CTU

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Cybernetics

Bachelor's thesis

Solving the Close Enough Multi Traveling Salesman Problem with the Hopfield Neural Network

Kristián Domažlický

Date May 2024

Supervisor: Ing. Jindřiška Deckerová

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Domažlický Kristián** Personal ID number: **498962**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Solving the Close Enough Multi Traveling Salesman Problem with the Hopfield Neural Network

Bachelor’s thesis title in Czech:

Řešení problému obchodního cestujícího se spojitým okolím a více cestujícími pomocí Hopfieldovy neuronové sítě

Guidelines:

1. Familiarize yourself with multi-vehicle routing problems such as the Multi Traveling Salesman Problem (mTSP) [1] and Multi TSP with Neighborhoods (mTSPN) [2], as well as single-vehicle problems such as the Close Enough TSP (CETSP) [3].
2. Familiarize yourself with the solution of TSP using the Hopfield Neural Network (HNN) [4].
3. Propose an energy function (a formulation) of the Close Enough mTSP (CEmTSP) to use in the HNN and adapt HNN accordingly.
4. Evaluate the proposed HNN and compare it with the existing approaches, such as [5].

Bibliography / sources:

[1] Cheikhrouhou, Omar, and Ines Khoufi. "A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy." Computer Science Review 40 (2021): 100369.
[2] He, Pengfei, and Jin-Kao Hao. "Hybrid search with neighborhood reduction for the multiple traveling salesman problem." Computers & Operations Research 142 (2022): 105726.
[3] Gulczynski, Damon J., Jeffrey W. Heath, and Carter C. Price. "The close enough traveling salesman problem: A discussion of several heuristics." Perspectives in Operations Research: Papers in Honor of Saul Gass’ 80 th Birthday (2006): 271-283.
[4] Li, Rong, Junfei Qiao, and Wenjing Li. "A modified hopfield neural network for solving TSP problem." In 2016 12th World Congress on Intelligent Control and Automation (WCICA), pp. 1775-1780. IEEE, 2016.
[5] Faigl, Jan. "An application of self-organizing map for multirobot multigoal path planning with minmax objective." Computational intelligence and neuroscience 2016 (2016).

Name and workplace of bachelor’s thesis supervisor:

Ing. Jindřiška Deckerová Department of Computer Science FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **22.01.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Jindřiška Deckerová	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Author statement for undergraduate work

I declare that the presented work was developed independently and that I have listed all sources of the information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

Prague, 24 May 2024

. . . . . .

Kristián Domažlický

Acknowledgment

I would like to thank Ing. Jindřiška Deckerová for supervising my thesis. As well would also like to thank coffee for keeping me awake and writing this thesis. As a note for translation of the Czech abstract DeepL was used.

Abstrakt

Tato práce se zaměřuje na zavedení, implementaci a vyhodnocení problému close enough multi traveling salesman problem (CEmTSP). Problém spočívá v optimalizaci celkové ujeté délky všech vozidel, přičemž každé místo je navštíveno jednou. CEmTSP je zobecněním problému cestujícího obchodníka (TSP). Je motivován sběrem dat z flotily bezpilotních letadel. V této práci je navrženo použít k řešení CEmTSP Hopfieldovu neuronovou síť (HNN). HNN je typ plně propojené neuronové sítě, která využívá gradientní sestup k nalezení optimálního řešení podle dané energetické funkce. Tato energetická funkce aproximuje studovaný problém a je navržena v tomto článku. Pokud jde o vyhodnocení HNN. S navrženou energetickou funkcí CEmTSP bylo prokázáno, že HNN poskytuje platné řešení. pro CEmTSP, ale kvalita není konkurenceschopná ve srovnání s jinými řešiteli. Lze ji zlepšit pomocí optimalizace. I když se však řešení zlepší, neznamená to, že bude konkurenceschopné. To nám ukazuje, že HNN není optimální pro použití jako řešení pro CEmTSP.

Klíčova slova: Hopfieldova neuronová síť, Problém Obchodního cestujícího se spojitým okolím a více cestujícími

Abstract

This thesis focuses on the introduction, implementation, and evaluation of the Close Enough Multi-Traveling Salesman Problem (CEmTSP). The problem is to optimize the total length traveled of all vehicles, while visiting every location once. CEmTSP is a generalization of the Traveling Salesman Problem (TSP). This is motivated by data collection from a fleet of UAVs. In this thesis, it's proposed to employ a Hopfield Neural Network (HNN) to solve CEmTSP. HNN is a type of fully connected Neural Network that uses gradient descent to find the optimal solution according to a given energy function. The energy function approximates the studied problem and is proposed by this paper. Regarding the evaluation of HNN. With the proposed CEmTSP energy function it has been shown that HNN provides a valid solution. for CEmTSP, but the quality is not competitive in comparison with other solvers. It can be improved with optimization. However, even if it improves the solution, it does not make it competitive. This shows us that the HNN is suboptimal to be used as a solution for CEmTSP.

Keywords: Hopfield Neural Network, Close Enough Multi Traveling Salesman Problem

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

HNN	Hopfiel Neural Network
NN	Neural Network
TSP	Traveling Salesman Problem
CEmTSP	Close Enought Multi Traveling Salesman Problem
mTSP	Multi Traveling Salesman Problem
CETSP	Close Enought Traveling Salesman Problem
SOM	Self Organizing Maps
UAV	Unmaned Areal Vehicle
euc	Euclidian distance

Notation

N	number of locations
M	number of traveling salesman (Robots)
δ	Radius of neighborhood
Σ	Sequence of Targets to be visited
σ	index of location that is being visited
S	Neiborhood of Location
P	Sequence of waypoints to be visited
p	Waypoint
s	Coords of location
V	State Matrix of HNN
U	Input Matrix of HNN
E	Energyfunction
t	Step size of derivation
L	length of tour
a	hyper parameter for E_a term
b	hyper parameter for E_b term
c	hyper parameter for E_c term
d	hyper parameter for E_d term
f	hyper parameter for E_f term
Δ	min cost in cost matrix
∇	max cost in cost matrix
$dist$	cost matrix

Chapter 1
Introduction

This thesis is motivated by data collection from a fleet of UAVs, that can collect data remotely through a camera or wireless network. This can be modeled as an advanced extension of the Traveling Salesman Problem (TSP) called the Close Enough Multi Traveling Problem (CEmTSP). The thesis is focused on solving the CEmTSP with a variant of a Neural Network (NN) called Hopfield Neural Network (HNN), as HNN has already been proven to be able to solve TSP [2, 3].

The TSP is one of the classic problems of computer science. It is a task to find the shortest path between N locations [4] . Due to the number of possible combinations, that is $\frac{(N-1)}{2}!$ the TSP is an NP-hard problem. The TSP has been solved in many ways such as Hopfield Neural Networks (HNN) [2] , K-opt [5], branch and bound [5], Self Organizing maps [6]. TSP has many applications such as drilling of printed circuit boards, genome sequencing [5].

In problems such as crew scheduling, bus routing, and printing press scheduling [5], it is more suitable to use multiple vehicles for a given problem. Therefore formulation of these problems as a variant of the TSP called the Multi Traveling Salesman Problem (mTSP) would be used. mTSP differs from TSP by extending the number of vehicles deployed to visit all locations. There are several ways to prioritize the solution such as, (i) *Min-sum*, where the total cost of all paths of all vehicles is the final cost, (ii) Min-max, where the final cost is the cost of the most expensive path, (iii) balanced workload between vehicles, where the amount of locations is to be the same between the vehicles [7] . This thesis will focus on the *Min-sum* variant of mTSP.

Another variant of the TSP focused on in this thesis is the Close Enough Traveling Salesman Problem (CETSP). CETSP is motivated by data collection with instruments that can collect data from a distance. These scenarios might be surveillance of complexes using drones, and disaster relief coordination [7]. The ”Close Enough” modification is the addition of disk-shaped neighborhoods, changing it so the location is visited if the vehicle visits the neighborhood instead of the center itself.

In this thesis, the CEmTSP with disk-shaped neighborhoods is solved by the HNN. HNN is a recurrent neural network with a full feedback structure that relies on the proposed energy function to find the global minimum by gradient descent [8]. The gradient descent makes the HNN susceptible to local minima, this can be solved through re-initialization of the neural network, or a change of weights in the neural network, so it escapes the local minimum. A motivation to solve CEmTSP through HNN is possible parallelization and the possibility of making custom-made hardware to solve HNN, which can speed up HNN significantly [9].

This thesis is structured as follows. The existing approaches to solving TSP-type problems and background to HNN is in Chapter 2 . Formulation of mTSP and CEmTSP is in Chapter 3. The HNN, on which the thesis solution is based, and a general deeper look into HNN is in Chapter 4. The thesis modifications to HNN are in Chapter 5. Results and comparison with competing algorithms are in Chapter 6. The conclusion is in Chapter 7.

Chapter 2
Related Work

This chapter briefly summarizes existing approaches to TSP, mTSP, and CETSP, since no direct methods exist to solve CEMTSPs exist, to the best of the author's knowledge. As this thesis builds on existing knowledge of Neural Networks, this section also summarises them.

TSP is one of the classical problems of computer science. This problem even preceded the invention of computers as a German business manual called "Geschäften gewiss zu sein—Von einem alten Commis-Voyageu" was released in the year 1832, which was interested in the shortest path between few german Cities [4]. One of the first computationally solved problems was in the year 1954 a path through all 49 capitals of the continental USA [10]. In contrast to the year 2006, when the Concorde algorithm was able to solve problems with 85 900 locations, making the record of solving practical applications optimistic [11].
As solutions for TSP have improved more approaches and extensions have been formulated. The interest in TSP is not only due to the direct solutions of TSP, but as TSP is an NP-complete problem a solution to TSP problem could solve other NP-complete problems [11] .

2.1 Traveling Salesman Problem

Regarding solutions for TSP, they can be divided into three types of approaches: heuristic, approximate, and exact [5].

Heuristic algorithms, their solution is to find a heuristic with good empirical performance [12]. Some of these algorithms are K-opt [5], Closest Neighbor [5]. By only giving a heuristic answer their computational time can be lower than other methods but their accuracy can not be ensured [11]. Expected accuracy depends on the type of heuristic from 25% with the Closest Neighbor heuristic to 2% with the Lin-Kernighan heuristic. Lin-Kernighan is a generalization of K-opt where differing k values are used, due to its complexity $O(n^k)$, mainly the 2-opt heuristic, which by itself can achieve 5% [5] accuracy, is used, or less often 3-opt which accuracy by its own can achieve 3% [5] accuracy [13]. Even though there is no accuracy guarantee their speed is used as an upper bound for the optimal value and if the non-optimal solution would suffice [11].

Exact Algorithms try to find exact solutions, their complexity depends on the type of solution. An example of these algorithms is brute-force, these algorithms try every combination resulting in $O(`n!)$, which makes this impractical for larger TSP, Dynamic programming, tries to use already calculated paths to prune un-optimal paths, a Hald-Karp algorithm using this method achieves $O(n^2 2^n)$, making them capable to calculate up to 60 locations paths, but any larger are impractical. The most capable approach of exact algorithms is Branch and Cut that was able to solve 85,900 Location problem when using Concorde algorithm [14], but this took 136 CPU-years [15].

Due to SOM algorithm will be used as a baseline solution and 2-opt will be used as an optimalization for found paths. These algorithms are briefly described.

2-opt algorithm takes a valid solution of TSP, which might be random, and takes 2 edges, then it replaces those edges with different ones that yield lower travel cost. This is repeated until no improvements can be found [16]. This tour is called 2-optimal, for a K-optimal tour it takes K edges instead of 2. For best results, the starting solution should be a greedy tour [5].

Self-organizing maps (SOM), are two layer Neural networks where the first layer is the input and the second layer is the solution where the weights are R^d, d is the dimension of the problem. In every

iteration, random neurons are generated, then winning neurons are decided and the rest are forgotten, it checks if the new path is better than the last best and the algorithm is rerun [6, 17]. Generation of neurons can be adapted through heuristics such as GENIUS, which is trying to generate the best neurons possible. In this work, the SOM solution is taken as a reference solution for CEMTSP [18].

2.2 Multi Traveling Salesman Problem

mTSP is an umbrella of problems that has many variants that will give us differing solutions, from Min-max, Min-sum, balanced task allocation, and maximum vehicles used, with any combination of those [7, 19] .

In this thesis, we focus on the Min-sum problem, this sub-problem objective is to find the shortest distance of all paths while visiting every location only once. As this problem is almost the same as TSP, however, some additional constraints must be implemented, for example, Multi Depot mTSP, where this problem can be translated into TSP where the path is not a singular closed cycle but multiple cycles starting and ending in a location that is designated as depot [20], or giving each vehicle minimum or maximum amount of locations that need to be visited.

Solutions to mTSP can be divided into exact and heuristic solutions as well. Among popular approaches to solving mTSPs are Genetic Algorithms, Ant Colony Optimization, or Self-Organizing Maps [7, 17] .

2.3 Close Enough Traveling Salesman Problem

CETSP is motivated by practical scenarios, such as remote data collection of gas and electricity consumption, collection of data from underwater vehicles on the ocean floor, and forest fire detection by use of UAVs [21] . The difference between TSP and CETSP is that every location has a disk-shaped neighborhood to visit.

CETSP was first been proposed by D. Gulczynski [22] and the author provided six heuristics, these heuristics assume that all locations have an equal disc-shaped radius of δ [23]. As CETSP is a newer modification of TSP, there are only a few exact approaches but among them, there are branch and bound algorithms by Coutinho, that can solve instances with hundreds of neighborhoods, but take four hours or more, and Integer-Programing based on Behdani and Smith [24]. As exact algorithms for CETSPs take significantly longer than heuristic and approximate approaches are used more like Super nodes heruistic [22] , genetic algorithms or SOM [19, 21, 24].

2.4 Close Enough Multi-Traveling Salesman Problem

CEmTSP is motivated by the combination of CETSP and mTSP, which could solve CETSP problems on larger scales and with multiple vehicles being used simultaneously saving time. As this problem is a combination of multiple modifications of TSP, little research has been done on this problem, to the best of the author's knowledge. This is quite limiting for this thesis as a comparison of accuracy is needed to see how well HNN solves this problem. This will be solved through two solvers that answer this problem but might not be the most accurate, SOM from paper [18] with omission of the Dubins problem, and greedy K-means heuristic from paper [25]. These should give us benchmarks for HNN to beat and be evaluated against.

2.5 Neural Networks

Neural Networks (NN) are inspired by neurons in the human brain, by creating connections between elements with differing weights to create an interpretation of neurons. Their main advantage is versatility, that there does not need to be a specific algorithm to solve a given problem, though it might not be the most efficient way to solve a given problem. Another advantage is that NNs can be computed in parallel making them suited to calculate real-time problems [9, 26]. The increasing popularity of NNs in past years has led NNs to be used in many problems.

Regarding HNN specifically, HNN was first proposed by J.J. Hopfield [27], the idea is to improve the simulation of biological neurons. As HNN requires energy to function, which is an approximation of the problem given to HNN. This leads to specific advantages and drawbacks of HNNs compared to other NNs. Among the drawbacks are, that the chosen energy function minima do not guarantee an optimal solution, artificial Hyper-parameters are difficult to balance due to their sensitivity, the HNNs sometimes can reach inadmissible solutions and HNN is likely to reach local minima of the energy function [3, 9]. On the other hand, there is a major advantage that makes HNNs an interesting area of study, and that is the fact that HNNs can be hardware-optimized, making the computation of solutions much faster than in soft-computing [9].

The problems in which HNN gives better results than NN are Mathematical Programming problems like Linear Programming and Non-Linear Programming. The problems solved by HNNs include and are not limited to TSP [2, 3], and facial recognition [28].

Chapter 3
Problem Statement

In this section, the definition of the studied problem is provided. CEmTSP is an extension of TSP, where every location has a non-zero neighborhood, and multiple vehicles are dispatched to solve the problem. As the solution for multiple vehicles can be differently prioritized this thesis focuses on the Min-sum definition.

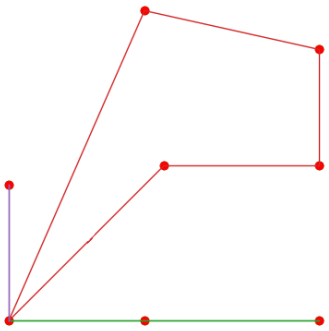

Figure 3.1: mTSP

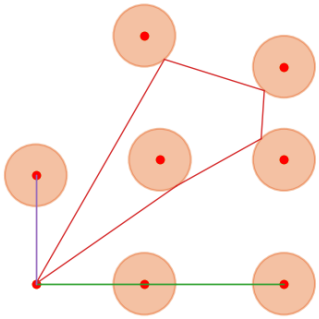

Figure 3.2: CEmTSP

3.1 Multi Traveling Salesman Problem

Having a set of n sensor locations s_i, each defined by its position $\mathbf{s_i} \in \mathbf{R}^2$, the mTSP goal is to determine a set of m paths such that the sum cost of paths is minimal, and all the locations s_i are visited once. Besides the depot location s_1 where all the paths start and end. The paths are defined as a set $\Sigma = \{\Sigma^1 \dots \Sigma^m\}$, where each element is a path $\Sigma^i = (\sigma_1^i \dots \sigma_{k_i}^i)$, and the path is a sequence of locations that are to be gone through. Each vehicle needs to visit at least l locations giving $k_i \geq l$, where k_i is the number of locations visited by path i. This definition is for the Min-sum variant of mTSP.

Problem 3.1.1 (Multi Traveling Salesman Problem (mTSP))

$$\underset{\Sigma}{\text{minimize}} \quad \mathcal{L} = \sum_{p=1}^{m} \left(\left\| \boldsymbol{s}_{\sigma_{k_i}^p} - \boldsymbol{s}_{\sigma_1^p} \right\| + \sum_{i=1}^{k_p - 1} \left\| \boldsymbol{s}_{\sigma_i^p} - \boldsymbol{s}_{\sigma_{i+1}^p} \right\| \right) \tag{1}$$

$$\text{s.t.} \quad \sigma_i^p \neq \sigma_j^a \text{ for } \forall i \neq j \textit{ and } i \neq 1 \textit{ and } \forall a \, \forall p \in \{1 \dots m\} \tag{2}$$

$$\forall i \in \{1 \dots m\} \;\; k_i \geq l \tag{3}$$

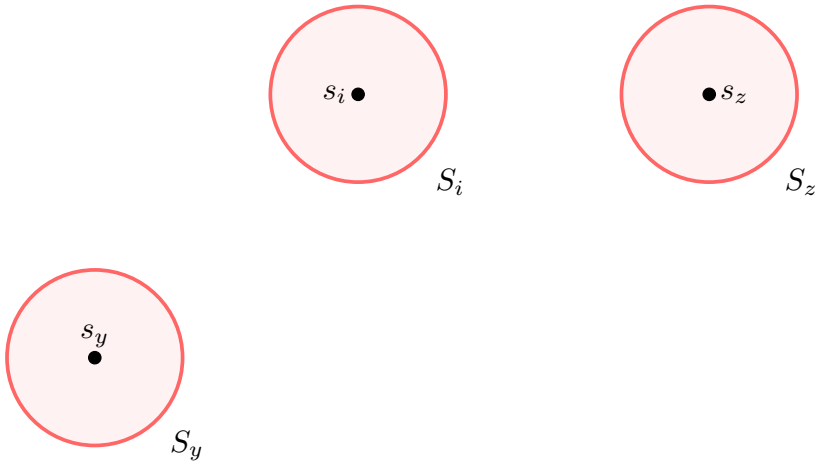

Figure 3.3: Vusualization of CEmTSP notation

3.2 Close Enough Multi-Traveling Salesman Problem

Having a set of n sensor neighborhoods S_i, each defined by $S_i \in (s_i, \delta_i)$, s_i being the location of the sensor i defined by $s_i \in R^2$ and δ_i is the size of the disk-shaped neighborhood. δ_i is always the same except for depot's $\delta_1 = 0$. The CEmTSP is to determine a set of m paths with minimal distance traveled, while visiting all s_i once, starting and ending in depot s_1. The paths are a sequence of $\Sigma = (\Sigma^1 \dots \Sigma^m)$. It denotes the sequence of visited locations, where $\Sigma^i = (\sigma_1^i \dots \sigma_{k_i}^i)$. Which is translated into to sequence of waypoints that are to be traveled $P = (P^1 \dots P^m)$, where $P^i = (p_1^i \dots p_{k_i}^i)$ where P_j^i is defined by $p_j^i \in R^2$ and follows constraint.

$$\left\| s_{\sigma_j^i} - p_j^i \right\| \leq \delta_{\sigma_j^i}$$

Each vehicle needs to visit at least l locations giving $k_i \geq l$ and k_i being the number of locations visited by path i.

Problem 3.2.1 (Close Enought Multi TSP (CEmTSP))

$$\underset{\Sigma}{\text{minimize}} \quad \mathcal{L} = \sum_{z=1}^{m} (\|p_k^z - \boldsymbol{p}_1^z\| + \sum_{i=1}^{k} \left\|\boldsymbol{p}_i^z - \boldsymbol{p}_{i+1}^z\right\|) \tag{4}$$

$$\text{s.t.} \quad \sigma_i^l \neq \sigma_j^g \text{ for } \forall i \neq j \textit{ and } z \neq g \textit{ and } i \neq 1 \tag{5}$$

$$\left\| s_{\sigma_j^i} - p_j^i \right\| \leq \delta_{\sigma_j^i} \textit{ for } \forall i \forall j \tag{6}$$

$$k_i \geq l \textit{ for } \forall i \in \{1 \dots m\} \tag{7}$$

Chapter 4
Hopfield Neural Network for TSP

In this chapter, Hopfield Neural Network (HNN) for TSP is introduced, as a basis on which an HNN for CEmTSP can be constructed, notation from [2] is followed. Hopfield Neural Network (HNN) [2] is a recurrent neural network, which utilizes the gradient descent method of a given energy function. The energy function is an abstraction of the problem, that gives an optimal solution to a given problem when it reaches the global minimum. The HNN is modeled as a matrix of numbers of dimension $n \times n$, where n is the number of Locations. HNN utilizes two matrices, U which is the input matrix that is put into the sigmoid Equation (8), and the output matrix V (referred to as the state matrix) which is produced by Equation (7). Each column x represents the time when a given location is visited and row y gives what location is being visited. Each number in the matrix is between 0 and 1 representing the likelihood of the location y being visited at time x. The update cycle iterates over all of the numbers in the matrix and uses gradient descent, to find the minima of the energy function.

$$V_{xy} = \frac{1}{1 + \exp(-U_{xy})} \tag{8}$$

This sigmoid function should ensure convergence, but U does not need to be saved and can be calculated from

$$U_{xy} = \log(V_{xy}) - \log(1 - V_{xy}) - t\frac{\partial E}{\partial V_{xy}} \tag{9}$$

V converges so that in every column and every row, there should be one 1 and the rest should be 0. Since the algorithm can reach local minima, it is re-run with a different initial V. The algorithm goes as follows:

1. Generate a V matrix, for the first matrix set every value to a random value.
2. Calculate U_{xi} with the help of Equation (9)
3. Update V by sigmoid Equation (8)
4. Check if the V columns are not close to having one 1 and the rest 0 if so you got the minimum, note it might be a local minimum, if not go to step 2. The V converges if the sum in every collum is close to 1 and the maximum number in every collum is close to 1 as well, the values depend on the setting of HNN.

HNN is shown in Algorithm 1.

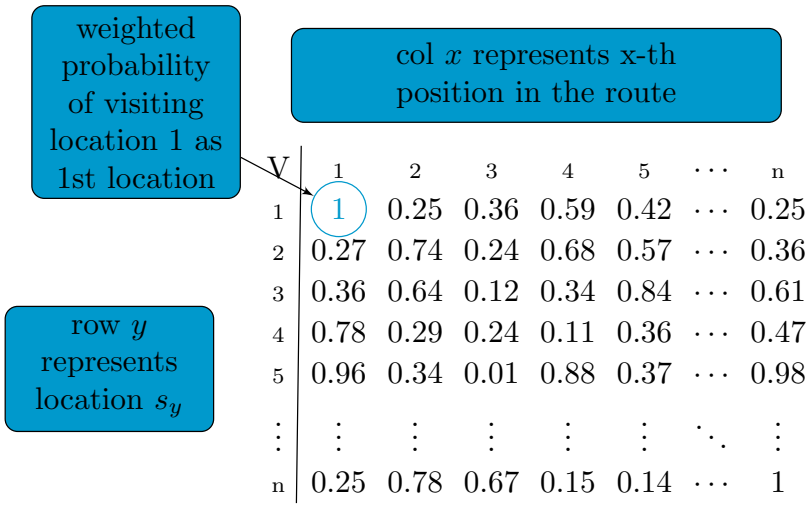

Figure 4.4: Vizualization of HNN matrix for TSP (inspired by [1])

4.1 Energy Function

The energy function for TSP [2] consists of four terms E_a, E_b, E_c, E_d each corresponding to a different constraint and has corresponding weights of a, b, c, and d. More discussed in Section 4.2 . The energy function follows

$$E = a\,E_a + b\,E_b + c\,E_c + d\,E_d \tag{10}$$

The first term E_a term minimizes the total sum in every column

$$E_a = \sum_{x=1}^{n-1}\sum_{i=1}^{n}\sum_{j=1 i\neq j}^{n} V_{xi}\,V_{xj} \tag{11}$$

E_b term minimizes the total sum in every row

$$E_b = \sum_{x=1}^{n-1}\sum_{i=1}^{n}\sum_{j=1 (j\neq x}^{n} V_{xi}\,V_{ji} \tag{12}$$

E_c term is to avert the situation when V_{xi} is stuck at 0 or 1

$$E_c = [\sum_{x=1}^{n}((\sum_{i=1}^{n} V_{xi}) - 1)^2 + \sum_{i=1}^{n}((\sum_{x=1}^{n} V_{xi}) - 1)^2] \tag{13}$$

E_d term is to calculate the distance between the Locations and minimize it. dist is the Euclidean distance between two Locations.

$$E_d = (\sum_{i=1}^{n-1}\sum_{y=1 y\neq i}^{n}\sum_{x=2}^{n-1} dist_{i,y}\,V_{x,i}\,(V_{x+1,y} + V_{x-1,y})) \tag{14}$$

4.2 Hyper-Parameters

Problem with the energy function is that it needs four hyper-parameters a, b, c, d, and one hyper–parameter for the step size t. These hyper-parameters have been deduced in [29].

$$c = n\ m \tag{15}$$

$$b = 3\ \Delta + c \tag{16}$$

$$a = b - d\ \nabla \tag{17}$$

$$d = \frac{1}{\Delta} \tag{18}$$

$$t = \frac{1}{20\ (n+m)\ \Delta} \tag{19}$$

where Δ is the shortest path between two locations and ∇ is the longest one.

4.3 Path Retrieval

To construct the final path from the state matrix V is to follow the rule that states, that there can be just one 1 in every row and column, and the rest should be 0. Hence, the algorithm goes through the matrix column by column finds the highest number, and saves the corresponding locations (the row) index to the corresponding position in the search (column). If the location was already assigned, the corresponding values of the matrix are compared. The one with the bigger value $V_{x,y}$ is assigned, the lower value is set to 0, and is recalculated. This is done till all the columns have been decided.

Algorithm 1 Hopfield Neural Network for TSP

Input: i_{max} – Maximum of trials
e_{max} – maximum epochs
S– set of locations
Output: Σ_{best} = $(\sigma_1 \ldots \sigma_n)$– best path yet found

```
t ← 1
while t ≤ t_max do
    V ← initialiseStateMatrix(n, m)
    e ← 1
    while e ≤ e_max do
        x ← 1
        while x ≤ n do
            y ← 1
            while y ≤ n do
                U_{x,y} ← log(V_{x,y}) − log(1 − V_{x,y}) − ∂E/∂V_{x,y}   ▷ Equation (9)
                V_{x,y} ← 1/(1+e^{−U_{x,y}})                              ▷ Equation (8)
    Σ ← pathRetrieval(V)
    currentCost ← pathCost(Σ)
    if currentCost < bestCost then
        bestCost ← currentCost
        Σ_best ← Σ
```

Chapter 5
Proposed HNN-based Solution for CEmTSP

The HNN approach described in Chapter 4 is extended to CEmTSP and these modifications of HNN are proposed:

- Change of state matrix V from a table of $n \times n$ to another from which a valid CEmTSP path can be constructed while the complexity of the problem is minimized.
- A new modified energy function that considers the disk-shaped neighborhoods.

As well as solutions for drawbacks of HNNs mentioned in Section 2.5 which are as follows:

- Hyper-parameters that are sufficient enough to make the energy function solve the CEmTSP.
- Inadmissible solutions to CEmTSP, are addressed.
- Methods to escape local minima of energy function are proposed.

5.1 Modified Model of HNN

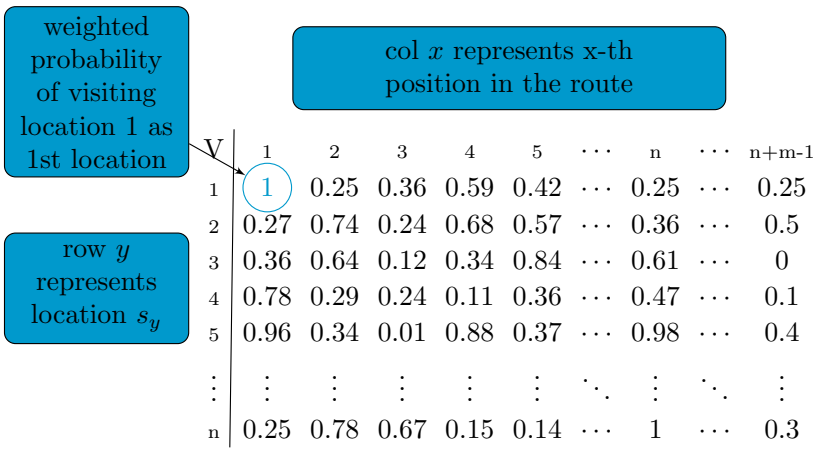

Figure 5.5: Vizualization of state matrix V matrix for CEmTSP (inspired by [1]).

The model that was used for the TSP in Chapter 4 would not be sufficient, since the depot needs to be visited m times. There are two possible ways to solve this. The first and least optimal is making an entire matrix for each vehicle making the model of a state matrix $m \times n \times n$, this leads to a few problems, getting a path from this matrix is harder due to the fact not every collum will have 1 somewhere, so it needs to be found which collum is redundant and which is required, as well the complexity of the problem is $O(m\ n^2)$, which compared to brute force $O((n-1)!/2)$ is better, but compared to TSP $O(n^2)$ it is not ideal.

The other method and the one that has been used is that for every additional vehicle, a collum is added making the matrix $n \times (m + m - 1)$, which leads to complexity of $O(n\ (n + m - 1))$, which is preferable to the other option and getting path from the matrix same as TSP, with an expectation to decide when to visit the depot. This model basis is that as the depot S_1 or row one is to be visited m times rest is the same as TSP so only $m - 1$ rows need to be added. This leads to a problem that E_d will optimize so S_1 will be visited m times in a row, this is solved in Section 5.2 with the addition of a new term to energy function E_f.

5.2 Modified Energy Function

The energy function is similar to the TSP energy function from section 4.1, the difference is that a new f term, to ensure the minimal distance between depots (l) is introduced, and a D term is changed due to a different cost matrix, explained in Section 5.3. As to hyper-parameters, all parameters a, b, c, d, and t are the same as described in Section 4.2 , except for a new hyperparameter f

$$E = a\ E_a + b\ E_b + c\ E_c + d\ E_d + f\ E_f \tag{20}$$

d tries to ensure that the distance between Locations is as low as possible

$$E_d = (\sum_{i=1}^{n-1} \sum_{y=1 y\neq i}^{n} \sum_{z=1 z\neq i}^{n} \sum_{x=1}^{n+m-1} dist_{y,i,z}\ V_{x,i}\ V_{x+1,y}\ V_{x-1,z}) \tag{21}$$

f tries to ensure that the distance between visiting depot is at least a minimum distance (l)

$$E_f = \sum_{x=1}^{m+m-1} \sum_{z=-l\ z\neq 0}^{l} V_{x+l,1}\ V_{x,1} \tag{22}$$

and hyper-parameter f is

$$f = a + c \tag{23}$$

5.3 Cost Matrix Calculation

To calculate the distance between neighborhoods S_y, S_i, S_z a simple calculation was done. The idea of the calculation is to find the closest point $p_{y,i,z}$ to the path between s_y and s_z that is still in the neighborhood S_i. This is done by finding the projection of s_i on the path between s_y and s_z, and then pulling the point $p_{y,i,z}$ until it reaches the neighborhood S_i. The calculation of the distance for $dist$ matrix is

$$dist_{y,i,z} = \|s_y - p_{y,i,z}\| + \|s_z - p_{y,i,z}\| \tag{24}$$

Note that, these distances are only expected distances and are used only for the energy function term E_d term, not for the final path cost.

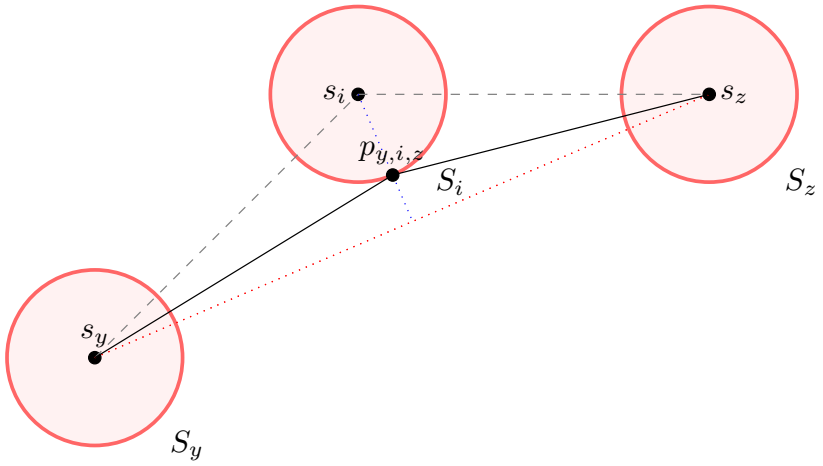

Figure 5.6: Visualization of the path between neighbourhoods S_y, S_i, S_z in this order

5.4 Local Minimum Problem

As mentioned, the issue with gradient descent is that the algorithm will end in a local minima and there is no certainty if the local minima is a global one or not. There are many ways to tackle the issue, one of them is re-initialization after a certain number of iterations of HNN or if the state matrix V has converged to a local minimum, the ways to reinitialize the state matrix are described in this section.

5.4.1 Random Re-initialization (Referred to as $Random$)

The random re-initialization method is the simplest, each time a local minimum is reached the V state matrix will be randomly reinitialized this ensures that multiple local minimums will be visited but does not ensure that progressively lower minima will be visited.

5.4.2 Biased Re-initialization

This type of re-initialization depends on the previous path Σ, which is a joint path of all Σ^i, to find a different local minimum. A certain amount of locations in Σ are shuffled. This should result in visiting local minima that are progressively better.

Unbiased shuffle (Referred to as $Unbiased$) draws two random numbers between 1 and the size of the path, if they are the same the numbers are re-rolled, and the corresponding values in Σ are swapped.

Biased shuffle (Referred to as $Biased$) depends on distances between Locations, a new array is created $T = (T_1, ..T_{n+m-1})$ where

$$T_i = \|p_{i-1} - p_i\| + \|p_i - p_{i+1}\| - dU + T_{i-1}$$

this gives every location in Σ a weight based on the distance to and from its surrounding locations. Then 2 random numbers are generated that $r_i \in 0..T_{n+m+1}$, $i \in 1, 2$, and index is found so

$$idx_i = l \text{ if } T_{l-1} < r_i < T_l$$

Suppose $idx_1 == idx_2$ new two random numbers are generated, if not the visited locations in Σ are swapped. This should ensure that the longer paths are more likely to be swapped.

Construction of state matrix V After a certain amount of locations were swapped, a new V is initialized such that

$$V_{x,i} = \begin{cases} Z & \text{if } \Sigma_x == i \\ Y & \text{otherwise} \end{cases}$$

Value of Z, Y is arbitrary but $Z > Y$.

5.5 Generating Path From State Matrix

As stated in Section 4.3 state matrix needs to be transformed into the most probable state, with the difference that m 1 will be in the first row, and a sequence of waypoints P is to be generated.
Getting the position of the depot in the path is the first thing that needs to be found. That is done by finding the maximum in the first row of V and setting surrounding l values to $-\infty$ then repeating this m times.

Then the rest of the path is found by the same algorithm as in Section 4.3 with the exception that those columns that have been decided in the previous step are skipped, and the first row is ignored for finding maximum.

After generating the path Σ, it now needs to be split between the vehicles, which is done by finding the position of depots, and each vehicle has its path from the depot to the next depot position where the next vehicle path begins.

After getting path Σ^i for each vehicle now it's important to find its waypoints. Those waypoints are the same that have been used to calculate distances between locations in Section 5.3 , thus the sequence of waypoints is constructed as follows

$$p_a^i = p_{\sigma_a^i - 1, \sigma_a^i, \sigma_a^i + 1}$$

where the point $p_{\sigma_a^i - 1, \sigma_a^i, \sigma_a^i + 1}$ is the same as $p_{,y,i,z}$ from Section 5.3. Gradual improvements to the position of the waypoint can be made by recalculating the position of $p_{\sigma_a^i - 1, \sigma_a^i, \sigma_a^i + 1}$ with a similar calculation as in Section 5.3, but instead of taking the Center point of the neighboring locations, waypoints p_{a-1}^i and p_{a+1}^i are taken.

As there can be invalid solutions to the CEmTSP obtained from HNN, there needs to be simple validation that every region is visited at least and only once, except for the depot. This can be done by checking if every value in Σ is different and then the waypoints in P are in the region of the corresponding location.

5.6 Parallelization of HNN

As mentioned in Section 2.5 , HNNs are suitable for massively parallel computation with the problem that the update of the state matrix V runs into memory synchronization difficulties. This can be solved in multiple ways, but this thesis uses only one method.

Neuron level in this parallelization each thread has an entire row of neurons to compute, in this case making the amount of threads possible $n + m - 1$ and each thread is computing n weights. The disadvantage of this approach is the new matrix V_n needs to be made, in which the new weights are stored and all derivations are still calculated from V, making the Neural Network learn slower. This is shown in Algorithm 2.

5.7 Path Optimization

As the path obtained from HNN might not be optimal, two optimizations are used, 2-opt [16] and path smoothing algorithm. The path smoothing algorithm is a simple algorithm that tries to find the best coordinates to visit neighborhood i when going from y to z the calculation is the same as in Section 5.3. Except taking the waypoints p_y and p_z instead of their centers. These calculations are iteratively done over the whole path to improve the location of waypoints to correspond to the final path.

Algorithm 2 Parallel Hopfield Neural Network for CEmTSP

Input: i_{max} – Maximum of trials
m –number of Vehicles to be used
n – Number of Locations to be visited
e_{max} – maximum epochs
Σ_{best} – Best path previously found - optional
S– set of locations
bestCost – cost for best path previously found - optional
Output: $\Sigma_{best} = (\Sigma_1 \dots \Sigma_m)$– best path found
bestCost – finally cost of the path
$P_{best} = (P^1 \dots P^m)$ – best path waypoints found

$t \leftarrow 1$
$\Sigma_{best} \leftarrow \Sigma_{init}$
$P_{best} \leftarrow \text{generateWaypoints}(\Sigma_{\text{best}})$
$bestCost \leftarrow \text{pathCost}(\text{P}_{\text{best}})$
while $t \leq t_{max}$ **do**
  $V \leftarrow \text{initialiseStateMatrix}(\text{n}, \text{m}, \Sigma_{\text{best}})$
  $e \leftarrow 1$
  while $e \leq e_{max}$ **do**
    $x \leftarrow 1$
    while $x \leq n + m - 1$ **do**
      $y \leftarrow 1$
      while $y \leq n$ **do**
        $U_{x,y} \leftarrow \log(V_{x,y}) - \log(1 - V_{x,y}) - \frac{\partial E}{\partial V_{x,y}}$ ▷ Equation (9)
        $V_{n_{x,y}} \leftarrow \frac{1}{1+e^{-U_{x,y}}}$ ▷ Equation (8)
  $V \leftarrow V_n$
  $\Sigma \leftarrow \text{pathRetrieval}(\text{V})$
  $P \leftarrow \text{generateWaypoints}(\Sigma)$
  $currentCost \leftarrow \text{pathCost}(\text{P})$
  if $currentCost < bestCost$ **then**
    $bestCost \leftarrow currentCost$
    $P_{best} \leftarrow P$
    $\Sigma_{best} \leftarrow \Sigma$

Chapter 6
Results

In this chapter, the results for the CEmTSP solved using proposed HNN with and without the optimizations as described in Chapter 5, are evaluated. The comparison is with the SOM_{euc} algorithm [17], omitting Dubins model and using only Euclidean distances (denoted as SOM_{euc}). The other is an Greedy K-means Heuristic [25], used as a baseline without neighborhoods denoted as K-means. As well a comparison of different proposals to escape local minima, which are described in Section 5.4, are evaluated.

The evaluation is done through two values %PDB and %PDM where L_{ref} is a solution from SOM_{euc} solver.

%PDB is a quality comparison between the best values of SOM_{euc} and HNN solution (L). Where 0% is the solution was equal and a positive percentage is that our solution is less optimal than SOM.

$$\%\mathrm{PDB} = \frac{\mathrm{L} - \mathrm{L_{ref}}}{\mathrm{L_{ref}}}\ 100\% \tag{25}$$

The algorithm robustness is calculated as %PDM, which compares the solver mean value with the SOM_{euc} reference solution.

$$\%\mathrm{PDM} = \frac{\mathrm{L_{mean}} - \mathrm{L_{ref}}}{\mathrm{L_{ref}}}\ 100\% \tag{26}$$

Solutions for HNN were running multi-threaded, on 2x Intel Xeon Scalable Gold 6146 with a maximum of 8 threads and a maximum of 4 hours per problem, with 10 Trials – how many times matrix was re-initialized, not all issues made all trials in time, 120 epochs – steps to improve matrix V by gradient descend, and starting matrix being random, hyper-parameters are described in Section 4.2. SOM_{euc} was run on Intel® Core™ i9-13900K with 20 trials. Both SOM_{euc} and HNN were run on $\delta = 40$. Greedy K-means heuristic was run on AMD Ryzen 9 3900X with $\delta = 0$ for 120 epochs. In all problems, m = 3 vehicles were used. Problems have been chosen from TSPLIB [30].

6.1 Comparison of Re-initialization

As mentioned in Section 5.4, multiple ways to escape local minimums were investigated, this section is interested in finding which performed the best and which should be used in the next problems.

As can be seen, in Figure 6.7 and Tables from B.1 and B.2, there isn't a certain re-initialization that would always be the best. There are multiple possible ways to decide the best re-initialization.

The first is average %PDB which gives us, $Biased = 1280.57\%$, $Random = 1225.30\%$ and, $Unbiased = 1210.48\%$. In this $Unbiased$ reinitialization has the best results. The disadvantage of this approach is that one "lucky" randomization or problem might have a great influence on the resulting %PDB. So this can be improved by mean %PDB but it turns out the results are the same as average.

Another option is to find the average %PDM which gives us, $Biased = 5721.68\%$, $Random = 5633.73\%$ and, $Unbiased = 5483.17\%$, and its mean values are $Biased = 1355.13\%$, $Random = 1334.30\%$ and, $Unbiased = 1292.65\%$.

As in all of the possible ways to decide the best re-initialization the $Unbiased$ one seems to be the most reliable followed by $Random$ and lastly the $Biased$ one.

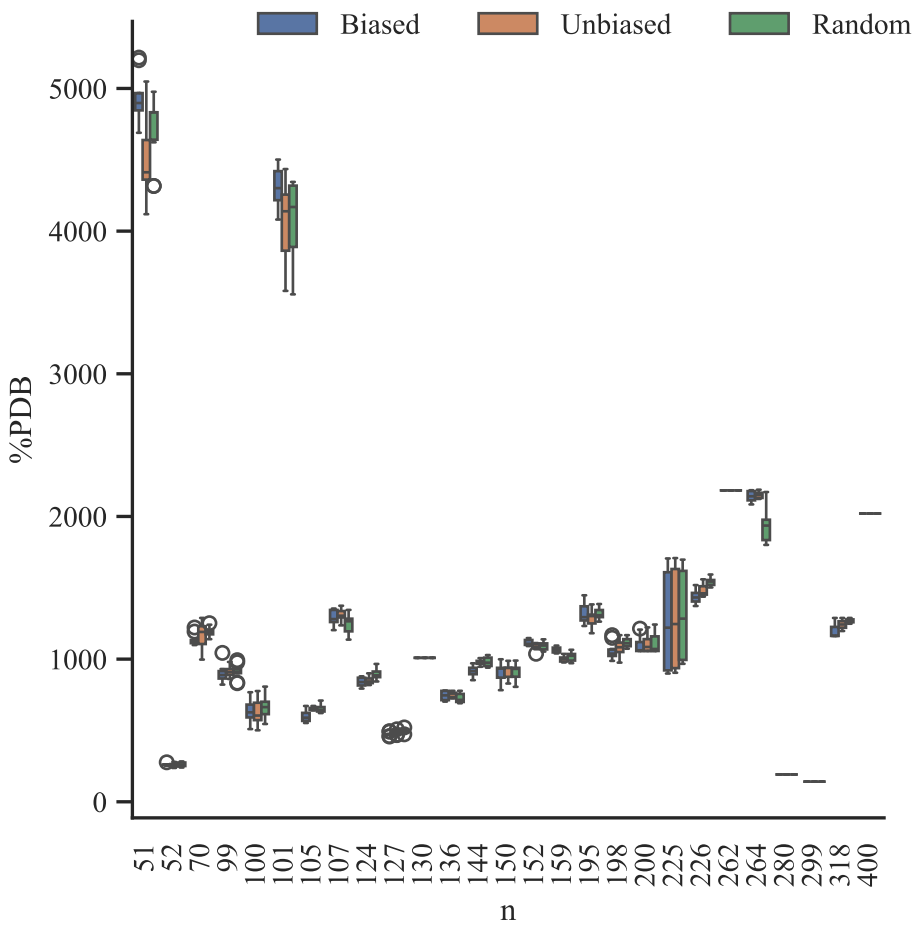

Figure 6.7: Visual comparison for re-initialization

6.2 Non-Optimized

Now it's time to compare the K-means and SOM_{euc} to non-optimized HNN.
As can be seen from Figures 6.9, 6.8 and Tables 1, 2, the solution from HNN might be an admissible

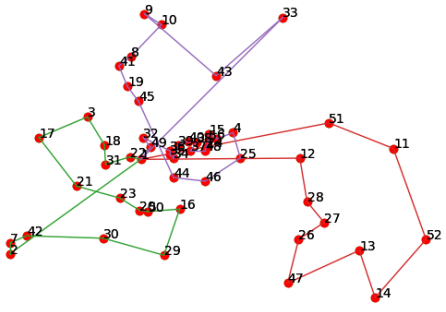

Figure 6.8: Solution of Berlin52 $\delta = 40$ given by Heruistic

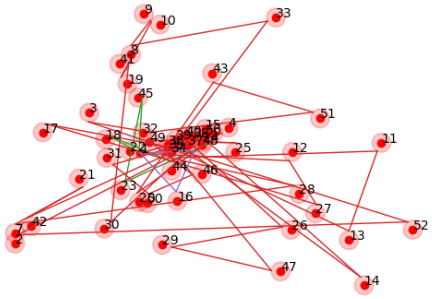

Figure 6.9: Solution of Berlin52 $\delta = 40$ given by HNN

solution for CEmTSP but far from optimal. Compared with SOM_{euc} unoptimized HNN is bad the average $\overline{\%\text{PDB}}$ is 1 210.48% the $\overline{\%\text{PDM}}$ is 54 83.17% and none of the solutions from HNN for any problem were better than the ones from SOM. Regarding the K-means, compared with that only one problem was solved with a better solution which is a280 any other is solved with a worse solution. This shows us that a non-optimized solution from HNN is not good enough to be a reliable solver for CEmTSP.

6.3 Optimalized of Path

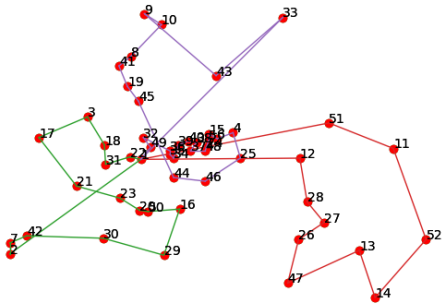

Figure 6.10: Solution of Berlin52 $\delta = 40$ given by Heruistic

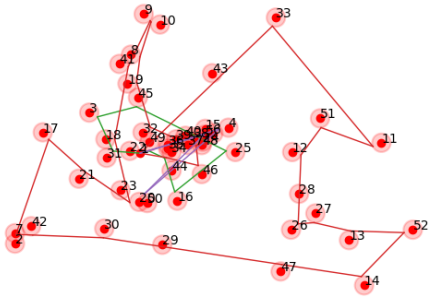

Figure 6.11: Solution of Berlin52 $\delta = 40$ given by HNN with 2-opt optimalization

As can be seen in Tables 1, 2, and in Figures 6.11 and 6.9 optimization gives improved solutions than the unoptimized solution. Compared to the K-means, the optimized HNN gives 13 better results out of 38 problems. When compared to the SOM_{euc} solver the HNN is now able to find better solutions in 2 problems and its mean $\overline{\%\text{PDB}}$ improved to 275.89 % from 1 210.48% and the mean $\overline{\%\text{PDM}}$ has reached 329.67% from 1 298.65%. This shows that the optimization greatly affects the final solution, but even with the optimization, it can't be said it is an improvement on the SOM approach. It hasn't been proven that there is a significant increase in time complexity while using optimization.

Table 1: Results HNN was solved with unbiased re-initialization, Part-1

Problem		K-means [25]			SOM_{euc} [18]				HNN			HNN_{2opt}	
	L	%PDB	%PDM	L	%PDB	%PDM	L	%PDB	%PDM	$T_{[cpu]}$ [H]	L	%PDB	%PDM	$T_{[cpu]}$ [H]
a280	4100.41	246.44	246.44	**1183.59**	**0**	**6.63**	2267.62	91.59	91.59	148134.83	-	-	-	-
berlin52	**9958.77**	**-6.49**	**-6.49**	10 650.41	0.00	4.09	25 266.52	137.24	157.18	116.67	12 045.19	13.10	20.01	115.54
bier127	149 431.17	18.59	18.59	**126002.6**	**0**	**4.24**	588 874.97	367.35	385.96	6618.42	181 799.69	44.28	48.72	6870.08
ch130	7976.28	88.67	88.67	**4227.58**	**0**	**8.74**	42 659.72	909.08	909.08	9600.35	6283.96	48.64	48.64	6304.2
ch150	9834.09	96.18	96.18	**5012.87**	**0**	**5.73**	47 069.47	838.97	838.97	21117.2	7635.26	52.31	52.31	11521.15
d198	26 673.00	56.61	56.61	**17031.85**	**0**	**10.95**	166 163.58	875.60	983.66	59907.19	29 966.70	75.95	107.84	19182.73
d493	53 811.48	76.38	76.38	**30508.93**	**0**	**2.36**	-	-	-	-	-	-	-	-
eil101	869.32	1669.29	1669.29	**49.13**	**0**	**36.37**	1759.55	3481.15	3975.48	3090.01	954.98	1843.65	2123.48	2051.49
eil51	630.38	2663.43	2663.43	**22.81**	**0**	**-0.0**	939.53	4018.68	4424.18	111.25	398.56	1647.21	1897.21	110.97
eil76	793.28	5532.51	5532.51	**14.08**	**0**	**54.28**	1344.35	9445.28	10 424.47	630.14	582.02	4032.49	4874.23	578.01
fl417	19 871.95	48.38	48.38	13 392.58	0.00	0.93	-	-	-	-	**12838.2**	**-4.14**	**-4.14**	139569.87
gil262	3595.47	324.00	324.00	**847.99**	**0**	**20.63**	18 512.36	2083.08	2083.08	171267.88	4949.29	483.65	483.65	169708.62
kroA100	29 828.32	21.34	21.34	**24582.45**	**0**	**3.2**	141 112.83	474.04	524.72	2578.84	34 381.97	39.86	64.77	1989.01
kroA150	35 233.74	25.40	25.40	**28096.19**	**0**	**2.3**	232 933.00	729.06	767.70	21940.82	37 818.31	34.60	67.95	10825.27
kroA200	42 222.62	51.34	51.34	**27899.4**	**0**	**3.03**	308 791.56	1006.80	1046.49	69892.56	43 468.90	55.81	86.74	37563.22
kroB100	29 491.45	32.39	32.39	**22275.35**	**0**	**7.78**	148 283.86	565.69	596.79	3848.01	26 780.46	20.22	50.01	2001.63
kroB150	38 011.49	45.22	45.22	**26175.77**	**0**	**3.78**	226 712.72	766.12	842.60	32120.88	41 090.71	56.98	83.29	11629.76
kroB200	40 326.21	33.54	33.54	**30198.56**	**0**	**3.81**	319 247.36	957.16	958.09	91132.05	38 393.13	27.14	27.70	38702.67
kroC100	30 689.16	37.63	37.63	**22298.1**	**0**	**4.29**	156 925.97	603.76	641.53	3852.02	30 752.29	37.91	83.75	1992.68
kroD100	28 885.35	13.32	13.32	**25489.88**	**0**	**6.33**	132 984.58	421.72	494.52	3838.4	35 096.56	37.69	55.89	1951.16
kroE100	34 203.52	27.51	27.51	**26825.13**	**0**	**5.66**	136 919.10	410.41	450.34	3736.92	31 427.02	17.16	41.04	2008.84
lin105	26 235.59	38.64	38.64	**18923.12**	**0**	**5.92**	121 160.74	540.28	551.90	4643.67	29 117.24	53.87	70.65	1681.95
lin318	61 892.48	34.22	34.22	**46111.92**	**0**	**2.91**	551 977.37	1097.04	1142.80	151978.91	84 558.67	83.38	84.62	152011.36
pcb442	69 131.24	54.09	54.09	**44864.4**	**0**	**3.17**	-	-	-	-	-	-	-	-
pr107	63 936.76	38.03	38.03	**46321.88**	**0**	**29.59**	572 865.39	1136.71	1209.73	3944.46	88 803.68	91.71	124.87	2634.28
pr124	87 505.43	11.39	11.39	**78560.1**	**0**	**0.8**	642 918.06	718.38	748.80	7221.5	128 905.90	64.09	71.00	4857.9
pr136	136 958.61	19.72	19.72	**114396.9**	**0**	**2.79**	828 466.19	624.20	652.34	12131.31	200 465.95	75.24	80.78	6961.88
pr144	97 720.40	16.77	16.77	**83682.75**	**0**	**8.23**	792 043.69	846.48	877.14	16045.58	141 918.83	69.59	76.84	8884.17
pr152	115 794.59	22.47	22.47	**94545.75**	**0**	**20.3**	980 868.07	937.45	986.13	31540.23	172 014.28	81.94	93.71	11335.69
pr226	134 803.45	22.99	22.99	**109602.1**	**0**	**6.52**	1 576 026.90	1337.95	1382.67	157272.25	165 177.76	50.71	67.30	78871.15

Table 2: Results HNN was solved with unbiased re-initialization, Part 2

Problem		K-means [25]			SOM_{euc} [18]				HNN				HNN_{2opt}	
	L	%PDB	%PDM	L	%PDB	%PDM	L	%PDB	%PDM	$T_{[cpu]}$ [H]	L	%PDB	%PDM	$T_{[cpu]}$ [H]
pr264	61 565.86	9.75	9.75	**56095.88**	**0**	**17.06**	1 190 817.13	2022.82	2051.02	218775.16	114 132.83	103.46	112.79	139307.29
pr299	74 352.74	38.64	38.64	53 631.41	0.00	4.21	75 675.85	41.10	41.10	120852.37	**51941.18**	**-3.15**	**-3.15**	165707.1
pr439	146 120.06	24.29	24.29	**117566.4**	**0**	**2.07**	-	-	-	-	-	-	-	-
pr76	170 659.75	32.40	32.40	**128901.4**	**0**	**2.03**	590 003.75	357.72	368.37	1011.27	202 738.97	57.28	63.83	570.13
rat195	3421.43	162.21	162.21	**1304.84**	**0**	**7.04**	15 417.99	1081.60	1190.36	86775.92	3974.96	204.63	225.01	35559.55
rat99	1917.49	181.86	181.86	**680.29**	**0**	**15.73**	5856.12	760.83	814.05	2257.19	1763.60	159.24	218.79	1912.3
rd100	13 024.79	54.80	54.80	**8414.1**	**0**	**5.25**	42 180.42	401.31	469.58	3259.75	8817.39	4.79	44.05	2515.9
rd400	20 854.46	113.51	113.51	**9767.59**	**0**	**3.21**	197 356.15	1920.52	1920.52	171483.98	16 262.11	66.49	66.49	146938.31
st70	1035.29	446.04	446.04	**189.6**	**0**	**13.85**	1890.17	896.92	1069.76	877.45	938.72	395.10	476.35	526.77
ts225	184 937.23	13.55	13.55	**162862.1**	**0**	**3.35**	1 473 144.83	804.54	836.21	143094.39	240 928.91	47.93	70.41	62787.93
tsp225	6018.32	200.68	200.68	**2001.56**	**0**	**9.1**	30 285.39	1413.09	1537.94	152051.31	6330.89	216.30	257.18	64267.95
u159	68 140.82	53.86	53.86	**44288.0**	**0**	**8.93**	432 562.23	876.70	901.67	25581.17	87 161.64	96.81	112.90	20442.22

Chapter 7
Conclusion

This thesis was interested in solving Close Enough Multi Traveling Problem (CEmTSP). With the main interest being to solve the Min-sum version of the mTSP. Part of the problem, with the use of a fully connected neural network called Hopfield Neural Network (HNN) using gradient descent to find the minimum in a given energy function. As well as adding optimization functions to the solution given by HNN to see if the optimizations will improve the solution quality.

The basis of the thesis is the HNN from [2]. To extend this HNN to cover CEmTSP, several modifications were required. One of the modifications was a change of energy function to solve the continuous neighborhood part of the problem. Another modification is the change of representation, from state matrix $n \times n$ to matrix $n \times (n + m - 1)$ to solve the "Multi" part of CEmTSP. Since the gradient method can get stuck in a local minima, multiple ways to escape it were suggested and evaluated.

As the baseline that HNN is compared to, SOM solver from [18] has been used, with Dubins replaced with Euclidean distance, and a heuristic approach of K-means with a greedy heuristic [25].

The results have shown HNN without optimization gives valid solutions, but the quality of solutions is never optimal this, in the author's opinion, can be caused by multiple reasons. (i) An erroneous choice of energy function can be the cause of problems. The energy function might not have global or local minima in the corresponding solutions resulting in the solutions from HNN being sub-optimal. (ii) Another possibility is that the number of epochs is insufficient to find the optimal solution. (iii) That the path retrieval function might be failing and giving incorrect translation from V to Σ, this author finds unlikely because it should give the most probable solution from V, the only case when that is not the case is the location of $\sigma = 1$, that is found thru the highest value in row 1 without input from any other values.

Regarding the escape of the local minima, three ways to escape were investigated $Random$ re-initialization, $Biased$ re-initialization, and $Unbiased$ re-initialization. From these, the best fairing re-initialization was the $Unbiased$ one.

As it comes to optimization, it was done by the 2-opt algorithm with path smoothing. The optimized results were an improvement, but it hasn't made the HNN approach competitive with SOM.

Overall, HNN is not competitive with the adapted SOM_{euc} algorithm [18] and has a problem defeating a simple heuristic [25]. The author thinks, the possible problem is in the energy function and the number of reinitializations, the algorithm could be improved by adding a heuristic solution as a jumping-off point from which to find better solutions.

References

[1] J. Deckerová, "Artificial neural networks in solution of the orienteering problems," 2018.

[2] Y. Luo, "Design and improvement of hopfield network for tsp," in *Proceedings of the 2019 International Conference on Artificial Intelligence and Computer Science*, 2019, pp. 79–83.

[3] Y. Takahashi, "Mathematical improvement of the hopfield model for tsp feasible solutions by synapse dynamical systems," *Neurocomputing*, vol. 15, no. 1, pp. 15–43, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231296000446

[4] D. L. Applegate, *The traveling salesman problem: a computational study*. Princeton university press, 2006, vol. 17.

[5] R. Matai, S. P. Singh, and M. L. Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches," *Traveling salesman problem, theory and applications*, vol. 1, no. 1, pp. 1–25, 2010.

[6] J. Faigl *et al.*, "An application of self-organizing map for multirobot multigoal path planning with minmax objective," *Computational intelligence and neuroscience*, vol. 2016, 2016.

[7] O. Cheikhrouhou and I. Khoufi, "A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy," *Computer Science Review*, vol. 40, p. 100369, 2021.

[8] R. Li, J. Qiao, and W. Li, "A modified hopfield neural network for solving tsp problem," in *2016 12th World Congress on Intelligent Control and Automation (WCICA)*. IEEE, 2016, pp. 1775–1780.

[9] U.-P. Wen, K.-M. Lan, and H.-S. Shih, "A review of hopfield neural networks for solving mathematical programming problems," *European Journal of Operational Research*, vol. 198, no. 3, pp. 675–687, 2009.

[10] Z. Hanzálek, Mar 2022. [Online]. Available: https://rtime.ciirc.cvut.cz/~hanzalek/KO/TSP_e.pdf

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

[12] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[13] D. Karapetyan and G. Gutin, "Lin–kernighan heuristic adaptations for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 208, no. 3, pp. 221–232, 2011.

[14] D. Sanches, D. Whitley, and R. Tinós, "Improving an exact solver for the traveling salesman problem using partition crossover," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 337–344.

[15] Q. T. Luu, "Traveling salesman problem: Exact solutions vs. heuristic vs. approximation algorithms," Mar 2024. [Online]. Available: https://www.baeldung.com/cs/tsp-exact-solutions-vs-heuristic-vs-approximation-algorithms

[16] S. Hougardy, F. Zaiser, and X. Zhong, "The approximation ratio of the 2-opt heuristic for the metric traveling salesman problem," *Operations Research Letters*, vol. 48, no. 4, pp. 401–404, 2020.

[17] J. Faigl, R. Pěnička, and G. Best, "Self-organizing map-based solution for the orienteering problem with neighborhoods," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2016, pp. 001 315–001 321.

[18] J. Faigl and P. Váňa, "Unsupervised learning for surveillance planning with team of aerial vehicles," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 4340–4347.

[19] P. He and J.-K. Hao, "Hybrid search with neighborhood reduction for the multiple traveling salesman problem," *Computers & Operations Research*, vol. 142, p. 105726, 2022.

[20] P. Oberlin, S. Rathinam, and S. Darbha, "A transformation for a multiple depot, multiple traveling salesman problem," in *2009 American Control Conference*. IEEE, 2009, pp. 2636–2641.

[21] J. Faigl, "GSOA: growing self-organizing array - unsupervised learning for the close-enough traveling salesman problem and other routing problems," *Neurocomputing*, vol. 312, pp. 120–134, 2018.

[22] D. J. Gulczynski, J. W. Heath, and C. C. Price, *The Close Enough Traveling Salesman Problem: A Discussion of Several Heuristics*. Springer US, 2006, pp. 271–283.

[23] F. Carrabs, C. Cerrone, R. Cerulli, and M. Gaudioso, "A novel discretization scheme for the close enough traveling salesman problem," *Computers Operations Research*, vol. 78, pp. 163–171, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054816302179

[24] D. Sinha Roy, B. Golden, X. Wang, and E. Wasil, "Estimating the tour length for the close enough traveling salesman problem," *Algorithms*, vol. 14, no. 4, p. 123, 2021.

[25] J. Faigl, M. Kulich, and L. Přeučil, "Goal assignment using distance cost in multi-robot exploration," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 3741–3746.

[26] M. Islam, G. Chen, and S. Jin, "An overview of neural network," *American Journal of Neural Networks and Applications*, vol. 5, no. 1, pp. 7–11, 2019.

[27] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons." *Proceedings of the national academy of sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[28] K. Ricanek, G. Lebby, and K. Haywood, "Hopfield like networks for pattern recognition with application to face recognition," in *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, vol. 5, 1999, pp. 3265–3269 vol.5.

[29] P. M. Talaván and J. Yáñez, "Parameter setting of the hopfield network applied to tsp," *Neural Networks*, vol. 15, no. 3, pp. 363–373, 2002.

[30] [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Appendix A
Content of the Enclosed CD

```
CD
├── etc
└── HNN
    ├── data
    ├── src
    │   ├── HNN.jl
    │   ├── run_hnn.jl
    │   ├── Pathtour.jl
    │   ├── help_functions.jl
    │   ├── TSP.jl
    │   ├── Energy.jl
    │   ├── plot.jl
    │   ├── structures.jl
    │   ├── function_and_derivations.jl
    │   └── heuristic.jl
    ├── tests
    │   ├── test_energy_terms.jl
    │   ├── test_gradient.jl
    │   ├── test_gradient_jd.jl
    │   └── test_normalization.jl
    ├── config.ini
    ├── Manifest.toml
    └── project.toml
```

Appendix B
Data

Table B.1: Comparison of reinitializations Part-1

Problem		*Random*			*Unbiased*			*Biaesed*	
	L	%PDB	%PDM	L	%PDB	%PDM	L	%PDB	%PDM
a280	**2267.62**	**91.59**	**91.59**	**2267.62**	**91.59**	**91.59**	**2267.62**	**91.59**	**91.59**
berlin52	25 580.40	140.18	161.75	**25266.52**	**137.24**	**157.18**	26 927.81	152.83	160.47
bier127	595 797.50	372.85	392.62	588 874.97	367.35	385.96	**576442.27**	**357.48**	**373.74**
ch130	**42659.72**	**909.08**	**909.08**	**42659.72**	**909.08**	**909.08**	**42659.72**	**909.08**	**909.08**
ch150	**47069.47**	**838.97**	**838.97**	**47069.47**	**838.97**	**838.97**	**47069.47**	**838.97**	**838.97**
d198	182 772.72	973.12	1015.65	**166163.58**	**875.6**	**983.66**	168 218.67	887.67	957.40
eil101	**1747.61**	**3456.85**	**3983.03**	1759.55	3481.15	3975.48	2005.23	3981.19	4208.92
eil51	984.65	4216.49	4622.68	**939.53**	**4018.68**	**4424.18**	1069.69	4589.28	4831.65
eil76	1389.41	9765.22	11 383.29	**1344.35**	**9445.28**	**10424.47**	1623.63	11 428.26	12 190.24
gil262	**18512.36**	**2083.08**	**2083.08**	**18512.36**	**2083.08**	**2083.08**	**18512.36**	**2083.08**	**2083.08**
kroA100	150 227.75	511.12	556.54	**141112.83**	**474.04**	**524.72**	145 493.92	491.86	552.74
kroA150	226 416.23	705.86	765.81	232 933.00	729.06	767.70	**219849.79**	**682.49**	**755.09**
kroA200	302 671.54	984.87	1070.15	308 791.56	1006.80	1046.49	**295485.36**	**959.11**	**1034.55**
kroB100	150 445.74	575.39	638.42	148 283.86	565.69	596.79	**127954.86**	**474.42**	**565.14**
kroB150	**213483.78**	**715.58**	**815.94**	226 712.72	766.12	842.60	226 712.72	766.12	835.40
kroB200	**319247.36**	**957.16**	**957.16**	**319247.36**	**957.16**	**958.09**	**319247.36**	**957.16**	**958.09**
kroC100	**141668.31**	**535.34**	**630.57**	156 925.97	603.76	641.53	147 111.08	559.75	611.47
kroD100	144 519.49	466.97	532.92	**132984.58**	**421.72**	**494.52**	137 452.31	439.24	515.17
kroE100	154 205.33	474.85	521.21	136 919.10	410.41	450.34	**136718.0**	**409.66**	**485.39**

Table B.2: Comparison of reinitializations Part-2

Problem		*Random*			*Unbiased*			*Biaesed*	
	L	%PDB	%PDM	L	%PDB	%PDM	L	%PDB	%PDM
lin105	117 687.94	521.93	550.20	121 160.74	540.28	551.90	**104521.69**	**452.35**	**495.64**
lin318	575 551.00	1148.16	1168.36	551 977.37	1097.04	1142.80	**536196.1**	**1062.81**	**1104.73**
pr107	**526573.57**	**1036.77**	**1147.83**	572 865.39	1136.71	1209.73	557 403.20	1103.33	1189.42
pr124	662 542.06	743.36	795.63	642 918.06	718.38	748.80	**623482.36**	**693.64**	**740.25**
pr136	**790298.97**	**590.84**	**627.33**	828 466.19	624.20	652.34	803 398.71	602.29	643.80
pr144	786 804.48	840.22	881.88	792 043.69	846.48	877.14	**713468.5**	**752.59**	**815.45**
pr152	995 376.49	952.80	993.25	**980868.07**	**937.45**	**986.13**	1 032 013.28	991.55	1015.02
pr226	1 646 541.42	1402.29	1443.25	1 576 026.90	1337.95	1382.67	**1504233.8**	**1272.45**	**1334.58**
pr264	**1010057.08**	**1700.59**	**1839.22**	1 190 817.13	2022.82	2051.02	1 169 274.98	1984.42	2041.47
pr299	**75675.85**	**41.1**	**41.1**	**75675.85**	**41.1**	**41.1**	**75675.85**	**41.1**	**41.1**
pr76	**568109.47**	**340.73**	**369.33**	590 003.75	357.72	368.37	571 501.91	343.36	359.38
rat195	16 478.26	1162.86	1216.93	**15417.99**	**1081.6**	**1190.36**	16 080.55	1132.38	1217.92
rat99	5654.27	731.16	811.36	5856.12	760.83	814.05	**5588.13**	**721.44**	**798.75**
rd100	45 892.20	445.42	500.23	**42180.42**	**401.31**	**469.58**	43 576.23	417.90	477.81
rd400	**197356.15**	**1920.52**	**1920.52**	**197356.15**	**1920.52**	**1920.52**	**197356.15**	**1920.52**	**1920.52**
st70	2161.10	1039.81	1092.29	**1890.17**	**896.92**	**1069.76**	2082.83	998.53	1035.63
ts225	1 575 161.17	867.17	900.32	1 473 144.83	804.54	836.21	**1464405.24**	**799.17**	**827.73**
tsp225	30 616.17	1429.62	1518.26	30 285.39	1413.09	1537.94	**29479.56**	**1372.83**	**1513.69**
u159	**430281.06**	**871.55**	**915.78**	432 562.23	876.70	901.67	460 563.68	939.93	964.06