study allow for the selection of the aforementioned neighborhood functions.

Elaborating on how the Kohonen Neural Network using a SOM is constructed and the way it works, the following figure is utilized.

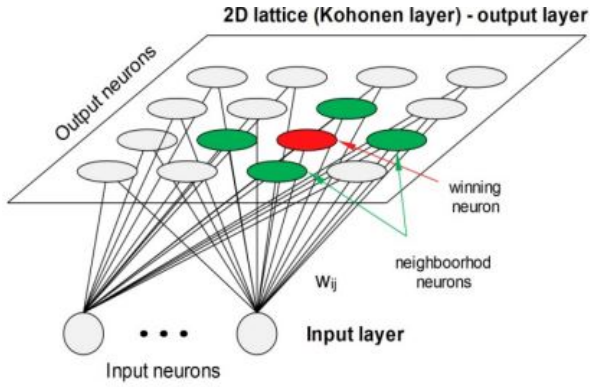


Figure 4.1: Visualization of a Kohonen SOM with a 2D grid of output neurons - (D.Markovic, M.Madic, V.Tomic, S.Stojkovic, 2012)

The Kohonen Neural Network translates input vectors of any dimension to a map of two dimensions, as previously mentioned (Kohonen layer). A two-dimensional grid representing the output neurons is used to construct the feature map. A weighted connection exists between all input neurons and the neurons in the output layer. As was already explained, in the competitive layer, the winning neuron is the one that is most similar to the input data. The winning neuron's and its surrounding neurons' weights are updated in the next phase. Each neuron has four neighbors because the neighborhood of a neuron is typically thought of as being two dimensional.

4.1.2 Algorithm

Taking into account how the Kohonen Neural Network is built utilizing the SOM principle, it is now going to be utilized to solve a n city problem. The Traveling Salesman problem is made up of two one-dimensional layers: a three-neuron input layer and a n-neuron output layer. The number of neurons must equal the number of cities.

The random selection of a city from the algorithm results to the activation of the input neurons, since neurons and a city's coordinates are related. The two city coordinates plus a third normalization component, ensure that all input vectors have the same Euclidean length and no two input vectors are col-linear. If the weights of a neuron are equal to the coordinates of a city, the city is represented by that neuron. A neuron is allocated to a city, and neurons are grouped into a vector that specifies the order in which cities should be visited.

A neuron in the output layer is linked to every neuron in the input layer. The output neuron with the highest activation denotes the image of that city. Initially, all connections are given random uniformly distributed values. The TSP necessitates the creation of an ordered vector mapping of the input vectors in order to calculate a solution. The position of a neuron in the output layer represents the position of a city in the solution once the aforementioned technique has been completed. By computing the neuron with the highest activation for each city and sorting the cities by the indices of their associated neurons, a route or a network of neurons may be formed. By computing every city's image and then sorting the cities by the indices, a route or pathway of neurons is constructed (Favata & Walker, 1991).

Predefined values are set for the gain parameters (learning rate θ and neighborhood radius λ) while the above process takes place until the gain parameters becomes non-positive. These values will be further explored in the computational experiments in section 7.

A pseudo-code of the aforementioned process is as follows:

Algorithm 1 TSP Solver based on a Self - Organizing Map

1: **Input** Data-set containing nodes representing cities organized in SOM.
2: **Begin** Randomize the map's nodes' weight vectors
3: **While** The number of iterations is lower than the iteration limit
4: Randomly pick an input data vector and traverse each node in the map
5: Use the Euclidean distance formula to find the similarity between the input vector and the map's node's weight vector.
6: Track the node that produces the smallest distance (this node is the winning neuron)
7: Update the weight vectors of the nodes in the neighborhood of the winner neuron (including the winning neuron) by pulling them closer to the input vector, using the aforementioned options of the neighborhood functions (equations (18),(19)).
8: Reduce the learning rate θ and the neighborhood radius λ
9: **End**
10: **End**
11: **Output** : Pathway of neurons in the Kohonen Neural Network

In order to illustrate how the tour is constructed the following figures are created (Figure 4.2 & Figure 4.3) using an example of 10 cities and observing how the SOM of the Kohonen Neural Network evolves by the

number of iterations. As mentioned in the above pseudo-code, an input data vector (that includes 3 cities, for example) is randomly chosen in the left hand side of Figure 4.2. It begins as an elastic ring seeking to adapt to its shortest shape possible. It maintains its position and tries to identify better local routes within of it using the techniques that are described above (Step 5, 6 & 7).

Aiming to successfully achieve this, many neurons including the winning neuron are added to the network every time. The parameters of the learning rate θ and the neighborhood radius λ are degraded ensuring the exploration of the map. Thus, the answer is refined and results to a pathway (of 10 cities) similar to the one depicted in the right hand side of Figure 4.3.

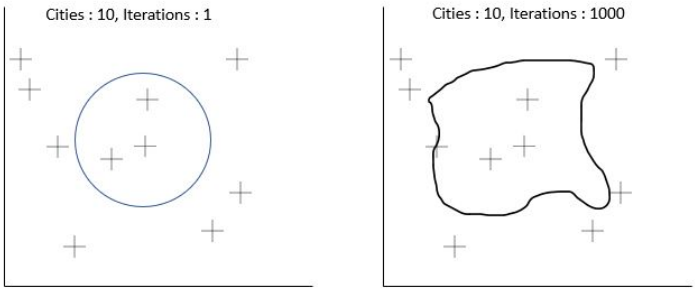


Figure 4.2: Kohonen SOM Algorithm for TSP - Author's creation

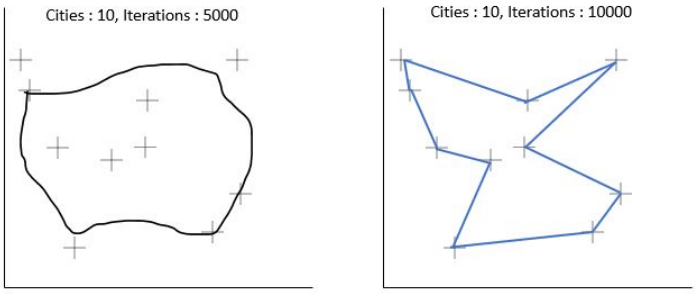


Figure 4.3: Kohonen SOM Algorithm for TSP - Author's creation

4.1.3 Settings

Using a SOM for our Kohonen Neural Network as a solver for the TSP rises comments on the parameters that are of high influence for the network. These are concerned to be the learning rate and the neighborhood function, especially in the training process.

Learning rates

In SOM learning, the learning rate is a training parameter that regulates the size of the weight vector. In SOM, linear, inverse of time, and power series are the most commonly employed learning rate functions. (J.

Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas, 2000). Linear, inverse of time and power series are defined below respectively.

$$\theta(t) = \theta(0)\frac{1}{t} \quad (20)$$

$$\theta(t,T) = \theta(0)(1 - \frac{t}{T}) \quad (21)$$

$$\theta(t,T) = \theta(0)e^{\frac{t}{T}} \quad (22)$$

Here T is the number of iterations and t is the order number of a current iteration.

Neighborhood functions

The neighborhood function defines the pace of the changes that will be happening around the winner neuron. Moreover, the neighborhood function has an impact on the SOM process's training result. Thus, choosing the right neighborhood function for every data set is extremely important. Many functions are employed in SOM, however the most common are Triangular and Gaussian, which are decreasing functions in the defined region of the winning neuron.

These functions play crucial role in the definition of the neighborhood radius as well. The neighborhood function depends on the grid-distance between the winning neuron and another neuron. The neighborhood function diminishes with time, independent of its functional shape. The map self-organizes on a global scale at first, when the neighborhood is vast and large. The weights are then linking up to local estimates when the neighborhood has decreased to just a few neurons. While the learning rate and neighborhood function decline consistently as the number of iterations grows in some implementations, they decrease in a step-wise way in others, once every T steps.

4.2 Elitist Ant System

In the field of Ant Colony Optimization, there is an alternative representation of the Traveling Salesman Problem. A complete weighted graph $G = (N, A)$ may be used to represent the TSP, with N representing the set of cities or nodes and A representing the set of arcs entirely linking the nodes. The distance between cities i and j is represented by the weight d_{ij} allocated to each arc. In this sense, the TSP issue may be thought of as finding the shortest Hamiltonian circuit of the graph, where a Hamiltonian circuit is a closed walk (a tour) that visits each node of G precisely once.

A solution to an instance of the TSP may be described as a permutation of the city indices, according to Dorigo & Stützle's book from 2004. This permutation is cyclic, which means that the absolute position of a city in a tour is irrelevant, only the relative order matters.

The TSP's only limitation is that all cities must be visited, and each city can only be visited once. This limitation is enforced if an ant picks the next city only among those it hasn't visited yet at each building phase. All cities that have yet to be visited make up an ant's viable neighborhood in city i, where k is the ant's identifier.

A permutation π of the node indices $\{1, 2, ..., n\}$ that has the shortest length $f(\pi)$ is an optimum TSP solution, where $f(\pi)$ is given by:

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)} \quad (23)$$

The set of all possible routes can be considered as the problem's state space. The completed connected graph $G_C = (C, L)$ that will be used for this ACO algorithm is similar to the problem graph mentioned above. The set L contributes to the connection of the cities in C, by giving each link a weight which is correlated with the distance between cities i and j. The pheromone trails are affiliated with the arcs. Thus τ_{ij} denotes the likelihood that node j will be visited after visiting node i. The heuristic information, on the other hand, is usually inversely related to the distance between nodes i and j, so $\eta_{ij} = 1/d_{ij}$.

According to Dorigo and Stützle (2004), η_{ij} indicates the heuristic attractiveness of traveling straight from city i to city j. Based on the pheromone and heuristic values, τ_{ij} and η_{ij} respectively, on the arcs that link unvisited cities, an ant arriving in city i selects the next city j as shown in Figure 4.2 below.

The following basic constructive method is applied to each ant in order to create tours:

- Determine the location of the ant's beginning city.
- Create a tour using the information from the pheromone and heuristic values. Iteratively add cities from the set that includes the unvisited ones, until all are visited by the ant.
- Return to your original location.

Pheromone trails and heuristic information values are used to make a probabilistic judgement. After all of the ants have finished their travels, they are allowed to deposit pheromone on the routes they have taken.

The following is an example of an algorithmic scheme:

Algorithm 2 ACO Algorithmic Scheme for TSP

1: Set parameters. Initialize pheromone trails.
2: While termination criteria are not met
3: Construct Ant Solutions
4: Apply Local Search
5: Update Pheromones
6: End

In an ACO algorithm, the first step that takes place is the initialization of the pheromone trails and parameters. Then a loop follows (Step 2), where the ants' routes are designed, constructed and enhanced by using a local search algorithm. In the final stage the ants evaporate the pheromone trails and update them using their search findings.
It is worth mentioning that in some circumstances, the routes that are constructed by the ants in step 3 can be improved by utilizing a local search technique. This can be considered as a potential daemon action in an ACO algorithm.

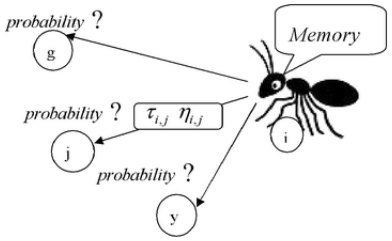


Figure 4.4: The ant's behavior - Olivier Rukundo and Hanqiang Cao, 2021

4.2.1 Algorithm Description

The Elitist Ant System algorithm follows from the reference on the Ant Colony Optimization by Marco Dorigo (1992). As an Ant System there are similarities with the ACO algorithm that was introduced in the previous section. The identical solution formulation approach and pheromone evaporation process are responsible for these commonalities. Differences are observed in the way that the pheromone update is performed and in the management of the pheromone trails. In this algorithm, we are referring to an ant-cycle version of the pheromone update, meaning that this update takes place after all the ants had constructed the tours and the amount of pheromone deposited by each ant was set to be a function of the tour quality.
In the implemented code set for the Elitist Ant System the ACO Algorithmic Scheme for TSP that was used is the following :

Algorithm 3 ACO Algorithmic Scheme for the EAS in TSP

1: Initialize values like clearing each ant's old tour and clearing the old tour cost.
2: Using the pheromone and heuristic information (equation 26), make the ants construct the tour.
3: The tour that is constructed by step 2 is improved in this step by Hill Climbing (Local Search).
4: The pheromone is evaporated after each iteration.
5: Let the ants deposit the pheromone.
6: End

In the Elitist Ant Strategy algorithm, the main idea behind it is to provide additional reinforcement to the arcs belonging to the best route so far found by the algorithm. This route or tour is defined as T^{bs}, where parameter e indicates the weight that is assigned to it and C^{bs} its length. By appending the value of $\frac{e}{C^{bs}}$ to the arcs of the tour the supplementary reinforcement is achieved. Thus, this can be seen as an additional pheromone level placed by an additional ant called best so far ant.

As depicted in both Algorithm 2 and 3 the two main steps of the Ant System algorithm concern the construction of the initial ants' solution and the pheromones' update. In order to efficiently initialize the pheromone levels in an Ant System, a pheromone value higher than the expected in one iteration can be chosen and set heuristically. In this implementation a greedy heuristic is used to generate a maximum, in a greedy manner, given an initial tour. A 2-opt Hill Climbing heuristic is used which basically marks 2 points in the sequence and reverse that part of the sequence searching for a better neighborhood. This is repeated until a local minimum is reached.

The way the pheromone trails are updated is another important aspect of this strategy. As previously mentioned, the best tour is reinforced by adding a quantity to its arcs, which is determined by a criterion that determines the tour's weight. Thus, an appropriate value for this parameter needs to be considered in order to allow the Elitist Ant System to provide better solutions with a low number of iterations. More specifically an equation for the pheromone deposit can be seen as:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs} \quad \forall(i,j) \in L \quad (24)$$

where $\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C^{k}}, \text{if } (i,j) \text{ belongs to } T^{k} \\ 0, \text{otherwise} \end{cases}$ and $\Delta\tau_{ij}^{bs} = \begin{cases} \frac{1}{C^{bs}}, \text{if } (i,j) \text{ belongs to } T^{bs} \\ 0, \text{otherwise} \end{cases}$

4.2.2 Settings

Initialization of the pheromone trails

One of the main settings that need consideration are the pheromone trails or levels. If the initial pheromone values are too low, the search is immediately skewed by the ants' first tours, which leads to the exploration of smaller areas of the search space in general. If the starting pheromone levels are too high, however, several iterations are lost while waiting for pheromone evaporation to drop enough pheromone values so that ant-added pheromone may begin to bias the search. In this particular Elitist Ant System the pheromone trails is to set them to a value slightly higher than the expected amount of pheromone deposited by the ants in one iteration.

A rough estimate of this value can be obtained by :

$$\tau_{ij} = \tau_0 = \frac{m}{C^{nm}} \quad (25)$$

where m is the number of ants, and C^{nm} is the length of a tour generated by the Hill Climbing heuristic.

Tour Construction

In the Elitist Ant System m artificial ants that build a route of the TSP at the same time are placed in randomly selected cities using the following rule. In every tour construction step the ant uses a stochastic rule, known as the random proportional rule to determine which city to visit next. Specifically, the likelihood that an ant k which is now in city i and chooses to travel to city j is :

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \quad if \quad j \in N_i^k \quad (26)$$

The parameters in (26) can be explained as follows : $\eta_{ij} = 1/d_{ij}$ is the heuristic value, while parameters α and β indicate the proportional effect of the pheromone level and the heuristic information. The unvisited cities compose a possible ant neighborhood with the ant's k location being city i. This is expressed by ${N_i}^k$. Extreme scenarios in the tour construction phase can be derived by the following cases. Selecting $\alpha = 0$ the closest cities have higher probability of being selected. In the case of $\beta = 0$ just pheromone is utilized, leading to unsatisfactory outcomes. On the contrary, when $\alpha > 1$ a fast convergence is achieved , since the ant construct the same tour and travel the same route, which offers a sub-optimal solution.

4.3 K-means clustering algorithm

One of the most common approach in order to tackle the Traveling Salesman problem is to use the 2-opt algorithm. The traveling salesman problem is solved using the 2-opt heuristic, which is a local search technique. The core principle of the 2-opt heuristic, first introduced by Croes in 1958, is that when a route crosses over itself, the nodes are reordered to remove the crossing. Elaborating on this, the goal of the local search method is that after the reordering, all edges are shorter than the edges in the previous version. The result of this, is the creation of a Hamiltonian cycle in the 2-opt neighborhood of a solution s. A local optima can be derived, implying that no better route can be found in this process. This does not, however, ensure that the TSP will be solved optimally.

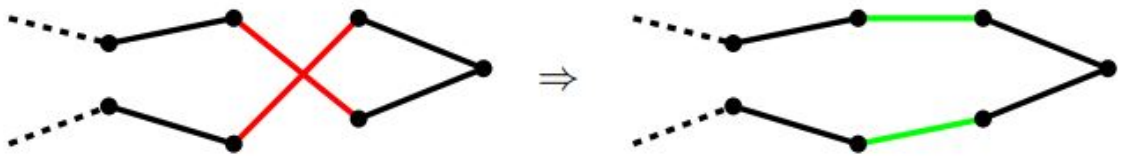


Figure 4.5: Author's creation - Idea behind the 2-opt algorithm (Croes, 1958)

The pseudo-code of the most common approach in order to solve the TSP using a 2-opt algorithm that checks if a swap of two edges yields to a better result by removing a crossing is as follows :

Algorithm 4 2-OPT pseudo-code

1: **Input** : A complete graph with distances defined on the edges and a route and its distance
2: **Output** : A 2-opt feasible route
3: **for** $i \in N$ eligible to be swapped and **for** $j \in N$ eligible to be swapped such that $j > i$ **do**
4: Apply 2-opt swap to i and j: create the new route as follows:
5: Take route up to i and add in order
6: Take route from j to i (both including) and add in reverse order
7: Take the route after k and add in order
8: Calculate new distance
9: **if** new distance $<$ distance **then** Update route to include new ordering
10: Repeat until no improvement is made
11: End

4.3.1 Algorithm Description

The approach that will be followed in this method instead of correcting the crossings as was described before, now it creates as few as possible costly ones by grouping the cities.

It can be considered as a modified version of the K-means clustering algorithm that clusters nearby cities, groups them and then groups the groups. It then links all groups by opening portals from the nearest city of C_1 to the closest city of C_2, thereby correcting the order of visits.

This K-means clustering algorithm can be seen as a pre - processing step but also as a merging step at the end. The idea is the following:

Grouping all the nearby (from the starting city) cities into clusters. Then aiming to find a good path within the clusters in order to minimize the problem into a smaller one. The sequence of visiting clusters is determined and is connected to the closest neighboring cities of the two clusters in sequence. A portal is the edge that connects two clusters. At the end the sequence of each cluster is merged in order to form a final answer to the Traveling Salesman Problem instance.

Exploring in more detail this K-means clustering algorithm, the following pseudo-code is being followed:

Algorithm 5 K-means clustering algorithm pseudo-code

1: **Input** : Inserted variables of N number of cities, the input of cities as objects with parameters : id, x, y and to which cluster they belong. Euclidean Distance to determine the distance between two cities and the cost of a tour is being evaluated.
2: **Output** : An optimal route
3: A TSP Solver is called out using the aforementioned K-means strategy
4: Nearby cities are grouped into clusters, finding a good path within the clusters minimizing the problem into a smaller one.
5: A standard procedure is followed to initialize the centroids of clusters using a greedy algorithm. Cities are assigned to new centroids and the TSP problem is solved within each cluster.
6: The pair of cities(vertices) of two closest centroids that are closest to each other is found and hence a good candidate for a portal is made.
7: The sequences of each cluster are merged in order to form a final answer to the TSP
8: End

4.3.2 Settings

Clusters

The initialization of the parameters is contained in the implemented code set for this approach's core class, which is where the K-means technique is applied. This refers to the number of clusters and iterations. A greedy approach is used to initialize the cluster centroids. Then surrounding cities are grouped into clusters,

and a good path within the clusters is discovered, reducing the problem to a manageable size. The order in which the visiting clusters are visited is established, and the clusters of the nearest neighboring cities are connected. This sequence includes a cluster ID, a list of cities that make up that cluster, and a tour that takes place within that cluster. A portal is a connection between two clusters that is found by determining the pair of cities (vertices) that are nearest to each other. The final phase and technique involves joining the sequences of each cluster to create a complete TSP solution.

5. Data

Taking into consideration of the content mentioned in the previous chapters, it can be easily perceived that datasets play a crucial role regarding the performance, running times and behavior of the methods. That is the reason why the planning of the dataset collection, consists of three components: the source, the size and the format.

5.1 Data Sources and Formats

All required data sets are distinguished by their origin. This means that two kind of data sets are used in order to simulate their performance on the described methodology described in the previous section. The first kind of data sets concern random simulated data that are carefully chosen from free online resources on the TSP research.
The most difficult part of the data collection has to do with the format of the data sets and how they are generated. It can be easily understood that having three different methods requires different format of the datasets that will be used as input. For the first kind of data, the format of the random simulated datasets is described as follows :

- For the Kohonen Neural Network (SOM): The data set that will be the input for this method will contain in the first line the amount of cities that will be explored (either 100 or 1000) and in the following lines the coordinates x and y of every city. In this way, a graph required for the Kohonen Neural Network can be constructed.

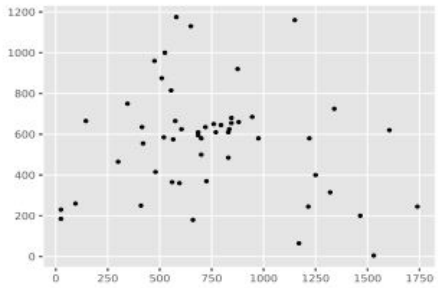
- For the Elitist Ant System: The data set that will be used as input for this method will contain in the first line the number of cities that will be explored (either 100 or 1000) and in the following $n-1$ lines an n x n of the adjacency matrix of the city graph with the assumption to be symmetric. More specifically, the format of the datasets that was used for the Kohonen Neural Network method and the K - means clustering algorithm has been converted to the adjacency matrix for the EAS.

- For the K-means clustering algorithm: Similar to the first method, the data set for this method contains in the first line the number of cities to be explored, while in the next lines contains the x and y coordinates of each city.

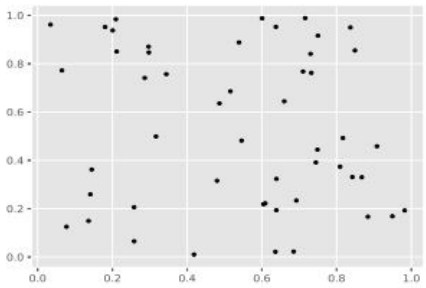
Regarding the randomly simulated datasets, the way that they are created as TSP instances is quite simple. Within a two - dimensional unit space $[0, 1000]$ x $[0, 1000]$, coordinate values have been chosen and generated randomly. With the size distinction that is discussed in the next subsection, the random generated datasets have graphs of either 100 or 1000 nodes. The small random simulated dataset is provided in the Appendix 10.2 for reference.

It is significant to mention that the randomly generated datasets do not follow a particular pattern. According to Nasrin Sultana, Jeffrey Chan, A. K. Qin, Tabinda Sarwar (2020), the main difficulty of the TSP problem is to indicate the way that the nodes representing cities are distributed in the space. Thus, in the figure (Figure 5.1.a) below we can see that Berlin52 which is a TSPLIB instance is considered a hard problem due to its strict constraints on the data points. Searching for solutions in that space is relatively hard. Similar patterns and constraints are included in the TSPLIB instances that are utilized in this thesis study (kroA100 & pr1002).

On the other hand, looking at Figure 5.1.b, TSP50 is an example of an easy TSP instance, similar to those that are used in this thesis study (either with 100 or 1000 cities), as the cities are widely spread all over the space and the data points are loosely constrained. Classifying how hard is a TSP instance to be solved is firmly related to the distribution of the grid space and the distances between the nodes.



(a) Set of points spread on the space for the problem Berlin52



(b) Set of points spread on the space for the problem TSP50 (TSP50 is randomly generated)

Figure 5.1: Distribution of cities for a TSPLIB instance and a randomly generated TSP instance - Nasrin Sultana, Jeffrey Chan, A. K. Qin, Tabinda Sarwar (2020)

The second kind of data sets is collected from the official free online library TSPLIB for Traveling Salesman Problem instances. TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types. The TSPLIB instances that are selected for this thesis study are the : kroA100.tsp & pr1002.tsp. The reason of why these are selected lies on their sizes which is explained in the next subsection.

In general, all data sets are expected to be provided as .txt files or .tsp instances. Therefore, the proposed data sets are seen to be legitimate for applying the concepts suggested in this thesis.

5.2 Data set size

As far as the data sets sizes are concerned, a distinction between large and small data sets is essential to be made. In order to justify this differentiation between large and small, a short literature review on the scientific research that will be used in this thesis study is required. According to Brocki and Koržinek (2007) an instance of 100 cities is small while an instance of 1000 cities is considered to be large.

Distinguishing the instances based on these sizes, one can clearly understand how the behavior of a Kohonen Neural Network can change. On the other hand, following Dorigo (1992), the symmetric TSPLIB instances D198 and RAT783 have been used in order to study the performance of the Elitist Ant System compared to other Ant Colony Optimization algorithms. Here, a TSP instance with 198 cities is considered small and a TSP instance with 783 cities is considered large.

Moreover, a similar distinction between large and small TSP instances has been made from Karakoyun (2019). In that study, problem instances for TSP taken from the TSPLIB with a size of 52 cities (Berlin52) and a size of 1002 cities (Pr1002) were considered as benchmark, on which a modified K-means clustering algorithm was applied.

Thus, a small data set contains the number of $n = 100$ cities, while a large data set contains the number of $n = 1000$ cities. By distinguishing the data sets in these sizes, it is expected that different kind of insights will be provided by this thesis study.

To sum up, in total, the algorithms described in the previous sections will be applied in four different data sets. Namely,

Name of data set	Number of cities
Random simulated data set (small)	100
kroA100 (TSPLIB)	100
Random simulated data set (large)	1000
pr1002 (TSPLIB)	1002

6. Results

The validity of the three proposed approaches discussed in section 4 is investigated in this part for the four different data sets. The results are presented in the form of tables and visualizations, allowing for a clear understanding of the metrics that were employed in order to provide a complete solution for the TSP. In the below summary table the best outcomes of every approach on each dataset are presented:

Table 6.1: Summary of best performing approaches for each data set

Approach	Kohonen Neural Network			
Data Set	Simulated data set (100)	kroA100	Simulated data set (1000)	pr1002
Route Cost	8952.0	22735.95	28263.99	268830.06
Approach	Elitist Ant System			
Data Set	Simulated data set (100)	kroA100	Simulated data set (1000)	pr1002
Route Cost	8064.79	23348.57	25278.73	280346.8
Approach	K-means Clustering Algorithm			
Data Set	Simulated data set (100)	kroA100	Simulated data set (1000)	pr1002
Route Cost	10376	26815	30055	321154

6.1 Kohonen's Neural Network

Random simulated data set with 100 cities - Using the Gaussian neighborhood function

The findings of Kohonen's Neural Network application on a random simulated data set of 100 cities are provided in the following two tables (Table 6.2 & Table 6.3). The neighborhood radius begins at 50 and decreases to 0.9 at the conclusion of iterations. Similarly, the learning rate begins at 0.99 and decreases to 0.01 when all iterations are completed. Various iteration values are also employed, as the behavior of Kohonen's neural network is dependent on them. It is clear that by employing the Gaussian as the neighborhood function and reducing the number of system iterations, better route cost solutions are found.

Table 6.2: Kohonen Neural Network on the random simulated data set for 100 cities using GNF.

Simulated data set (100)	Using the Gaussian	Simulated data set (100)	Using the Gaussian	Simulated data set (100)	Using the Gaussian
Iterations	10000	Iterations	5000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate	0.01	Learning Rate	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius	0.9	Neighborhood Radius	0.9
Route Cost	10658.97	Route Cost	17713.12	Route Cost	28499.21
Running Time	5.2 s	Running Time	3.1 s	Running Time	0.76 s

Random simulated data set with 100 cities - Using the Triangular neighborhood function

For the same data set, using the Triangular as the neighborhood function, it is clear that lowering the number of iterations increases the cost of the routes, yet running times follow the same pattern as previously, suggesting that running times are completely dependent on the number of iterations.

Table 6.3: Kohonen Neural Network on the random simulated data set for 100 cities using TNF.

Simulated data set (100)	Using the Triangular	Simulated data set (100)	Using the Triangular	Simulated data set (100)	Using the Triangular
Iterations	10000	Iterations	5000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate	0.01	Learning Rate	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius	0.9	Neighborhood Radius	0.9
Route Cost	8952.00	Route Cost	13293.43	Route Cost	14070.16
Running Time	2.99 s	Running Time	1.57 s	Running Time	0.56 s

TSP Library data set with 100 cities kroA100 - Using the Gaussian neighborhood function

Exploring the data set generated from the TSP Library containing 100 cities, it can be easily seen that the results do not follow a particular pattern.

In Table 6.4 running times decrease as the number of iterations decreases, however the cost of the routes does not follow the same trend as before. The route cost is relatively high in the case of 1000 iterations. Executing more iteration the route cost decreases.

Table 6.4: Kohonen Neural Network on kroA100 using GNF.

TSPLib data set (kroA100)	Using the Gaussian	TSPLib data set (kroA100)	Using the Gaussian	TSPLib data set (kroA100)	Using the Gaussian
Iterations	10000	Iterations	5000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate	0.01	Learning Rate	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius	0.9	Neighborhood Radius	0.9
Route Cost	22735.95	Route Cost	64538.71	Route Cost	81846.8
Running Time	6.3 s	Running Time	3.46 s	Running Time	1.2 s

TSP Library data set with 100 cities kroA100 - Using the Triangular neighborhood function

A different pattern applies in this situation (Table 6.5). The situation of 5000 iterations with the triangular neighborhood function is too expensive, hence the case of 10000 iterations is the most appealing in terms of route cost and running times.

Table 6.5: Kohonen Neural Network on kroA100 using TNF.

TSPLib data set (kroA100)	Using the Triangular	TSPLib data set (kroA100)	Using the Triangular	TSPLib data set (kroA100)	Using the Triangular
Iterations	10000	Iterations	5000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate	0.01	Learning Rate	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius	0.9	Neighborhood Radius	0.9
Route Cost	42416.9	Route Cost	44864.57	Route Cost	43825.3
Running Time	3.45 s	Running Time	1.8 s	Running Time	0.8 s

Random simulated data set with 1000 cities - Using the Gaussian neighborhood function

The number of iterations set in the below tables (Table 6.6 & Table 6.7) is for the goal of not exceeding the time restriction of 1000 seconds in the 1000 city scenarios below, both for the random simulated data and the TSP Library instance of 1002 cities.

It is clear from the table below (Table 6.6) that devoting more time to running the algorithm leads in lower route costs.

Table 6.6: Kohonen Neural Network on the random simulated data set for 1000 cities using GNF.

Simulated data set (1000)	Using the Gaussian	Simulated data set (1000)	Using the Gaussian	Simulated data set (1000)	Using the Gaussian
Iterations	10000	Iterations	5000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate	0.01	Learning Rate	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius	0.9	Neighborhood Radius	0.9
Route Cost	41312.07	Route Cost	55005.96	Route Cost	60732.30
Running Time	444.9 s	Running Time	280.59 s	Running Time	54.49 s

Random simulated data set with 1000 cities - Using the Triangular neighborhood function

When utilizing the Triangular Neighborhood function (Table 6.7), however, a selection of roughly 1000 iterations might result in a very expensive route cost in a short amount of time.

Table 6.7: Kohonen Neural Network on the random simulated data set for 1000 cities using TNF.

Simulated data set (1000)	Using the Triangular	Simulated data set (1000)	Using the Triangular	Simulated data set (1000)	Using the Triangular
Iterations	10000	Iterations	5000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate	0.01	Learning Rate	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius	0.9	Neighborhood Radius	0.9
Route Cost	28263.99	Route Cost	29861.10	Route Cost	80290.76
Running Time	302.45 s	Running Time	156.06 s	Running Time	30.86 s

TSP Library data set with 1002 cities pr1002 - Using the Gaussian neighborhood function

The following tables (Table 6.8 & Table 6.9), which contain findings from the TSP Library instance of 1002 cities, show minor changes from the tables (Table 6.6 & Table 6.7) before. Similarly, using both the Gaussian

and the Triangular neighborhood functions, picking a number of iterations around 10000 might produce a near-optimal outcome in a reasonable amount of time.

Table 6.8: Kohonen Neural Network on pr1002 using GNF.

TSPLib data set (pr1002)	Using the Gaussian	TSPLib data set (pr1002)	Using the Gaussian	TSPLib data set (pr1002)	Using the Gaussian
Iterations	10000	Iterations	5000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate	0.01	Learning Rate	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius	0.9	Neighborhood Radius	0.9
Route Cost	268830.06	Route Cost	842995.02	Route Cost	1468329.97
Running Time	407.93 s	Running Time	175.72 s	Running Time	38.5 s

TSP Library data set with 1002 cities pr1002 - Using the Triangular neighborhood function

Table 6.9: Kohonen Neural Network on pr1002 using TNF.

TSPLib data set (pr1002)	Using the Triangular	TSPLib data set (pr1002)	Using the Triangular	TSPLib data set (pr1002)	Using the Triangular
Iterations	10000	Iterations	5000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate	0.01	Learning Rate	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius	0.9	Neighborhood Radius	0.9
Route Cost	403194.81	Route Cost	464683.21	Route Cost	601807.49
Running Time	245.6 s	Running Time	126.0 s	Running Time	28.9 s

6.2 Elitist Ant System

It's important discussing and demonstrating how the number of iterations chosen influences the route cost in the Elitist Ant System results display. Furthermore, the value of parameter beta fluctuates between 2 and 5, as stated in the method explanation. It is also, examined how the route cost varies and how the running durations are influenced by picking for beta the extreme values of that interval while keeping the other parameters values constant. The values that are used in this implemented algorithm and are considered good parameter values for the EAS according to Dorigo and Stutzle (2004) are the parameter $\alpha = 1$, the evaporation rate: $\rho = 0.5$, the number of ants: $m = n$ (the number of cities) and the initialization value of pheromone trial: $\tau_0 = \frac{e+m}{\rho C^{nn}}$.

In the instance of the random simulated data set of 100 cities, choosing to perform 10 iterations with the parameter beta set to 5 yields the optimum route cost result.

Table 6.10: EAS on the Random simulated data set with 100 cities

Simulated data set (100)	Values	Simulated data set (100)	Values	Simulated data set (100)	Values
Iterations	10	Iterations	10	Iterations	5
Parameter alpha	1	Parameter alpha	1	Parameter alpha	1
Parameter beta	2	Parameter beta	5	Parameter beta	2
ρ	0.5	ρ	0.5	ρ	0.5
Route Cost	8141.27	**Route Cost**	8064.79	**Route Cost**	8239.31
Running Time	4.11 s	**Running Time**	1.38 s	**Running Time**	4.35 s

For the TSP instance kroA100 produced from the TSP Library, the same rules apply as previously.

Table 6.11: EAS on the kroA100

TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values
Iterations	10	Iterations	10	Iterations	5
Parameter alpha	1	Parameter alpha	1	Parameter alpha	1
Parameter beta	2	Parameter beta	5	Parameter beta	2
ρ	0.5	ρ	0.5	ρ	0.5
Route Cost	23536.74	**Route Cost**	23348.57	**Route Cost**	27776.76
Running Time	0.95 s	**Running Time**	1.26 s	**Running Time**	1 s

A large instance of 1000 cities does not exhibit the same variations as a small one. In every situation and combination of characteristics, the route costs are extraordinarily expensive.

Table 6.12: EAS on the Random simulated data set with 1000 cities

Simulated data set (1000)	Values	Simulated data set (1000)	Values	Simulated data set (1000)	Values
Iterations	10	Iterations	10	Iterations	5
Parameter alpha	1	Parameter alpha	1	Parameter alpha	1
Parameter beta	2	Parameter beta	5	Parameter beta	2
ρ	0.5	ρ	0.5	ρ	0.5
Route Cost	25344.09	**Route Cost**	25278.73	**Route Cost**	25440
Running Time	2190 s	**Running Time**	2159.8 s	**Running Time**	1088 s

For the TSP Library instance, this is not the case. The behavior of pr1002 is similar to that of the small instances, indicating that doing 10 repetitions with parameter beta set to 5 yields to the best results.

Table 6.13: EAS on the pr1002

TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values data set	TSPLib data set (pr1002)	Values
Iterations	10	Iterations	10	Iterations	5
Parameter alpha	1	Parameter alpha	1	Parameter alpha	1
Parameter beta	2	Parameter beta	5	Parameter beta	2
ρ	0.5	ρ	0.5	ρ	0.5
Route Cost	281538.93	**Route Cost**	280346.8	**Route Cost**	281538.9
Running Time	2045.8 s	**Running Time**	1869.3 s	**Running Time**	1069 s

6.3 K - means Clustering Algorithm

Since running times are incredibly rapid, the route cost is only affected by the choice of iterations and clusters.

For the random simulated instance of 100 cities, the scenario of 50 iterations with 10 clusters appears to be the most ideal, as it delivers the lowest route cost.

Table 6.14: K-means clustering algorithm on the Random simulated data set with 100 cities

Simulated data set (100)	Values	Simulated data set (100)	Values	Simulated data set (100)	Values
Iterations	10	Iterations	10	Iterations	50
Number of clusters	10	Number of clusters	2	Number of clusters	10
Route Cost	10581	**Route Cost**	11245	**Route Cost**	10376
Running Time	0.15 s	**Running Time**	0.12 s	**Running Time**	0.15 s

The amount of clusters in the TSP Library should be carefully considered. This is just additional confirmation that the TSP Library instances' properties differ from the random simulated data. The lowest route cost can be attained by using two clusters and less than 50 iterations.

Table 6.15: K-means clustering algorithm on kroA100

TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values
Iterations	10	Iterations	10	Iterations	50
Number of clusters	10	Number of clusters	2	Number of clusters	10
Route Cost	27894	**Route Cost**	26815	**Route Cost**	27786
Running Time	0.12 s	**Running Time**	0.13 s	**Running Time**	0.12 s

The instances with 1000 cities has low enough running times to obtain satisfactory outcomes. For both cases, a selection of less than 100 clusters and less than 50 iterations results in low route costs.

Table 6.16: K-means clustering algorithm on the Random simulated data set with 1000 cities

Simulated data set (1000)	Values	Simulated data set (1000)	Values	Simulated data set (1000)	Values
Iterations	10	Iterations	10	Iterations	50
Number of clusters	100	Number of clusters	2	Number of clusters	100
Route Cost	32327	**Route Cost**	30055	**Route Cost**	32223
Running Time	13.5 s	**Running Time**	13.88 s	**Running Time**	13.79 s

Table 6.17: K-means clustering algorithm on pr1002

TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values
Iterations	10	Iterations	10	Iterations	50
Number of clusters	100	Number of clusters	2	Number of clusters	100
Route Cost	362677	**Route Cost**	321154	**Route Cost**	371600
Running Time	13 s	**Running Time**	14 s	**Running Time**	15 s

Visualization of Routes for the best scenarios of Tables 6.14, 6.15, 6.15 & 6.17 obtained with the K-means clustering algorithm

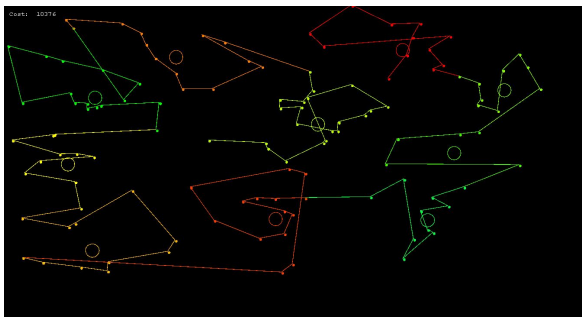


Figure 6.1: Visualization of the 10 clusters and the routes for data set with 100 cities obtained in 50 iterations

Figure 6.2: Visualization of the 2 clusters and the routes for kroA100 obtained in 10 iterations

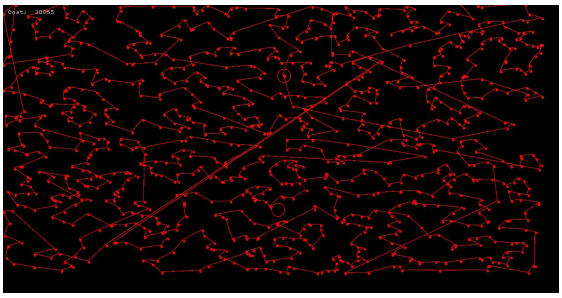


Figure 6.3: Visualization of the 2 clusters and the routes for data set with 1000 cities obtained in 10 iterations

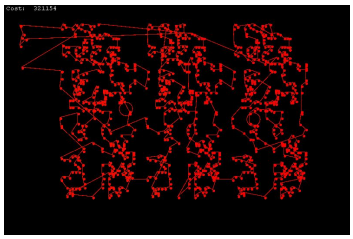


Figure 6.4: Visualization of the 2 clusters and the routes for pr1002 obtained in 10 iterations

6.4 Sensitivity Analysis and Parameter Tuning

The final section of this chapter will describe a variety of computational experiments using the parameters and components of the aforementioned methodologies that affect performance. As a result, the behavior of different approaches will be thoroughly investigated, and an indicator of their robustness will be obtained. This section will also answer questions about what occurs if the settings or other attributes are changed. It will offer us a clear picture of which strategy is the most efficient and the amount of the improvement it can achieve by researching other scenarios and conducting computational trials.

6.4.1 Kohonen's Neural Network

In the approach of the Kohonen Neural Network, it is worth mentioning that the components that affect the algorithm performance and effectiveness are the neighborhood function, the number of iterations, the learning rate and the neighborhood radius.

Neighborhood function and iterations

As demonstrated in the results section 6.1, the Gaussian function should be used in all comparisons. The Kohonen Neural Network produced the best results by using the Gaussian function as a neighborhood function. Now, for the number of repetitions, a wide range was chosen for all the instances, ranging from 10000 to 1000, considering time constraints in mind. It seems logical to investigate what happens if we choose a number of 10 iterations.

In the appendix table 10.1, we can see that with 10 iterations, Kohonen Neural Network produces quite similar solution values especially for the TSPLIB instances, but not for the randomly generated instances. It provides competitive but less attractive outcomes in contrast to versions that utilize a large number of iterations.

Learning Rate

The values that were used for the learning rate in the results section are the ones that were suggested in the explored research. Learning rates can not exceed the value of 1. Iteration by iteration the learning rate is decreasing reaching the final value of 0.01. It will be interesting though to explore how the Neural Network is responding, if the initial values of learning rates are starting from different values in the interval $[0.01, 0.09]$, keeping the neighborhood radius to its initial values, as described in the results in the previous subsections. In the appendix tables 10.2 and 10.4, it can be derived that starting from a lower learning rate than 0.99, meaning starting from 0.7 a better solution can be achieved, while starting from even lower learning rate 0.4 does not offer the desirable results. So the learning rate should be selected carefully, by not selecting the values close to the borders of the interval $[0.01, 0.99]$. For the TSPLIB instances kroA100 and pr1002, it seems that the low value 0.4 for the starting learning rate is preferred (Tables 10.3 & 10.5).

Neighborhood Radius

The values of the neighborhood radius follow a different pattern compared to the learning rate. In the results

presented in section 6.1, the initial value of the neighborhood radius was 50. Executing the iterations of the Kohonen Neural network the neighborhood radius drops to its final value of 0.9. It will be interesting to see what happens if we start from a lower or a higher value for it. Appendix tables 10.6, 10.7, 10.8 & 10.9, dedicated to every instance, provide surprising insights, keeping the values of the learning rates similar to the initials presented in the results section 6.1.

6.4.2 Elitist Ant System

In the approach of the Elitist Ant System, it is worth mentioning that the components that affect the algorithm performance are the parameter beta and the parameter alpha. This algorithm achieves a fast convergence so there is no point exploring the number of iterations. For the results presented in section 6.2 the iterations that are selected can not exceed the 10 repetitions.

Parameter beta

As we saw in 6.2 we have already done a computational experiment by allowing the parameter beta to take its extreme values of 2 and 5. But it is worth seeing the results achieved by allowing beta to taking a value in between. So for beta equal 3 the appendix table 10.10 is presented.

It can be easily seen that using the parameter beta equal to 3, balances out the trade off between iterations, for the simulated instance of 100 cities, while for the simulated data set of 1000 cities reaches almost the best result provided with the parameter selected at 2. For the instances generated from the TSP Library we can see that selecting beta equal to 3, offers better results.

Parameter alpha

Keeping parameter beta equal to 3, as it seems the best choice, appendix tables 10.11, 10.12, 10.13, 10.14, are presented. They offer insights concerning the computational experiments on the parameter alpha. For every instance, parameter alpha can take the values of 0, 1 and 2. In the results presented in 6.2 parameter alpha was set at 1. By selecting the values of 0 or 2 we can derive that similar or even better results are achieved depending on the size and the source of the instance.

6.4.3 K - means Clustering Algorithm

The only tuning of parameters concerning the K - means clustering Algorithm, that can affect its performance is considered to be the clusters. Various values on the number of clusters has been already explored and presented in the results section 6.3.

7. Discussion

The goal of this thesis is to answer with confidence, if machine learning approaches perform better than a metaheuristic for the Traveling Salesman Problem. On several types of TSP cases, the performance of the Kohonen Neural Network is examined and compared to that of the Elitist Ant System and the K-means clustering approach. The performance analysis and computational experiments sections following will explain how the offered ways' performance is tested and which method is the most efficient and preferable method for certain TSP instance characteristics.

Performance analysis

To evaluate the performance of the three proposed methods described in section 4 suitable metrics were chosen to contribute to that analysis. The common main metrics were chosen to be the route cost, since the TSP is solved, the number of iterations and the running times. Certainly, every approach has additional metrics or parameters that are affecting their performance for sure.

As the results in the previous section show, for the random simulated data set for the 100 cities in table 6.3, it can be easily seen that the Kohonen Neural Network offers the best solution in terms of cost. The scenario of the triangular neighborhood function with the 10000 iterations outperforms all the best scenarios of the other two approaches. For the EAS, the best solution is reached by selecting beta equal to 5, ρ equal to 0.5 and α equal to 1 (Table 6.10). The convergence of the EAS algorithm is reached very early, even before the completion of the 10 iterations that were selected. The best case in the K-means clustering algorithm is reached when 50 iterations are executed and the clusters are set to 10 (table 6.14). Even if we increase the number of iterations and clusters the route cost that the Neural Network provided can not be met.

The TSP Library generated instance of 100 cities, kroA100, is subjected to a similar analysis. TSP Library sets are notoriously difficult to solve. The reason for this is that cities are no longer picked at random as they formerly were (Brocki & Korinek, 2007). When compared to the best scenarios of the other two approaches for the identical data set, the situation where the Gaussian neighborhood function is employed in conjunction with 10000 iterations (table 6.4) has no competition. Any tweaking of the settings in the EAS and the K-means clustering method will not yield a better result than the Neural Network.

As predicted, the Kohonen Neural Network outperforms the other two methods for big data sets, with the K - means clustering method having the weakest performance. Table 6.7 shows the best result obtained by repeating the triangular neighborhood function without exceeding 1000 iterations.

TSP Library instances, as previously said, are more difficult to solve, especially in larger data sets with several smaller patterns (Brocki & Korinek, 2007). This did not deter the Kohonen Neural Network, which continues to be the best solution for the TSP library data set of pr1002, offering the cheapest route cost. Similarly, after 10000 iterations of picking the Gaussian function, the best result is obtained (Table 6.8).

To summarize, the Kohonen Neural Network, as mentioned above, provides the sub-optimal solution values

in terms of route cost. Every scenario shown in the Neural Network application's results, appears to be the most effective. The primary drawback of the Kohonen Neural Network is the long running times and vast number of iterations that may be chosen. The clustering algorithms EAS and K-means are incredibly fast. In the case of the EAS, convergence occurs around 10 iterations, which is why this figure was chosen. On the other hand, the number of iterations in the K - means clustering strategy can be raised, to reach the iteration levels of the Kohonen Neural Network, although better solutions were not presented.

Comparison

Apart from the computational experiments, comparison can also be conducted using the available outputs of existing studies on similar or identical instances and the results of the approaches used in the previous sections. Obviously, the simulated data sets are out of this benchmarking since they are randomly generated. For the TSPLIB instances that were used by Yanping Bai, Wendong Zhang & Hongping Hu in 2006, 20 iterations of a similar SOM application were applied to rd100 which contains the same number of cities as kroA100. The best route cost from this application is 18024.3 which is close to the best results obtained from the Kohonen Neural Network in Tables 6.4 & 6.5, although a different TSP instance is in discussion. In the majority of the scenarios, the implemented Kohonen Neural Network outperforms the aforementioned SOM application even with more iterations. Same holds for the instance U1060 which is close in terms of cities' number with pr1002 that was examined in this thesis. Diving into the results, presented by Brocki & Korinek (2007), it can be seen that quite similar results have been obtained for pr1002, as well. Additionally, the use of the Gaussian Neighborhood Function and the number of iterations are major factors that affect the performance of the algorithm in terms of route costs.

Similar studies in terms of results compared to the deployed application were not available for the EAS in the research literature.

On the other hand, Karakoyun (2019) has presented an approach based on a K-Means clustering algorithm. The results obtained for the TSPLIB instance pr1002 were on average around 578583 in terms of route cost. This is a value far from the optimal solution of 259045 for this particular instance. The values that were obtained in our case seem to be closer to the optimal. The table 6.17 offers a clear view, with the best scenario providing a solution with a route cost of 321154.

8. Conclusion

This thesis addresses the question of whether a Kohonen's Neural Network is a more reliable approach than the metaheuristic of the Elitist Ant System and the K - means Clustering Algorithm in order to solve the TSP. To answer it, the performance of these approaches was required by analysing metrics, such as the route cost and the running times. Moreover, implementing these approaches to data sets with different characteristics could offer insights concerning their efficiency and adaptability. Thus, two randomly simulated data sets and two TSPLIB data sets were utilized.

The results in section 6 have shown that the Kohonen Neural Network provided the best solution in terms of route cost for all data sets. Especially, for the data sets that contained 100 cities. For the larger TSP instances we can see that the other two approaches can be competitive, since the solutions that they provided are relatively close to the ones derived from the Kohonen Neural Network. The main drawback of the Kohonen Neural Network approach concerns running times and parametrization. We can conclude that in the majority of the scenarios and more specifically for the large TSP instances, running times are extremely long. On the other hand, K - means Clustering algorithm appeared to be the fastest methodology, but its solutions for all TSP instances do not seem competitive. Thus, the main competitor for the Neural Network is considered to be the EAS. Achieving better running times and in some cases better results, the EAS provided solutions close to the ones obtained from the Neural Network, for both the TSPLIB instances and the large random simulated data sets. This implies that the EAS works better for large data sets but especially for the data sets generated from the TSPLIB.

Regarding the question, which approach is the most efficient a sensitivity analysis was conducted. Taking into consideration that the majority of solutions are really close, it is worth exploring how tuning the parameters of these methodologies would affect the performance. Namely, for the Kohonen Neural Network we can point out that the selection of the neighborhood function (Gaussian or Triangular) as the initialization technique for this method plays a crucial role. The best results have been achieved by selecting GNF as the neighborhood function. Furthermore, different values regarding the number of iterations, the learning rate and the neighborhood radius affect the solution directly. By applying more iterations, the running times are increasing but the route costs are decreasing dramatically. A safe choice for the learning rate would lie in the interval of $[0.01, 0.99]$, while for the neighborhood radius we need to be cautious and not select extreme values creating a very small or a vast neighborhood to be explored.

As far as the EAS, is concerned, parameter alpha and beta are the two components that can affect the performance of the system. Since the EAS converge extremely fast (after 10 iterations) the selection of the aforementioned parameters is important. Parameter beta can take values ranging between the interval $[2, 5]$. Allowing it to take the mean off this interval, we can see that the EAS can provide slightly better results than before. On the other hand, we can easily observe that letting parameter alpha taking the values of 0,

1 and 2 offers no flexibility, since there are not any notable fluctuations on the results.

The evaluation has also shown that the only factor that affects the K - means Clustering algorithm is considered to be the number of clusters. In the majority of the scenarios, decreasing the number of clusters surprisingly offers better results, but unsuccessfully never reach the solution level obtained by the other two approaches.

All in all, Kohonen Neural Network has been proved to be the most effective and adaptable technique for resolving various TSP instances. Although the EAS and the K - means clustering algorithm provided fair outcomes, the Kohonen Neural Network outperformed them.

Future and further research should be focused on this study's limitations. First, a variety of data set types were intended to be addressed. We have distinguished them by size (small or large, 100 or 1000 cities) and by source (randomly simulated or TSPLIB generated). It would be interesting to see how the aforementioned approaches behave in the world of "big data" when extremely large data sets are employed. Moreover, the TSP instances that were used in this study are symmetric, so another aspect to be explored is the use of asymmetric TSP instances. Obviously, the machine learning methods will behave differently. Secondly, a lot of parameters used in these approaches can be changed as the methods offer flexibility. More specifically, for the Kohonen Neural Network it would be interesting to see another neighborhood function apart from the implemented ones. An example of such, could be the Bubble function. Similarly, an improvement for the Elitist Ant System, can be considered the use of another local search technique rather than the Hill Climbing. Last but not least, in this study, K - means Clustering Algorithm is considered to be an extension of the 2-opt algorithm. Hence, selecting another algorithm to start and solve the TSP by enhancing it with the K - means clustering algorithm would be plausible.

9. References

[1] Anmar Abuhamdah. Adaptive elitist-ant system for solving combinatorial optimization problems. *Applied Soft Computing*, 105:107– 293, 2021.

[2] Altinel IK. Aras N, Oommen BJ. Kohonen network incorporating explicit statistics and its application to the traveling salesman problem. *Neural Networks*, 1999.

[3] Yanping Bai, Wendong Zhang, and Hongping Hu. An efficient growing ring som and its application to tsp. 2006.

[4] Fernando Bação, Victor Lobo, and M. Painho. Self-organizing maps as substitutes for k-means clustering. *Lecture Notes in Computer Science*, 3516:476–483, 05 2005.

[5] Christopher M. Bishop, Markus Svensén, and Christopher K.I. Williams. Developments of the generative topographic mapping. *Neurocomputing*, 21(1):203–224, 1998.

[6] Łukasz Brocki and Danijel Koržinek. Kohonen self-organizing map for the traveling salesperson problem. pages 116–119, 01 2007.

[7] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. A new rank based version of the ant system. a computational study. 1997.

[8] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[9] Vojislav TOMIĆ Sonja STOJKOVIĆ Danijel MARKOVIĆ, Miloš MADIĆ. Solving travelling salesman problem by use of kohonen self-organizing maps. 2012.

[10] Rafal Dlugosz, Marta Kolasa, and Witold Pedrycz. Programmable triangular neighborhood functions of kohonen self-organizing maps realized in cmos technology. 05 2010.

[11] M. Dorigo. Optimization, learning and natural algorithms [in italian]. *Dipartimento di Elettronica, Politecnico di Milano, Milan*, page PhD thesis, 1992.

[12] Maniezzo V. Colorni A. Dorigo, M. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics—Part B*, page 29–41, 1996.

[13] F. Favata and Richard Walker. A study of the application of kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics*, 64:463–468, 04 1991.

[14] B. Fritzke. Growing cell structures—a self-organizing network for unsupervised and supervised learning. *Neural Netw.,vol.7*, 7:9–13, 01 1994.

[15] B. Fritzke. Growing cell structures—a self-organizing network for unsupervised and supervised learning. *Neural Netw.,vol.7*, 7:9–13, 01 1994.

[16] Ian P. Gent and Toby Walsh. The tsp phase transition. *Artificial Intelligence*, 88(1):349–358, 1996.

[17] Jaakko Hollmen. Self-organizing map (som). *Aalto University*, 03 1996.

[18] John Hooker. Testing heuristics: We have it all wrong. journal of heuristics 1(1): 33-42. *Journal of Heuristics*, 1:33–42, 09 1995.

[19] Tank D.W. Hopfield, J.J. "neural" computation of decisions in optimization problems. *Biol. Cybern*, 52:141–152, 1985.

[20] E. Alhoniemi J. Vesanto, J. Himberg and J. Parhankangas. Som toolbox for matlab 5. *Laboratory of Computer and Information Science (CIS).*, 2000.

[21] Ameera Jaradat, Bara'a Matalkah, and Waed Aldiabat. Solving traveling salesman problem using firefly algorithm and k-means clustering. pages 586–589, 04 2019.

[22] Fort JC. Solving combinatorial problem via self-organizing process: an application of the kohonen algorithm to the traveling sales-man problem. *Biol Cyber*, 59, 1988.

[23] Nassirou Lo Jeremiah Ishaya, Abdullahi Ibrahim. A comparative analysis of the travelling salesman problem: Exact and machine learning techniques. page 23 – 37, 09 2019.

[24] Timo Honkela Jorma Laaksonen. Advances in self-organizing maps. *Conference proceedings,8th International Workshop, WSOM 2011, Espoo, Finland*, 06 2011.

[25] Risto Miikkulainen José C. Príncipe. Advances in self-organizing maps. *Conference proceedings, 7th International Workshop, WSOM 2009, St. Augustine, Florida*, 06 2009.

[26] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. Chapter 4 the traveling salesman problem. 7:225–330, 1995.

[27] Murat Karakoyun. A new approach based on k-means clustering and shuffled frog leaping algorithm to solve travelling salesman problem. *International Symposium on Innovative Technologies in Engineering and Science 22-24 November 2019 (ISITES2019 SanliUrfa - Turkey)*, 7, 2019.

[28] Rahul Karmakar, Ramkrishna Mitra, Avishek Dey, Vaswati Chakraborty, and Arabinda Nayak. Solving tsp using improved elitist ant system based on improved pheromone strategy and dynamic candidate list. pages 8–15, 2016.

[29] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43*, page 59–69, 1982.

[30] Teuvo Kohonen. Self-organizing maps. *Springer Berlin, Heidelberg*, 30, 2001.

[31] Usha A. Kumar and Yuvnish Dhamija. Comparative analysis of som neural network with k-means clustering algorithm. *2010 IEEE International Conference on Management of Innovation & Technology*, pages 55–59, 2010.

[32] Bert La Maire and Valeri Mladenov. Comparison of neural networks for solving the travelling salesman problem. *11th Symposium on Neural Network Applications in Electrical Engineering,NEUREL 2012 - Proceedings*, 09 2012.

[33] Bert La Maire and Valeri Mladenov. Comparison of neural networks for solving the travelling salesman problem. *11th Symposium on Neural Network Applications in Electrical Engineering,NEUREL 2012 - Proceedings*, 09 2012.

[34] Majd Latah. Solving multiple tsp problem by k-means and crossover based modified aco algorithm. *International journal of engineering research and technology*, 5, 2016.

[35] K. Kyuma M. Takashi and E. Funada. 10000 cell placement optimization using a self-organizing map. *Proceedings International Joint Conference on Neural Networks 3*, page 417– 2420, 1993.

[36] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 5, 1967.

[37] Quanta Magazine. Computer scientists break traveling salesperson record. 2020.

[38] Thomas Stützle Marco Dorigo. Ant colony optimization, "a bradford book.". *Massachusetts Institute of Technology*, 64, 2004.

[39] Thomas Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. *Springer London*, pages 427–434, 1993.

[40] Forgáč R Mokriš. Decreasing the feature space dimension by kohonen self-orgaizing maps. *2nd Slovakian – Hungarian Joint Symposium on Applied Machine Intelligence, Herľany, Slovakia.*, 2004.

[41] A. K. Qin Tabinda Sarwar Nasrin Sultana, Jeffrey Chan. Learning to optimise general tsp instances. 11 2020.

[42] Kaski S. Oja, M. and T. Kohonen. Bibliography of self-organizing map (som). *Addendum. Neural Computing Surveys, 3*, 1998-2001:1–156, 2003.

[43] Hanqiang Cao Olivier Rukundo. Advances on image interpolation based on ant colony algorithm. 2021.

[44] K. Papadimitriou, C.H.; Steiglitz. Combinatorial optimization: algorithms and complexity. *Mineola, NY: Dover,*, pages 308–309, 1998.

[45] Simon Parsons. Ant colony optimization by marco dorigo and thomas stützle. *Knowledge Eng. Review*, 20:92–93, 03 2005.

[46] Honkela T. Pöllä, M. and T. Kohonen. Bibliography of self-organizing map (som) papers: 2002-2005. *Addendum. Neural Computing Surveys*, page 1742–1759, 2007.

[47] R.Dhanalaksmi P. Parthiban R.Nallusamy, K.Duraiswamy. Optimization of non-linear multiple traveling salesman problem using k-means clustering, shrink wrap algorithm and meta-heuristics. *Department of CSE, K.S.R.College of Technology Tiruchengode-637215, Tamilnadu, India*, 171-177, 2010.

[48] Kate Smith-Miles. Neural networks for combinatorial optimization: A review of more than a decade of research. *INFORMS Journal on Computing*, 11:15–34, 02 1999.

[49] Enkawa T. Somhom S, Modares A. Competition-based neural network for the multiple traveling salesman problem with minmax objective. *Comput Oper Res*, 26, 1999.

[50] Thomas Stützle and Holger Hoos. Max-min ant system. 16, 11 1999.

[51] Joab O. Lima Sueli A. Mingoti. Comparing som neural network with fuzzy c-means, k-means and traditional hierarchical clustering algorithms. *European Journal of Operational Research 174 (2006)*, page 1742–1759, 07 2005.

[52] A. I. Vakhutinsky and B. L. Golden. Solving vehicle routing problems using elastic nets. *Proceedings IEEE International Conference on Neural Networks 7*, page 4535–4540, 1994.

[53] Marc Van Hulle. Faithful representations and topographic maps from distortion based to information based self-organization. 01 2000.

[54] Mark Velednitsky. Short combinatorial proof that the dfj polytope is contained in the mtz polytope for the asymmetric traveling salesman problem. *Operations Research Letters*, 45 (4):323–324, 2017.

[55] G. V. Wilson and G. S. Pawley. On the stability of the tsp algorithm of hopfield and tank. *Biological Cybernetics 58*, page 63–70, 1988.

10. Appendix

10.1 Computationl Experiments Tables

Table 10.1: Computational experiment for all data sets using the Kohonen Neural Network with 10 iterations

Simulated data set of 100	Values	kroA100	Values	Simulated data set of 1000	Values	pr1002	Values
Iterations	10	Iterations	10	Iterations	10	Iterations	10
Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99	Learning Rate	0.99
Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50	Neighborhood Radius	50
Learning Rate Final	0.01	Learning Rate Final	0.01	Learning Rate Final	0.01	Learning Rate Final	0.01
Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9
Route Cost	15322.66	**Route Cost**	29909.22	**Route Cost**	53863.13	**Route Cost**	275079.82
Running Time	0.3 s s	**Running Time**	0.31 s	**Running Time**	7.46 s	**Running Time**	3.75 s

Table 10.2: Kohonen Neural Network with different learning rate starting values for the simulated data set of 100 cities

Simulated data set of 100	Values	Simulated data set of 100	Values	Simulated data set of 100	Values
Iterations	1000	Iterations	1000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.7	Learning Rate	0.4
Learning Rate Final	0.01	Learning Rate Final	0.01	Learning Rate Final	0.01
Route Cost	28499.21	**Route Cost**	11413.12	**Route Cost**	12650.38
Running Time	0.76 s	**Running Time**	3.24 s	**Running Time**	5.16 s

Table 10.3: Kohonen Neural Network with different learning rate starting values for kroA100

TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values
Iterations	1000	Iterations	1000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.7	Learning Rate	0.4
Learning Rate Final	0.01	Learning Rate Final	0.01	Learning Rate Final	0.01
Route Cost	81846.8	**Route Cost**	51473.40	**Route Cost**	34583.66
Running Time	1.2 s	**Running Time**	2.7 s	**Running Time**	2.7 s

Table 10.4: Kohonen Neural Network with different learning rate starting values for the simulated data set of 1000 cities

Simulated data of 1000	Values	Simulated data of 1000	Values	Simulated data of 1000	Values
Iterations	1000	Iterations	1000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.7	Learning Rate	0.4
Learning Rate Final	0.01	Learning Rate Final	0.01	Learning Rate Final	0.01
Route Cost	60732.30	**Route Cost**	35163.61	**Route Cost**	35462.71
Running Time	54.49 s	**Running Time**	237.25 s	**Running Time**	230.46 s

Table 10.5: Kohonen Neural Network with different learning rate starting values for pr1002

TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values
Iterations	1000	Iterations	1000	Iterations	1000
Learning Rate	0.99	Learning Rate	0.7	Learning Rate	0.4
Learning Rate Final	0.01	Learning Rate Final	0.01	Learning Rate Final	0.01
Route Cost	1468329.97	**Route Cost**	287314.01	**Route Cost**	275563.31
Running Time	38.5 s	**Running Time**	180.7 s	**Running Time**	178.74 s

Table 10.6: Kohonen Neural Network with different neighborhood radius starting values for the simulated data set of 100

Simulated data set of 100	Values	Simulated data set of 100	Values	Simulated data set of 100	Values
Iterations	1000	Iterations	1000	Iterations	1000
Neighborhood Radius	20	Neighborhood Radius	50	Neighborhood Radius	100
Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9
Route Cost	11492.82	**Route Cost**	28499.21	**Route Cost**	9409.06 1
Running Time	2.8 s	**Running Time**	0.76 s	**Running Time**	2.68 s

Table 10.7: Kohonen Neural Network with different neighborhood radius starting values for kroA100

TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values
Iterations	1000	Iterations	1000	Iterations	1000
Neighborhood Radius	20	Neighborhood Radius	50	Neighborhood Radius	100
Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9
Route Cost	48600.09	**Route Cost**	81846.8	**Route Cost**	46946.18
Running Time	2.98 s	**Running Time**	1.2 s	**Running Time**	2.88 s

Table 10.8: Kohonen Neural Network with different neighborhood radius starting values for simulated data set of 1000

Simulated data set of 1000	Values	Simulated data set of 1000	Values	Simulated data set of 1000	Values
Iterations	1000	Iterations	1000	Iterations	1000
Neighborhood Radius	20	Neighborhood Radius	50	Neighborhood Radius	100
Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9
Route Cost	6134.05	**Route Cost**	60732.30	**Route Cost**	45244.89
Running Time	156.9 s	**Running Time**	54.49 s	**Running Time**	221.4 s

Table 10.9: Kohonen Neural Network with different neighborhood radius starting values for pr1002

TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values
Iterations	1000	Iterations	1000	Iterations	1000
Neighborhood Radius	20	Neighborhood Radius	50	Neighborhood Radius	100
Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9	Neighborhood Radius Final	0.9
Route Cost	290804.1	**Route Cost**	1468329.97	**Route Cost**	339550.11
Running Time	142.65 s	**Running Time**	38.5 s	**Running Time**	179.8 s

Table 10.10: EAS for all data sets with parameter beta equal to 3

Simulated data set of 100	Values	kroA100	Values	Simulated data set of 1000	Values	pr1002	Values
Iterations	10	Iterations	10	Iterations	10	Iterations	10
Parameter alpha	1	Parameter alpha	1	Parameter alpha	1	Parameter alpha	1
ρ	0.5	ρ	0.5	ρ	0.5	ρ	0.5
Parameter beta	3	Parameter beta	3	Parameter beta	3	Parameter beta	3
Route Cost	8228.42	**Route Cost**	22005.25	**Route Cost**	26470.65	**Route Cost**	288401.74
Running Time	1.72 s	**Running Time**	1.03 s	**Running Time**	1.7 s	**Running Time**	9 s

Table 10.11: EAS for the simulated data set of 100 with different values of parameter alpha

Simulated data set of 100	Values	Simulated data set of 100	Values	Simulated data set of 1000	Values
Iterations	10	Iterations	10	Iterations	10
Parameter alpha	1	Parameter alpha	2	Parameter alpha	0
ρ	0.5	ρ	0.5	ρ	0.5
Route Cost	8141.27	**Route Cost**	8303.35	**Route Cost**	8407.11
Running Time	4.11 s	**Running Time**	10.31 s	**Running Time**	9 sec

Table 10.12: EAS for kroA100 with different values of parameter alpha

TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values	TSPLib data set (kroA100)	Values
Iterations	10	Iterations	10	Iterations	10
Parameter alpha	1	Parameter alpha	2	Parameter alpha	0
ρ	0.5	ρ	0.5	ρ	0.5
Route Cost	23537.74	**Route Cost**	23992.31	**Route Cost**	22232.48
Running Time	1.83 s	**Running Time**	4.6 s	**Running Time**	3.6 s

Table 10.13: EAS for the simulated data set of 1000 with different values of parameter alpha

Simulated data set of 1000	Values	Simulated data set of 1000	Values	Simulated data set of 1000	Values
Iterations	10	Iterations	10	Iterations	10
Parameter alpha	1	Parameter alpha	2	Parameter alpha	0
ρ	0.5	ρ	0.5	ρ	0.5
Route Cost	25344.09	**Route Cost**	26308.43	**Route Cost**	26141.14
Running Time	2190 s	**Running Time**	251 s	**Running Time**	148 s

Table 10.14: EAS for pr1002 with different values of the parameter alpha

TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values	TSPLib data set (pr1002)	Values
Iterations	10	Iterations	10	Iterations	10
Parameter alpha	1	Parameter alpha	2	Parameter alpha	0
ρ	0.5	ρ	0.5	ρ	0.5
Route Cost	281538.93	**Route Cost**	296920.67	**Route Cost**	282999.66
Running Time	2045.8 s	**Running Time**	494 s	**Running Time**	392 s

10.2 Random Simulated Data set of 100 cities

Table 10.15: Random Simulated data set of 100 cities

100

840.0 263.0 540.0 439.0

236.0 684.0 119.0 818.0

249.0 288.0 715.0 70.0

34.0 933.0 556.0 621.0

38.0 620.0 15.0 499.0

571.0 314.0 153.0 363.0

875.0 369.0 618.0 433.0
877.0 468.0 564.0 105.0
130.0 360.0 317.0 250.0
850.0 672.0 590.0 153.0
708.0 375.0 642.0 294.0
153.0 383.0 952.0 591.0
560.0 343.0 785.0 163.0
616.0 465.0 755.0 119.0
122.0 325.0 991.0 308.0
549.0 377.0 366.0 110.0
877.0 288.0 739.0 642.0
676.0 389.0 922.0 241.0
667.0 406.0 811.0 220.0
107.0 204.0 842.0 477.0
32.0 786.0 483.0 508.0
113.0 52.0 590.0 508.0
71.0 950.0 517.0 760.0
221.0 351.0 487.0 528.0
556.0 713.0 500.0 714.0
170.0 377.0 757.0 864.0
385.0 309.0 882.0 348.0
344.0 673.0 131.0 653.0
92.0 479.0 178.0 371.0
262.0 145.0 252.0 103.0
133.0 548.0 912.0 391.0
451.0 204.0 407.0 137.0
526.0 605.0 316.0 868.0
533.0 772.0 518.0 845.0
131.0 807.0 605.0 464.0
192.0 948.0 533.0 957.0
744.0 185.0 738.0 937.0
520.0 576.0 821.0 743.0
594.0 499.0 76.0 186.0
466.0 712.0 440.0 724.0
102.0 551.0 967.0 226.0

388.0 795.0 191.0 982.0
141.0 751.0 127.0 82.0
509.0 349.0 510.0 380.0
769.0 67.0 287.0 361.0
791.0 711.0 217.0 67.0
329.0 308.0 181.0 238.0
89.0 480.0 705.0 586.0
824.0 114.0 770.0 760.0
553.0 357.0 279.0 194.0