# AI utilization in route planning for delivery trucks within the supply chain

ANDRÉ REIS & SEBASTIAN NORDQVIST

BACHELOR´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY |
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Bachelor's Thesis

# AI utilization in route planning for delivery trucks within the supply chain

By

André Reis & Sebastian Nordqvist

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

## Abstract

The supply chain has potential for growth. Many different parts can be optimized, and every possible improvement has not yet been tested. This study focuses on route optimization with the specific goal of figuring out how machine learning can be applied to route planning and how key factors that impact travel time for a route can be taken into account for this problem.

Time was dedicated to learning about machine learning, how it is applicable to route planning, as well as potential key factors that can be used with different machine learning algorithms. Once enough knowledge had been gathered, a prototype was implemented to verify key factors usability in route planning and test different route planning problems, such as point-to-point routes and traveling salesman problem.

Key factors were gathered during the thesis work, and based on the result of the thesis, their ability to be used in route optimization was verified. Methods of collecting the key factors were looked into, and two algorithms were tested that had the potential of using these factors. The two algorithms proved the usability of key factors and showed their potential in route planning problems. First, the "Neural Evolution of Augmenting Topologies" algorithm was tested and verified that it could solve simple route planning problems. Although, it was later overshadowed by a genetic algorithm solution, which could solve point-to-point travel better and showed usefulness in the traveling salesman problem.

The thesis work did not provide a full-scale solution to optimizing route planning. However, several conclusions were made on the topic, such as the possibility of training neural networks using supervised learning to calculate edge cost and genetic algorithms showing its potential in multi-stop route planning. We believe that several of the conclusions made in the thesis could show promise in the area of route optimization given enough resources.

## Keywords

Route optimization, Route planning, NEAT, Genetic algorithm, Key factors, Load planning, Machine learning.

# Sammanfattning

Försörjningskedjan har stor potential till optimering. Många olika delar av kedjan kan optimeras och alla tänkbara förbättringar har inte testats än. Denna studien har sitt fokus på ruttoptimering, med det specifika målet att förstå hur maskininlärning kan användas inom ruttplanering, samt hur nyckelfaktorer som påverkar restiden för en rutt kan tas i åtanke för problemet.

Tid lades på att studera maskininlärning, hur maskininlärning kan användas inom ruttplanering samt vilka potentiella nyckelfaktorer som kan användas i olika maskininlärnings algoritmer. När god kunskap inom maskininlärning hade erhållits, skapades en prototyp för att verifiera om nyckelfaktorerna var användbara inom ruttplanering. Prototypen användes även för att testa olika typer av ruttplaneringsproblem, såsom punkt-till-punkt rutter och handelsresande-problemet.

Nyckelfaktorer samlades in under examensarbetet och baserat på resultatet av examensrapporten så utvärderades möjligheten att använda dem i ruttplanering. Metoder för att samla faktorer undersöktes, och två algoritmer testades som hade möjligheten att utnyttja nyckelfaktorerna. Dessa två algoritmerna visade förmågan att använda nyckelfaktorer samt potential i ruttplanering. Först så testades "Neural Evolution of Augmenting Topologies" algoritmen och dess egenskap att lösa enkla ruttplaneringsproblem verifierades. Dock så överträffades den senare av en genetisk algoritm, som kunde lösa punkt-till-punkt planering bättre men också visade sin användbarhet i handelsresandeproblemet.

Under examensarbetet så skapades ingen helhetslösning för att optimera ruttplanering. Dock så kunde många slutsatser dras under examensarbetets gång, såsom att det kan vara gynnsamt att träna neuronnät med övervakad inlärning för att beräkna kostnader av bågar, samt att genetiska algoritmer visar potential i fler-stop ruttplanering. Vi tror att flera av de slutsatserna som har gjorts i denna examensrapport kan bidra till att förbättra ruttoptimering om tillräckligt mycket resurser läggs på att undersöka dem.

## Nyckelord

Ruttoptimering, Ruttplanering, NEAT, Genetisk algoritm, nyckelfaktor, Planering av last, Maskininlärning.

# Contents

## Acknowledgments

We would like to thoroughly thank Christin and Christian from LTH for their support and guidance in writing this thesis.

We would also like to thank Amit and Rahul from Capgemini Sverige for giving us the opportunity to cooperate with them on this task and for all the resources, guidance, and time they put in to help us.

André and Sebastian

# Preface

All of the thesis work has been done in cooperation with both authors. However, certain parts of the thesis were worked on separately. These parts are still deemed to have been worked on together, though, as both authors have spent equal time going through each other's work, and no sentence can be seen as only the work of a single author.

# 1. Introduction

As commonly used routing algorithms cannot keep up with the advancing complexity of delivery routes within the supply chain, new methods that can deal with the complexity have to be explored.

The following chapter introduces the concept of route optimization and provides the background information needed to understand why this area has to be optimized. The chapter also brings up ideas pertaining to how the routes can be optimized and explains the proposed way to tackle this problem with a couple of limitations to narrow the scope.

## 1.1 Capgemini Sverige AB

This thesis was done with the cooperation of Capgemini Sverige AB. Capgemini is working with consulting, digital transformation, technology, and engineering services, to address the entire breadth of clients' opportunities in the evolving world of cloud, digital, and platforms. Capgemini's experience and knowledge within the Digital Supply Chain were leveraged to study this topic.

In cooperation with Capgemini Sverige, two supervisors from the company helped in realising this thesis work, Amit Chougule and Rahul Baviskar.

Amit is leading end to end delivery and sales for a cluster within one of the world's top furniture retailers from Nordics. He also has the role of  Capability management within the Helsingborg area.

Amit has 19+ years of progressive leadership experience within the Software Industry and has seen significant success in delivering 'Application Development' & 'Support', as well as 'Pre-sales' for leading companies which include CapGemini and Tata Consultancy Services (TCS). One of his goals is to build consultative relationships with the Fortune 500 client base, envisioning cutting-edge business and technical solutions for clients across the U.S., Europe, India, and APAC.

Rahul is an Enterprise/Lead Architect who works with decision makers, bridging the gap between business and IT to enable digital transformation for various customers across the world. He is leading and delivering modernization initiatives including digital transformation, strategies, IT roadmaps as well as hands-on implementation of enterprise applications portfolio development for large scale transactions.

Rahul’s key responsibilities are to engage with sector experts, CoE's, CTO networks and Partner/Vendor Key Account executives across the world to follow up on emerging market/technology trends and advise on opportunities for how these trends can impact the business.

Optimizing the supply chain is interesting for Capgemini as they have many clients who would benefit from it. Capgemini's supply chain sector (CP&R) provides over €500 million of services annually, and consists of a network with more than 5,000 resources. Optimization in the supply chain could help improve their clients' infrastructure and in that sense increase Capgemini's influence.

## 1.2 Background

Society is becoming increasingly reliant on having supplies delivered in a fast and timely manner. Thus, the supply chain has to constantly evolve to meet the needs of the population and support corporate expansion. Add to this the fact that the market competition is tough, and any way to reduce cost and increase efficiency helps supply the ever-growing demand.

The supply chain is a vast area with many different parts which could be optimized. One of the most relevant parts of this is route optimization, specifically the delivery of goods using trucks. The area is currently seeing attention from worldwide corporations like Amazon and Google. Amazon, in cooperation with MIT, is even hosting a competition with the task of exploring route optimization as this thesis is written [1].

Currently, there is no best method to plan routes for delivery trucks to use. Instead, companies that handle route planning follow different solutions. Generally, this is done in one of four different approaches.

1. Driver discretion
2. Manually pre-planned route
3. Using a GPS
4. Using route planning software

As the supply chain is constantly expanding and routes are getting more complex due to the increasing number of deliveries, using route planning software is slowly becoming the dominant choice by companies in the area of route optimization. As this is happening, approaches 1 - 3 are slowly becoming inefficient and infeasible. While route planning software is seeing breakthroughs in recent years, it is still faced with concerns due to route planning's rapidly evolving complexity.

In order to plan efficient routes, information such as road restrictions (weight, height, and truck restrictions) and stops required (eating, sleeping, resting) have to be taken into account. Furthermore, if an optimal route is planned using software, traffic accidents or road maintenance can disturb the route and cause delays. This entails that another approach to the problem is needed that can handle the more complex cases.

What is needed is a method that can incorporate more key factors, real-time data, and prior routes. The solution proposed in this thesis should help delivery companies lower their transportation costs, CO2 emissions, and time spent delivering goods. This will allow the companies to save money, keep up with the growing demands, and grant them an edge over their competitors.

In order to be able to incorporate the key factors, real-time data, and prior routes, artificial intelligence (AI) and, more specifically, machine learning (ML) will be utilized. Which ML algorithms that will be used, which the key factors are, and how the data can be collected will be determined in the study.

In addition to what has been previously mentioned, the thesis work also consists of surveying which machine learning algorithm is the best to use for solving route planning problems and what key factors need to be incorporated to receive the best result. A prototype has been developed to visualize the potential of the developed ML algorithm, which offers a starting point for future work on route planning. It also provides insight into the effectiveness and potential of the chosen key factors and the ML algorithms.

## 1.3 Purpose

The purpose of this thesis is to survey how AI can be used to optimize how route planning is done within the supply chain. The expected result was to find a method that successfully utilizes AI to ensure that deliveries of goods arrive on time, at a lower cost and energy expenditure than today.

## 1.4 Goals

The thesis explored which key factors could be utilized to create a route optimization tool using AI. The thesis also determined how the data of these factors could be collected and used in real-time. Additionally, the thesis also explored which ML algorithms that suited the development of a route optimization software best based on the key factors. Finally, a working prototype incorporating the chosen ML algorithms and key factors was developed and evaluated in order to expand and discuss the information gathered during the thesis work.

## 1.5 Problem Description

In this thesis, the following questions are answered to help utilize AI in route planning solutions.

1. How is route optimization done today?
2. Which key factors can be used in route optimization?
3. Which key factors have the most significant impact on transport efficiency and utilization of fleets?
4. How can data relevant to these key factors be collected?
5. Which ML algorithm can utilize these key factors?
6. Which ML algorithm can utilize the key factors the best?
7. How can load planning be incorporated to optimize routes further?

## 1.6 Motivation

We chose this thesis topic to deepen our knowledge about AI and contribute to the optimization of the supply chain. In addition, this thesis also grants us experience within a relevant topic, which provides us with merit for our future job search.

This thesis’s subject is relevant for Capgemini as they are currently looking for ways to optimize the supply chain. The thesis focuses on an essential part of this chain and will provide insights into AI as a solution to route optimization. If done well, our thesis work could save Capgemini's clients time and money while also helping the environment.

Optimizing the supply chain contributes to the whole of society, both in the way of faster delivery of products and limiting detrimental factors to earth like vehicle emissions.

## 1.7 Limitations

The following limitations were applied to the thesis:

1. The selection of optimal ML algorithms could easily be a thesis of its own. Therefore the selection was scaled down, and the algorithms were selected based upon the initial study.

2. An endless amount of key factors could be gathered. However, many of them would have a minimal impact on performance. Therefore the key factors used were limited to only the ones that made a noticeable difference in the optimized route.
3. The prototype and the data to test it were entirely theoretical as setting up the collection of data for these would be a whole problem on its own.
4. The prototype made was not to be seen as a finished product but rather a tool to deepen understanding and contribute to the result.

# 2. Technical Background

This chapter presents the technical aspects needed to understand the rest of the thesis better. The information in this chapter does not serve to provide a deeper understanding of the topic but rather explain it to make the rest of the thesis more comprehensible. The aspects introduced range from common concepts within route optimization to algorithms able to solve the questions presented in the problem description.

## 2.1 Route Planning

Route planning as a concept appears in many ways. It can signify something as simple as finding the path from one place to another but can also mean mapping out the exact routes vehicles have to follow to visit several stops within a certain timeframe. While these problems can and have been solved by hand, computers introduce a more efficient approach.

Visualizing all roads as a node network with nodes and edges makes it possible to think of route planning as graph traversal. This problem is now solvable by computers and can introduce breakthroughs in route efficiency. UPS managed to apply this by utilizing Google Cloud Service. The service helped UPS improve their supply chain and managed to save them up to $400M and 10M gallons of fuel a year [2] [3].

The node network in question can be modeled by using intersections as nodes and combining one node to another with edges. This creates a node network able to be solved by graph traversal algorithms. Graph traversal solutions generally utilize depth-first search and breadth-first search[4]. However, these techniques tend to fall flat in modern-day route planning solutions. When the cases become more complex, such as when several nodes must be visited, the number of possible combinations becomes too many for computers to process, and different approaches have to be made. By using ML algorithms, it is possible to avoid testing every single combination in a node network, and instead create optimal routes based on the algorithm's ability to make intelligent choices.

There are two essential topics to understand in route planning: The traveling salesman problem (TSP) and the vehicle routing problem (VRP). These problems are two of the most researched problems in route planning as they handle some of the most common cases in the delivery industry.

TSP: Given a pool of nodes, where each node has a distance to all other nodes in the graph, how can all nodes be visited and then the start returned to, at the lowest cost (distance traveled) possible. In practical terms, this can be seen as a truck having to deliver goods to a certain amount of delivery points.

VRP: VRP, just like TRP, focuses on a routing problem. However, instead of just trying to figure out the fastest route for one vehicle that must visit a certain number of nodes, it does the same but with several vehicles and within a specific timeframe. Traditionally VRP is utilized to optimize fleets.

Different algorithms have been used to find solutions to these two problems. However, it is possible to wonder why software such as Google Maps cannot be used for these problems. According to [5], Google Maps utilizes graph traversal, a node network, and Dijkstra's algorithm. Because of this, Google Maps is bound by some limitations. For example, the service can only plan a route between one point and another and does not solve problems such as VRP and TSP. The reason for this being that the visitation order of the nodes matter in these problems.

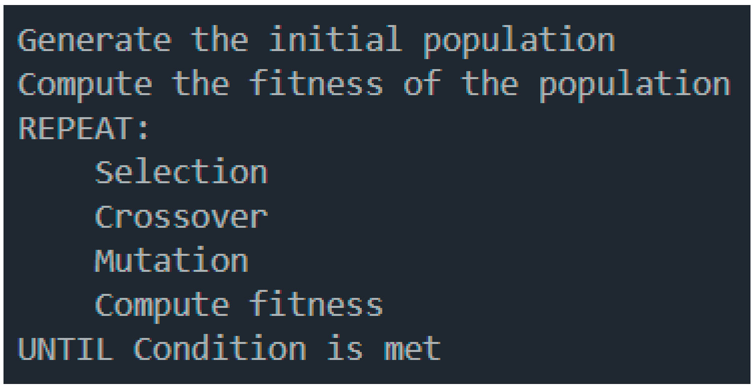
## 2.2 Genetic Algorithms

A genetic algorithm (GA) is a heuristic search that is deeply inspired by Charles Darwin's Theory of natural evolution, more specifically survival of the fittest. The algorithm aims to emulate the process of natural selection, where the fittest individuals have a much higher chance to reproduce and pass on their DNA to the next generation [6].

Genetic algorithms (GAs) can be applied to optimization problems when no standard optimization algorithm is suitable. GAs function as follows. [7]

A GA repeatedly iterates over, evaluates, and modifies a population of elements, where each element contains DNA. Additionally, each element also has a variable for keeping track of its "Fitness score." The fitness score is an evaluation of how well the element performed during the iteration. After each iteration over the population, three main steps are performed.

1. Selection - Elements (parents) from the population are selected based on their fitness score. A higher fitness score means that the element has a higher probability of being selected.
2. Crossover - Parents pair up and combine their DNA in order to generate children. As a result, each child's DNA will consist of some mix of its parents' DNA.
3. Mutation - Each child has a slight chance to have some bits of its DNA randomly changed.



```
Generate the initial population
Compute the fitness of the population
REPEAT:
    Selection
    Crossover
    Mutation
    Compute fitness
UNTIL Condition is met
```

Fig. 1. Pseudo-code for a generic GA.

When all children are generated, they form a new population and have their fitness score evaluated. The process then repeats until a specific condition is met, as shown in Fig. 1. An example of a condition could be that the fitness score surpasses 100. [7]

Even though the concept of GAs has been around since the 1950s[8], the technique is still widely used today. It is a versatile algorithm that can be applied in many different cases and be tweaked and improved to fit the desired solution better. GAs have previously been used to solve both the VRP and the TSP [9][10][11].

### 2.2.1 Implementation

When it comes to the actual implementation of a GA, some extended knowledge on implementation is needed. The elements in the population are often represented as objects in the code. These objects can then contain the required variables such as the fitness score and DNA for that individual object. The DNA is often represented as an array or list. The contents of the DNA array depend on what problem is being solved, and the behavior of the element is directly linked to what this content is. [12]

When the population is filled with elements, the algorithm can start looping through the steps described in Fig. 1. to find the desired solution.

## 2.3 Neural Networks

Artificial intelligence is a field dedicated to teaching computers to mimic intellect. There are many methods of doing so, but one of the most utilized methods is neural networks.

Neural networks can be seen as a system focused on fulfilling a purpose. Moreover, depending on the nature of that purpose, different neural networks can be utilized to solve the issue. There are many types of neural networks, the most common being perceptrons, multi-layer perceptrons, convolutional neural networks, and recurrent neural networks. [13]

### 2.3.1 Perceptron

Neural networks dated back to the beginning of the perceptron algorithm in 1958. The algorithm was invented by Frank Rosenblatt and showed potential in the field of mimicking human intellect. Rosenblatt even presented the perceptron as a way for computers to potentially walk, talk, see, write, reproduce and be conscious of its existence [14]. Although the perceptron has not accomplished all of this, it was the first-ever neural network and caused an expanding amount of research within the area.

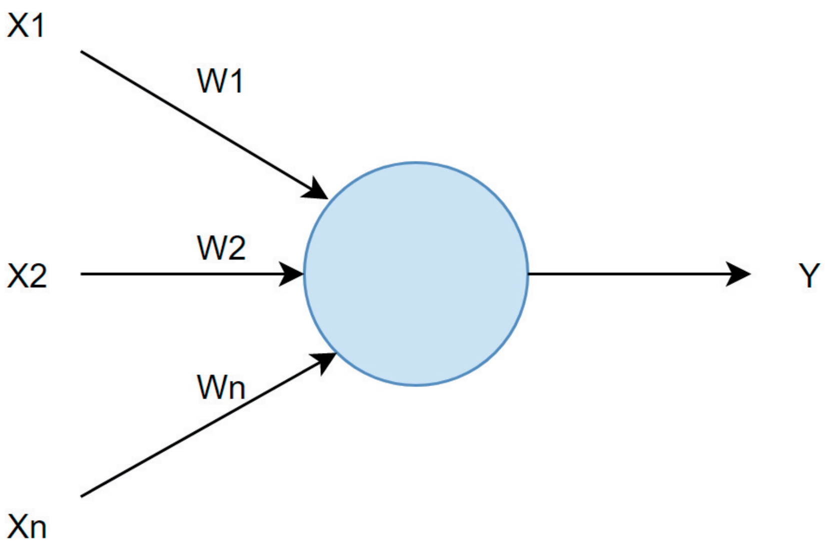


Fig. 2. Picture depicting a perceptron

The perceptron, depicted in Fig. 2, can be seen as the computational model of a single neuron[12]. The model has at least one input, a processor, and a single output. The X's represents inputs, and Y represents the output. The W's represent weights that generally are a value between -1 and 1 and define the neural network's overall behavior.

$$f(x) \;=\; \{ \begin{matrix} 1 \; if \; w \times x + b > 0, \\ 0 \; otherwise \end{matrix}$$

Equ. 1. Formula sending active/inactive signal

If the perceptron only has one input, the output Y = f(x) can be calculated with the function shown in (1). The equation can handle more inputs by expanding upon the formula, which generally is done by calculating the sum of all inputs multiplied by their weight and utilizing an activation function to receive a non-linear value within a specific boundary. Activation functions will be expanded upon later in the chapter.

### 2.3.2 Layers

Neural networks utilize a layered structure in order to recognize patterns. Every neuron can be imagined to look for a particular condition. If the condition is fulfilled, the neuron will send its signal further into the network, combining several of these signals and giving a final output.

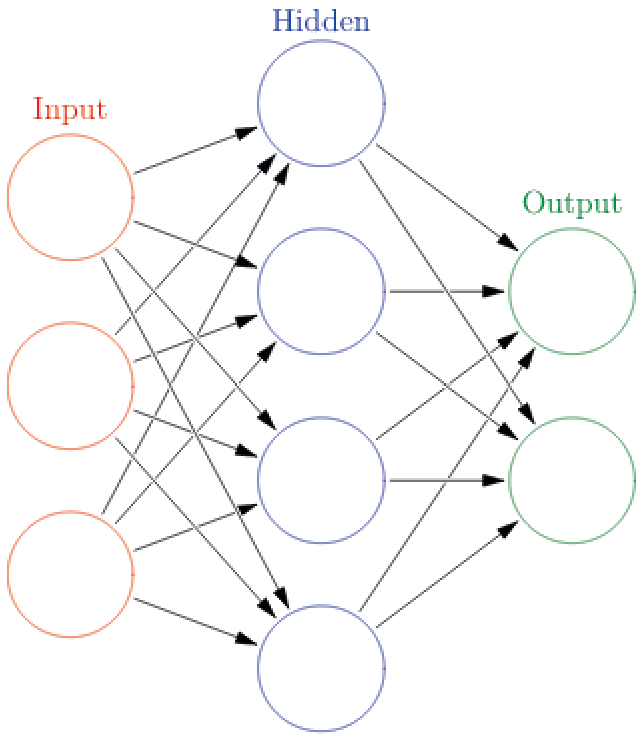


Fig. 3. A simple neural network with one hidden layer.

Fig. 3. shows how a neural network can be set up. It consists of three layers that mimic how regular neurons work together. The three layers have the following behavior.

1. Input layer - The input layer describes which data goes into the network. Generally, these inputs can be varying depending on the nature of the problem. For pattern recognition, such as utilizing neural networks to recognize handwritten numbers, the input would generally be an image split up into pixels and fed into the network [15]. However, for problems such as route planning, it is possible to consider factors such as day, time, weather, and more. Based on the information in [15], these inputs are of varying importance based on their effect on the neural network's output. Some inputs can be of great importance to increase the network's efficiency and sought-out behavior, while some can be disruptive.
2. Hidden layer - The hidden layer can consist of many layers of nodes. These layers are traditionally utilized to feed forward information in the network. The layer recognizes if specific patterns have been fulfilled and

feeds forward this information to the next layer. This process is repeated until the information reaches the output. [15]

3. Output layer - The output layer simply returns the result that the neural network decided on, given the inputs into the network.

### 2.3.3 Activation Functions

Neurons have two states: active or inactive, signified by 1's or 0's in computer terms. Activation functions are generally utilized to mimic this behavior and introduce non-linearity into the system [16]. By converting inputs to numbers between a lower boundary and upper boundary, the signals are converted into easier data for the network to utilize for training.

Activation functions can have varying ranges and varying efficiency in neural networks. While there is no solid theory behind choosing an activation function [15], some activation functions are still more popular than others as they can have a better probability of being applicable to different problems.

$$ReLU(x) = max(0, x)$$

Equ. 2. Activation function returning a value between [0,∞]

Equation (2), shows the ReLU activation function, which is the most widespread activation function and has proven to be both efficient and consistent[17]. This means that ReLU has proven to help train a varied range of neural networks faster than most of its predecessors, such as the sigmoid function.

### 2.3.4 Feedforward and Backpropagation

Most Neural Networks utilize feedforward and backpropagation. Feedforward is a concept very commonly used in neural networks. It is the process of input data traveling through the system to the output without looping any of this information back through the network. However, some networks, such as recurrent neural networks (RNN), utilize a loop of information to solve problems of sequential nature [18].

While the feedforward process travels from the start to the end of the neural network, backpropagation does the same but reversed and with additional functionality. Backpropagation is the process of teaching the neural network what changes it needs to make to its weights. This is generally done by changing the neural networks' weights based on a cost function and gradient descent.

### Cost function

$$C(w,b) = \frac{1}{2n}\sum_x |y(x) - a|^2.$$

Equ. 3. The MSE cost function, often utilized in neural networks.

Cost functions are utilized to calculate an error rate by comparing the neural networks' results with the expected result. The error rate is later used to update the weights in the neural network. There are many different Cost functions, and one of the most used is the quadratic cost function, also known as the mean squared error (MSE), which is shown in function (3).

### Gradient descent

Gradient descent and cost functions build upon each other. The goal of any neural network that utilizes cost functions is to minimize the error rate. By doing so, the neural network's behavior aligns with the intended purpose of the network[15].

When we have a function that needs minimization, an algorithm that can solve this problem is the next line of action, and this is where gradient descent can be introduced. Gradient descent focuses on finding which direction the function has to go in order to minimize the cost as fast as possible[15].

### Process

The entire process of having inputs feed forward through the system and then the system backpropagating can be summarized as follows, based upon [15]:

1. Input data is fed into the system and is fed forward towards the output layer.
2. The output is compared with the expected result by using a cost function.
3. The cost is calculated by utilizing derivatives to see how much of an effect the change in weights have on the intended result.
4. In order to optimize the previous step, gradient descent is utilized to figure out the most optimal way of changing the weights for the intended result.

## 2.3.5 Learning Algorithms

For neural networks (NNs) to learn, they must know what is wrong and what is right (or what is desired and what is not). There are several different learning algorithms available, and the three most utilized ones are the following, as described in [19].

### Supervised Learning

Supervised learning is the means of inputting labeled data into the NN to train it. The labeled data consists of the input that the NN will take and the desired output that the NN should give. Labeled data is also known as “training data” when used for training and “testing data” when used to test the network’s ability to classify data correctly.

The labeled data will go into the NN, and the NNs output will be compared with the desired answer. The error that this produces is calculated using a cost function and is used in conjunction with backpropagation to teach the NN the correct answer.

### Unsupervised Learning

Unsupervised learning, on the other hand, only provides the NN with an input. No other direction is given to the NN. Since there is no labeled answer to compare with, no backpropagation or tweaks to the weights can be done.

As a result of this, the NN cannot “learn” how to correctly classify the data, but rather it orders the data into groups based on similarities and patterns in the data.

Another approach is to utilize a reward system. By using this reward system, the NN can be rewarded or punished based on the actions it performs.

### Reinforcement Learning

Reinforcement learning has many similarities with GAs. When using reinforcement learning, the NN is trying to maximize some fitness score based on how well the NN performs. After the NN has been evaluated, crossover and mutation can be applied to evolve the NN, where a NN with a higher fitness score has a higher chance to reproduce.

## 2.3.6 Deep Learning

Deep learning can be seen as a subset of neural networks (NNs). Deep learning incorporates a NN and expands and enhances it by adding more hidden layers to the network. This expanded NN that features multiple hidden layers is called a deep neural network (DNN). The word "deep" stems from the fact that it is several hidden layers deep.

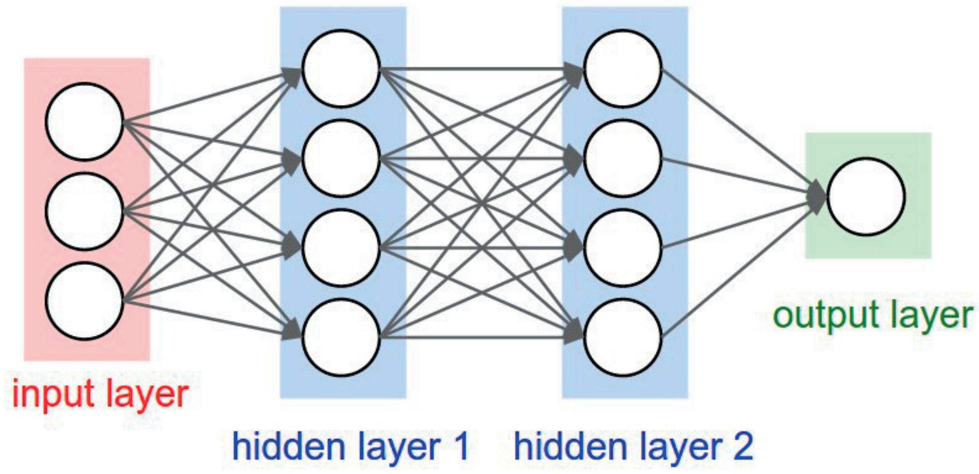


Fig. 4. A DNN featuring multiple hidden layers.

As shown in Fig. 4. a NN does not need more than two hidden layers to be classified as a DNN. However, the more hidden layers the network has, the deeper it is [20].

This structure causes deep learning to be a powerful algorithm with many benefits. For example, it can handle complex problems such as speech recognition, image classification, and natural language processing[21].

There is no easy solution or optimal method for finding the optimal number of hidden layers and nodes per layer. However, a few different main approaches can be utilized to solve this problem, "neuro-evolution" being one of them. [22]

## 2.4 Neuro-evolution

Neuro-evolution can be seen as a mix between neural networks (NNs) and genetic algorithms (GAs). It is a ML technique that uses the concept of GAs to generate and optimize NNs. The neuro-evolution algorithm will take into consideration the architecture and topology of the NN and try to optimize it to achieve better performance (fitness)

Neuro-evolution can be used together with deep learning to help optimize the NNs depth, and thus a more efficient deep learning algorithm can be obtained [23].

As mentioned in subsection 2.3.4, most NNs use gradient descent as their backpropagation method (training the NN). However, recent studies have shown that simple GAs can train DNNs with millions of parameters to outperform NNs trained with modern techniques and perform the training faster as well [24].

### 2.4.1 NEAT Algorithm

NEAT stands for "Neuro-Evolution of Augmenting Topologies" and is technically a GA. According to [25], the algorithm aims to evolve NNs by changing weights and the networks' topology. Since NEAT is a GA, it follows the same steps as mentioned in the GA section 2.2.

The algorithm starts with a population consisting of minimal NNs. A minimal NN is a network that only consists of input and output nodes, no hidden layer, and no connections between the nodes. During the crossover and mutation phase, the algorithm gradually adds new connections and nodes to the NNs (thus augmenting the topology) and changes the weights between the connections.

Like other GAs, the algorithm repeats its evaluation, crossover, and mutation phases until a condition is met. That condition could be that an optimal NN has been found or that no improvement to the NN can be found.

NEAT has specific methods in place that are used during the different stages of the algorithm to ensure that during every mutation and crossover step, no progress is lost and the algorithm is evolving in the right direction.[25]

## 2.5 Libraries

A library is a collection of functions containing written and compiled code. Libraries are used to lower time spent on reinventing functionality, and allows for easy access to code that can be used or expanded.

### 2.5.1 Cytoscape.js

Cytoscape is a graph library with functionality to initialize, traverse and visualize node networks consisting of nodes and edges. The functions in the library are optimized, and the library is compatible with most browsers. The library is made for JavaScript and can therefore easily be visualized in HTML.

The library is open source and is used by several worldwide companies. The complete documentation and user description can be found on their website.[26]

### 2.5.2 Neataptic.js

Neataptic is a library that offers NNs with many options to modify and functions to use. The library contains all functions needed to initialize, train, evolve, and use NNs. Neataptic uses neuro-evolution to evolve and optimize the topology of the NNs. Since Neataptic is made for JavaScript, it also offers a function to display the created NNs in HTML, which makes it easy to visualize the networks.

The library is outdated, as it is no longer maintained. Thus the library contains a few known bugs and problems that need to be worked around. The complete documentation can be found on the creator's GitHub page. [27]

# 3. Method and Analysis

The method of the thesis work consisted of four main phases, with each phase consisting of a few steps each.

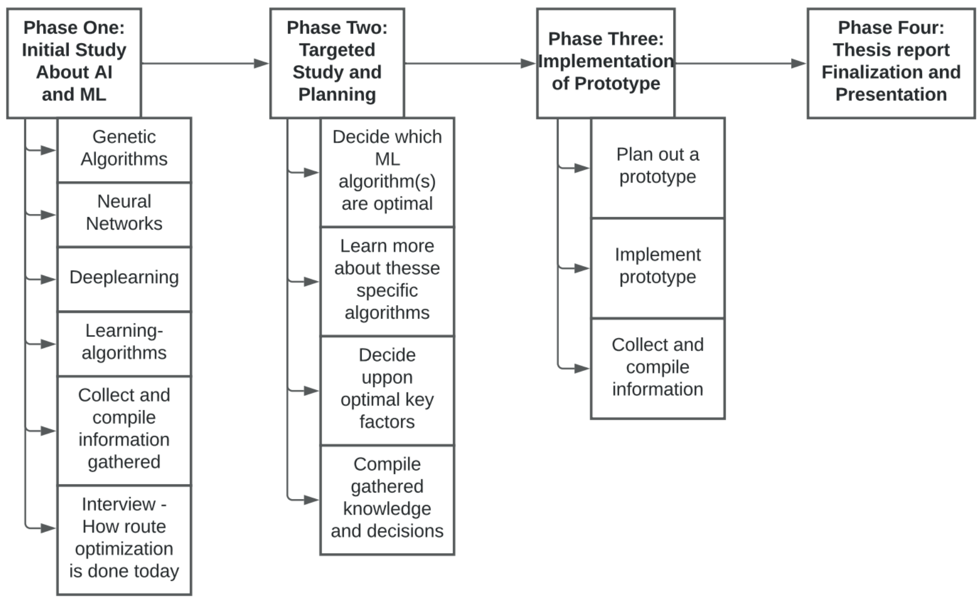

Phase One: Initial Study About AI and ML
Genetic Algorithms
Neural Networks
Deeplearning
Learning-algorithms
Collect and compile information gathered
Interview - How route optimization is done today
Phase Two: Targeted Study and Planning
Decide which ML algorithm(s) are optimal
Learn more about thesse specific algorithms
Decide uppon optimal key factors
Compile gathered knowledge and decisions
Phase Three: Implementation of Prototype
Plan out a prototype
Implement prototype
Collect and compile information
Phase Four: Thesis report Finalization and Presentation

Fig. 5. Sketch showing the phases and steps that make up the thesis work

The phases and steps can be seen in Fig. 5. The phases were worked through one at a time, completing all steps in a phase before moving on to the next one, and each phase has a section in the current chapter.

The thesis work started with an initial study to learn more about machine learning (ML) in general. Afterward, in phase two, more specific knowledge was gathered regarding the algorithms and key factors that were deemed optimal for the thesis work. The gathered knowledge was then put to use in phase three, where a prototype was created to test the eligibility of the algorithms and key factors that were decided on in the previous phase. Finally, in phase four, the thesis will be finalized.

However, before the phases started, some planning was done, as described in section 3.1

## 3.1 Planning and Work Structure

The whole thesis work process began with reaching out to Capgemini and booking a meeting. At the meeting, discussion ensued, and it was decided that a thesis would be cooperated upon. Capgemini wanted the thesis topic to be focused on the optimization of the supply chain. As a common interest, it was decided that the application of machine learning would be involved. A couple more meetings were booked to determine the specifics, and after some discussion, the topic of this thesis was decided.

With the specifics out of the way, the initial background, Chapter 1, was written and approved by the supervisors at Capgemini and LTH, and planning of the thesis could commence.

At this point, the creation of the timeline for the thesis work was set in motion, and certain decisions were made that would have a direct impact on the timeline. It was decided that the thesis work will follow the waterfall model in the sense that every step would be pre-planned and finished before moving onto the next step [28]. However, upon agreement with the supervisors at Capgemini, it was also decided that the prototype would be worked on according to an agile model and that the model would be specified when the prototype phase began.

The waterfall model was chosen because each subsequent step of the thesis required a fundamental understanding of the previous steps. The model enforced this by requiring pre-planning, which helped create a logical structure of the thesis work.

By working on the prototype according to an agile model, a few benefits were seen. Knowledge gathered during the prototype phase was utilized to introduce changes in how the prototype was worked upon. Additionally, it made sure there was always something to work on, allowing time to be distributed effectively during the phase.[29]

Because of the ongoing pandemic and governmental restrictions, it was decided that all work on the thesis would be done from home. This means that all communication was done digitally, including communication with the supervisors at Capgemini and the university.

To make the communication with the supervisors at Capgemini as fluent as possible. Bi-weekly meetings were pre-scheduled for the entire duration of the thesis work. In addition to this, the supervisors made it clear to mail them with any issues that would surface during the thesis work. A similar method was to be followed regarding the communication with the supervisor at the university. Meetings were to be scheduled in advance, and any questions or ideas that appeared before the meetings could be discussed over email.

All communication between the thesis workers took place in Discord [30]. Discord allows for communication using both voice and text-chat and allows for managing relevant information in different channels.

This structure for communication worked well for all parts involved, and much time was saved by not having to gather at a specific location to meet up in person.

Work on the thesis took place five days a week, starting at 9 o'clock, for at least 6 hours per day. This led to the thesis workers working from 9-15/16 most of the days, working primarily on weekdays but sometimes moving a workday to the weekend if needed.

When working on the thesis, information was stored in Google Drive and Discord. Discord stored information that could be interesting to look into, such as articles and different GitHub projects. If the information in the Discord channel was worth mentioning in this report, the information was added to Google Drive. Either by writing a new document with the relevant information that could be used as material when writing the thesis or as notes inside the thesis. This allowed information to be managed efficiently and made it possible to distinguish important information from non-important information.

During the entire thesis, all information was studied and learned together by the thesis workers. This was done in order to share knowledge between both parties through communication.

## 3.2 Thesis Work Timeline

A timeline, devised as a GANTT chart, was created for the thesis in order to structure the work that has to be done. The timeline also served to grant an overview of the project. Furthermore, the timeline served as a schedule that could be used to make sure that all work was done in time. The timeline was divided into four phases, which can also be seen in Fig. 5.

Phase 1. Initial study about AI and ML
Phase 2. Targeted study and planning
Phase 3. Implementation of prototype

Phase 4. Thesis report finalization and presentation

As specified in section 3.1, the waterfall model was chosen, which is represented in the timeline. The phases were selected with the model in mind by allowing for a gradual build-up of knowledge through each phase. The results and findings from each previous phase were built into the next one while also limiting stagnation by distributing time effectively through proper planning.

When devising the timeline, work was done to determine which ML techniques should be considered to learn more about during the first phase. After reading previous work on route optimization, the techniques that were decided upon were: genetic algorithms, neural networks, and deep learning. These were chosen since they had shown promise within the field previously, and served as a useful starting point for learning ML.

The final thesis timeline was followed throughout the entire thesis work, and no changes to the phases or steps were made. An aspect that could have been improved was to have a more prolonged prototype phase. However, for the thesis to come to a result, limitations had to be made. Overall the timeline is deemed to have been satisfactory. The thesis plan can be seen in full here [31].

As specified in the introduction to this chapter, the phases are divided into several steps. The steps that phase 1 consists of focus on learning about the ML techniques previously decided upon, which is done to have a solid foundation of knowledge to build upon. The phase also consists of documenting the information learned and conducting interviews regarding route optimization.

The steps in phase 2 consist of deciding the optimal ML algorithms for route optimization, learning more about the chosen algorithms, deciding upon optimal key factors, and documenting all information and decisions of the phase. The steps were selected to gain a deeper understanding of ML techniques that can be used in cooperation with key factors while developing the prototype.

The steps in phase 3 consist of planning, implementing the prototype, and documenting any conclusions or results retrieved during the phase. The steps were chosen to construct a logical order for implementing the prototype during the phase. Not much time is allocated for planning the prototype because the implementation requires experimentation with the ML algorithms, which is why this phase worked on following an agile model.

Phase 4 is time set aside for finishing the thesis and preparing for the final presentation of the thesis. The phase does not contain any steps, and therefore it will not be discussed in the methodology.

## 3.3 Initial Study

The first phase of the thesis work was the initial study. Its purpose was to provide a fundamental understanding of how machine learning works, making sure that no misunderstandings follow into the other phases of the study.

The phase consists of six steps. The first four of these steps cover learning about different ML algorithms and concepts. The last two steps cover collecting and compiling information and conducting interviews on how route optimization is done today. How these steps were conducted will be explained in this section.

### 3.3.1 Information Gathering

The phase started with information gathering, focused on finding a varied range of sources to learn from. The first four steps of this phase cover learning about different ML algorithms and concepts. The time had to be spent efficiently by gathering knowledge, from basic information to scientific articles that properly relate to route planning and ML.

Three main ways of gathering resources were used. First, LTHs courses on ML were surveyed to see if they have any recommendations for ML material. Second, Youtube was used to gather sources, as many videos on the topic of genetic algorithms, neural networks, and deep learning can be found. Mostly videos recommended by professors or uploaded by renowned universities were used to gather information, but also videos uploaded by people who want to share their knowledge on ML. Third, articles on route planning solved with ML were looked for by using Google Scholar and LUBSearch.

### 3.3.2 Learning

Many sources were gathered in subsection 3.3.1, and a process of gradually going through them was needed, which led to the following process:

1. Youtube - Used to gather an initial understanding of a topic.
2. Books - To skim through and gain a more nuanced understanding.
3. Articles - To understand how route planning and machine learning can be intertwined.

Every week that was spent learning about a specific technique of machine learning ended with documentation of different critical points of these techniques. The information was documented in Google Drive and included: how the algorithm works, how it can be implemented in practical terms, and how it can be utilized for a route planning solution. This allowed for planning into later phases, for instance, giving a general idea of implementation for the prototype. But also to document specific techniques in chapter 2 of this thesis.

## 3.4 Targeted Study and Planning

The second phase of the thesis work was based on preparing for the prototype and getting a better understanding of specific algorithms that can be implemented in the prototype.

This phase consists of four different steps, evaluating ML techniques and looking into which techniques to utilize for the prototyping phase, learning more about the techniques that will be used in the prototype, deciding upon optimal key factors, and compiling the information gathered and any decisions made. This section of the thesis will cover the process behind these steps.

### 3.4.1 Evaluation of ML Algorithms

There are many different ML algorithms, and it would be impossible to study every algorithm's usability within route planning. Because of this, the evaluation starts by limiting this amount.

Initially, this was done by studying commonly used ML algorithms and concepts that could potentially be implemented in the prototype and adding them to a list. How common the algorithms and concepts are were based upon how widespread their usage is.

The population of the list was mostly done using different books and articles such as these sources. [12] [32] [33]. Less commonly used algorithms were also added to the list if they have previously been utilized in graph traversal solutions. At this point, the list contained the following algorithms and concepts.

- Genetic algorithm
- Linear regression
- Binary decision trees
- Neural networks
- Deep learning (DL)
- Recurrent neural networks (RNN)

- Ensemble learning
- Ant colony optimization (ACO) algorithms

Filtering of this list was done after a more extensive study regarding what the listed algorithms were used for. For example, some algorithms have not been utilized in optimization problems and graph traversal previously and were therefore removed from the list. This was done since these algorithms have existed for a long time and would already have been used for route optimization problems if possible.

While learning more about the concepts and algorithms in the list, another way to teach neural networks was found. Combining genetic algorithms (GAs) and neural networks (NN) introduces a new concept called neuro-evolution, which has shown promise in optimizing NNs. Thus it was added to the list.

Since both neuro-evolution and deep learning are implemented using NNs, it was redundant to keep NNs in the list as a concept. Therefore NN was removed from the list.

Table 1. was created to evaluate the different algorithms. The criteria used to assess each algorithm are based upon if the algorithm can be used for graph traversal, optimization problems, or route optimization, if it has been used for route optimization, and if the algorithm can somehow be expanded upon.

If the algorithm can be expanded upon is one of the more important criteria. Since if the algorithm has no potential of expansion, working with the algorithm would not contribute with any new knowledge to the field of route optimization. Instead, the algorithm and previous findings regarding the algorithm should be able to be expanded upon for it to be valuable to the thesis.

Table 1. The evaluation table used for deciding upon which algorithms to consider. The boxes with green color and an X signifies that a criterion is fulfilled, while the boxes with red color and no marking indicate the opposite.

| **Algorithm** | **Can be used for graph traversal** | **Can be used for optimization problems** | **Can be used for route optimization** | **Has been used for route optimization** | **Previous findings can be expanded upon** | **Can utilize most key factors as input** |
|---|---|---|---|---|---|---|
| **Genetic algorithm** | X | X | X | X | X | |
| **Linear regression** | | | | | | |
| **Binary decision trees** | | X | | | | |
| **Neuro-evolution** | X | X | X | | | X |
| **Deep learning** | X | X | X | X | X | X |
| **Recurrent neural networks** | X | X | X | X | X | X |
| **Ensemble learning** | | X | X | | | |
| **ACO algorithms** | X | X | X | X | | |

Table 1. was filled out by the following process:

1. Studying the algorithms and figuring out how they work and what they traditionally are used for.
2. Using google scholar and LUBSearch to search for the specific algorithm with keywords such as optimization, route optimization, route planning, and graph traversal.
3. The "Previous findings can be expanded upon" criteria was filled by browsing conclusions from different articles and verifying if the authors believed there were findings to expand on, and by thinking about ways to potentially apply/improve the algorithms' ability to solve route planning problems.

The algorithms that remained in the list were the ones who fulfill most of the criteria and can also be expanded upon, which are:

1. Genetic algorithm
2. Deep learning
3. Recurrent neural networks

However, suppose the algorithm has not been used before within the field of route optimization. In that case, this will correlate to it being impossible to expand on the algorithm as well since expansion signifies that it has already been tested prior. However, this would not necessarily be negative since it would allow for exploring an algorithm that has not been tested previously. Therefore neuro-evolution was still considered a prime candidate and was added to the final list.

The final list of algorithms to be implemented in the prototype, which will be referred to as "the final algorithm list", is shown below:

1. Genetic algorithm
2. Deep learning
3. Neuro-evolution
4. Recurrent neural networks

Having a list with multiple algorithms made it possible to explore several solutions in this thesis work, which was thought to correlate to a more diverse study and limit problems such as stagnation if the study cannot easily be expanded upon.

### 3.4.2 Targeted Study of Algorithms

To prepare for the prototype, decisions regarding how the algorithms in the final algorithm list would be studied had to be made. Thanks to earlier planning, the final algorithms aligned with what had been previously studied in phase one. With this in mind, less time could be spent on learning the theoretical parts of the algorithms, and more time could be spent focusing on how they were implemented in practice.

To learn the general approach of implementing the algorithms, the initial plan consisted of two steps. The first step was to watch Youtube videos implementing the techniques. This was done to gain an understanding of the step-by-step process to follow when implementing the algorithms and pick up any general tips that would be communicated through the videos. An example of this can be seen here [34].

Step two was to move on to browsing through projects on GitHub and reading their code to more quickly gain an understanding of how certain problems are solved. Many types of problems were looked into. However, GitHub projects closer to the topic of route optimization with ML were prioritized.

### 3.4.3 Key Factors for Route Planning

The list of key factors in this subsection was made by looking at route planning solutions [35] [36] [37] and by brainstorming potential key factors based on the experience acquired during phases one and two. This list, sorted in alphabetical order, will be referred to as the “key factor list”.

- Break frequency for the trucker - Helps to plan when a spot for a break needs to be passed.
- End node - An end node is required for every route.
- Nodes to visit - If several nodes are to be visited, these have to be included as input.
- Road angle - How much of a slope the road has, impacts fuel usage.
- Road availability - If the road is often closed, it becomes a risk when dealing with timely deliveries.
- Road conditions - A poorly maintained road can cause damages to the vehicle; snow and rain on the road will affect traveling speed.
- Road length and speed limit - Can be used to calculate how long a road takes to travel.

- Service areas - The planned route has to take service areas into account during long routes.
- Start node - A start node is required for every route.
- Temperature - Cold conditions can raise the risk of accidents. If the driver has to drive in hot conditions, then taking breaks and having time to drink water is essential.
- Time of day the road will be traversed - Can be used to identify rush hours.
- Truck capacity - Can be used to incorporate load planning.
- Weather - Can cause effects such as roads shutting down, diminishing speed for deliveries, and risk for accidents.
- Which day the road will be traversed - Can be used to predict traffic patterns such as an increase in traffic during holidays.

The intention was to evaluate all key factors while testing the prototype.

### 3.4.4 Collection of Key Factors

Since it was impossible to retrieve real data for all key factors, theoretical values had to be applied to the prototype regarding these factors. Some key factors where real data could be collected used theoretical values, to begin with. This was done to focus on getting the intended behavior of the prototype working.

Methods for retrieving data relevant to the key factors:

Amazon - Amazon is, as of writing this thesis, hosting a routing challenge where they want contestants to build an AI solution for routing. To help the contestants train their ML models, Amazon is distributing data regarding previous deliveries they have done. This data includes historic routes, the distance traveled, and the time it took. This will be leveraged in order to receive realistic routes for the prototype to train with.

OSM - OpenStreetMap can be used to generate a node network from a map.

Cytoscape - Can be used to create and display node networks.

Trafikverkets API [38] - Can be used to obtain information regarding roads, weather, and other conditions.

These methods were kept in mind while developing the prototype in case their application was deemed useful.

### 3.4.5 Collection of Key Factors for Real-world Application

During the process of collecting key factors, different methods for collecting key factors were found, and the methods were documented in Google Drive. Certain choices were looked into, such as Trafikverket's API, Control towers, and others. The results from these were also documented in Google Drive and were leveraged to answer questions in the problem description.

## 3.5 Implementation of Prototype

To answer the questions in the problem description, a prototype that allowed for visualization and efficient testing was needed. Planning went into the prototype phase to make it possible to test multiple algorithms. This allowed for comparison between different algorithms and increased the possibility of finding a solution to the problem described in chapter one.

The purpose of the prototype was to gain experience with the algorithms and experiment with them to answer several of the questions in the problem description. The prototype approached this by first checking the algorithms' ability to traverse node networks. As this quality was confirmed, more complex route problems could be tested, such as TSP and VRP. The algorithm's ability to use key factors was confirmed by assessing how the key factors were incorporated into the algorithm and by monitoring their effect on the algorithm's performance.

The prototype phase consisted of planning and implementing the prototype as well as compiling any information or decisions made during the phase. This segment covers the process behind how these steps were fulfilled.

### 3.5.1 Prototype Ideas

Based upon the algorithms in the final algorithm list in subsection 3.4.1, ideas for potential prototypes were devised. These ideas were brainstormed and will therefore be accommodated with a proper explanation as to why they were chosen. The prototype ideas that were devised are as follow:

#### NEAT

NEAT allows for a solution that can take all the key factors from the key factor list into account. Either as inputs or general guidelines for the node network.

Since deep-learning, NNs, and GAs have been used previously to solve route optimization problems, they are known to work. However, the use of neuro-evolution has not been as deeply explored when it comes to route optimization. Nevertheless, since the potential is there, and route optimization using neuro-evolution has not been widely researched before. Therefore, NEAT would be a prime candidate for solving route optimization problems.

#### GA with NN

While NEAT utilizes a GA architecture to evolve and augment its NN, it is also possible to combine both GAs and NNs with a different approach. The envisioned algorithm is based upon the idea of using a NN to rate a road's probability of being the optimal choice to reach the end node. By using supervised learning, it is possible to use the NN to take a multitude of key factors into account. The NN can then be trained to a point where it can present a statistical probability of how prime a road is to traverse in order to reach the end node. The GA then uses the calculated probabilities to travel through the node network. This allows the algorithm to potentially find better solutions than if NN was not used initially.

#### RNN with reinforcement learning

RNNs are built upon NNs and allow for solving problems of sequential nature. As route planning can be seen as a sequential problem, since routes tend to be based upon prior routes, RNNs could show promise in the field. If reinforcement learning is incorporated, it would be able to modify its weights based upon how well the network performs. Since RNNs utilize NNs, they would also be able to use the key factors from the key factor list.

### 3.5.2 Prioritization of Algorithms

Ideally, all the ideas listed in subsection 3.5.1 would be able to be implemented and tested. However, if this could not happen because of the timeframe of the thesis work, prioritization had to be made to counteract this problem.

The prioritization was made following this reasoning. NEAT has not been explored previously but follows a NN structure which likely would lead to fast implementation and easily verifiable differences between itself and other NN structures. It was therefore chosen as a starting point for the prototype implementation. GA with NN was put second, as NEAT and RNN share the same structure, both being different ways to implement NNs. GA with NN was assumed to be hard to implement. Still, since it differentiated itself and because GAs were deemed successful in solving difficult optimization problems, it was put second in prioritization.[39] RNN was put last to ensure that there was something to work on if the first two algorithms stagnate in performance.

The prioritized list was thus the following:

1. NEAT
2. GA with NN
3. RNN

This led to an initial plan of dedicating two weeks to NEAT and the following two weeks to GA with NN. RNN was seen as a substitute if the first two implementations did not go the intended way because of stagnation. The reason time was distributed this way was because the entire phase stretched four weeks, and it was presumed that after two weeks, enough information would be retrieved to answer questions in the problem description.

### 3.5.3 Tools

Several tools were used to simplify the implementation of the prototype.

Initially, three different programming languages for implementing the prototype were considered. These were JavaScript, Python, and Java. These were considered because of a few reasons:

- They are well-known languages used by many developers, which means that help and libraries can be found online with ease.
- They were the main languages used in other articles and academic texts that were read in preparation for the thesis work.
- The thesis workers have previous knowledge of these three languages.

However, it was finally decided that JavaScript would be used. The main reason being that it was the language that the thesis workers had the most experience with, and Node.js being convenient to use for utilizing external libraries. Additionally, JavaScript made it easy to visualize the results using HTML. More information on what JavaScript is and how it works can be found in the following sources[40][41].

Node.js is a JavaScript runtime designed to build scalable network applications. It was used while creating the prototype to simplify the process of installing and using libraries. Further reading regarding Node.js can be done on the official Node.js website[42].

Visual Studio Code (VSC) was chosen as the IDE to use while implementing the prototype. It was chosen because it is powerful while still being easy to use. VSC features a wide library of addons that can be used, which contributes to pair-programming and development being a smoother experience. The IDE also features a built-in GUI for using GIT, which is more accessible than using the terminal.

Two add-ons were used in VSC, namely “Live Share” and “Live Server.” Live Share is used because it allows multiple developers to simultaneously edit and write code inside the same document. Live Server was used since it automatically sets up a local server where the code is deployed each time a document is saved. This simplifies the processes of writing HTML and CSS since there is instant feedback on the code written.

Since JavaScript is a programming language that many developers use, many libraries exist that can be used to ease the workload. Two libraries were chosen to be used in the prototype.

Neataptic.js was chosen because it has efficient NN and neuro-evolution algorithms built into it and offers the possibility to display the NN on an HTML page. It is also the library with the most active users out of the ones investigated.

Cytoscape.js was chosen because it offers functions to set up, visualize, and later access a graph consisting of nodes and edges. The selected libraries can be read about more in-depth in section 2.5.

### 3.5.4 Work Structure

The algorithms specified in subsection 3.5.1 were decided to be implemented using a similar structure of work. It was specified in section 3.1. that work should be done according to an agile model. Scrum was chosen as the agile model, and all principles were followed with the extensions of using pair programming for the entire duration of the prototype.

When working on the prototype, the following principles were followed:

- Setup any libraries needed and troubleshoot any immediate problems before starting.
- Try solving an easy problem first to become familiar with the libraries.
- Pre-plan every part of the implementation to make sure that a direct line of thinking can be followed.
- Start solving a simple route planning problem and add more functionality in increments.

Other than following these steps, the following points were also taken into consideration during every part of the prototype:

- Store all written code in a GitHub repository.
- Document every finding in the thesis either as notes or fully explained paragraphs.
- Continuously find and record problems with the prototype and document them.
- Send a daily email to the supervisors at Capgemini containing: What was accomplished yesterday, What will be accomplished today, What are the current impediments.

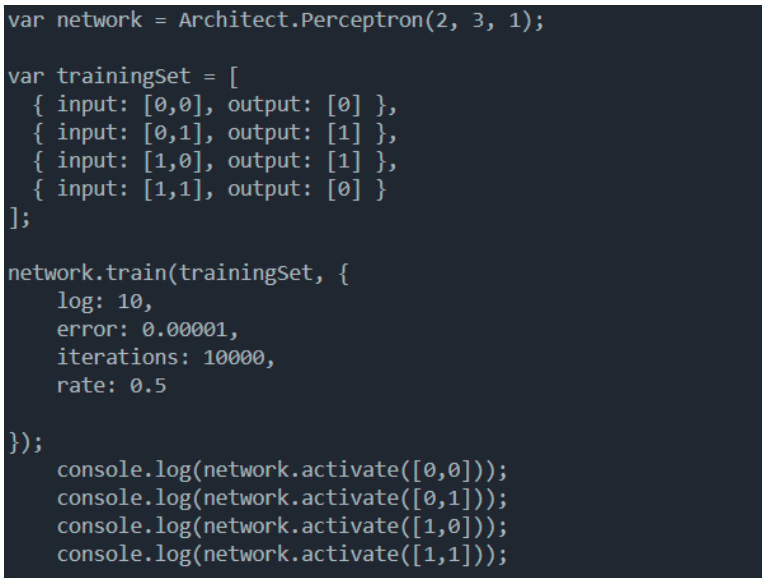
Every two weeks of implementation led to a meeting where the findings and the current prototype were presented to the supervisors at Capgemini. This was done by preparing documents, screenshots, concepts, and code to show. Feedback was then received, and future plans for the line of work were discussed.

### 3.5.5 NEAT

Based on the prioritization specified in subsection 3.5.2, NEAT was the first algorithm to be implemented. However, before this could be done, some initial preparations were needed.

The workspace was initially set up containing HTML- and JavaScript files and Node.js. However, this led to some unexpected problems. Node.js could not communicate well with the front-end client-side application. More specifically, the imported libraries could not be reached client side. Solutions to the problem were found on Stack Overflow, and it was decided to run the code through the browser instead of using Node.js.

When the libraries were imported, getting acquainted with how the libraries work was the next step. Neataptic was looked into first, as a basic understanding of how the library works was required to start working on the route planning solution. This experience was gathered by training a NN to solve the XOR problem, a typical example NNs are trained to solve.



```
var network = Architect.Perceptron(2, 3, 1);

var trainingSet = [
  { input: [0,0], output: [0] },
  { input: [0,1], output: [1] },
  { input: [1,0], output: [1] },
  { input: [1,1], output: [0] }
];

network.train(trainingSet, {
    log: 10,
    error: 0.00001,
    iterations: 10000,
    rate: 0.5

});
    console.log(network.activate([0,0]));
    console.log(network.activate([0,1]));
    console.log(network.activate([1,0]));
    console.log(network.activate([1,1]));
```

Fig. 6. A neural network trained to solve the XOR problem.

Fig. 6. shows the implementation of a solution to the XOR problem using the neataptic library. This was planned to be done using the NEAT algorithm. Still, since supervised learning fits the problem better, the focus was directed to learning the library and testing the different ways to manipulate the NN.

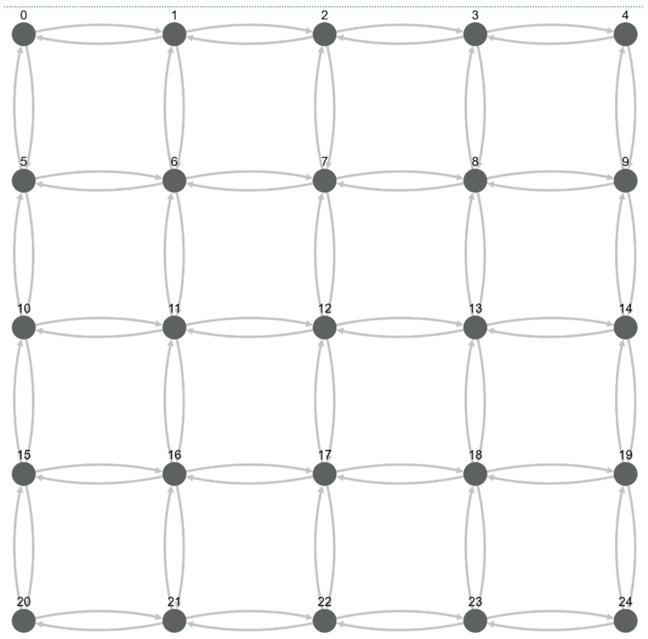


Fig. 7. Node network featuring nodes and edges in a grid layout.

The next step to start working on the route planning solution was to create a node network, as seen in Fig. 7. To make it harder for the NN to find paths in the node network and make the node network more similar to a road system in the real world, some way to make the optimal route between two specified nodes less straightforward had to be implemented. This was done by giving every edge a random cost to travel, making it possible for the best path to consist of more nodes while still being the cheapest path. With this structure, the NEAT algorithms' proposed route through the system could easily be compared with a route calculated by Dijkstra's algorithm to see how well the NEAT algorithm was performing. It is also worth noting that the node network was created with the ability to scale up the number of nodes in the network easily.

With the node network created, it was displayed on the HTML page to allow visualization of the network. This made it possible to visualize any paths created when the solution started to produce a result.

With all the needed preparations made, implementation of the NEAT algorithm started, and the first goal of making the NN pick a path between two nodes in the network went underway. However, before implementing the solution, decisions regarding which inputs would be used and how the output of the NN would be utilized had to be made.

To decide the inputs to the neural network, a precise plan of implementation had to be specified. This plan was based upon the initial idea specified in subsection 3.5.1, and with this in mind, three specific approaches to the problem were planned.

The first approach was to think of the solution to the problem as a sequence. Every node in the network has an optimal edge to travel in order to reach the destination. With this in mind, the NN could potentially be used to evaluate which the best edge to reach the destination is and solving the problem with the following steps:

1. Select a start and end node
2. Set current node = start node
3. Evaluate all edges from the current node with the NN
4. Select the edge that the NN classifies as most optimal
5. Set current node to the destination node of the edge
6. Loop steps 3 - 6 until the end node has been reached

The second approach was to feed a neural network all relevant information for a certain node, such as its edges, their costs to travel, and its destination. These could then be used to create an output of which node the network should travel to.

The third approach has been explored before [43] and was to train the NN using supervised learning. This would allow the NN to be trained with paths generated by Dijkstra's algorithm and would provide the NN with a general idea of how to find the best paths in node networks given the correct inputs.

By comparing the different approach against each other, it was decided that the first approach would be implemented because of the following reasons:

- Approach one can easily use attributes of an edge as inputs. For instance, factors such as traffic conditions, weather, and road condition, could be easily retrieved and evaluated by the NN.
- Approach two follows the same idea as approach one, but with more uncertainties as no immediate solution to make the NN evaluate several nodes at once could be brainstormed.
- Approach three has previously shown that it yields slightly worse results than simply using Dijkstra's algorithm.
- Approach two and three would require more inputs to be fed into the neural network. This would be a problem because neural networks tend to work better with fewer inputs. With more inputs, any new inputs added

would have a smaller effect on the output, causing key factors not to be as important.

With the chosen approach in mind, the following inputs were chosen as a starting point before gradually adding new key factors:

- Current node coordinates
- Coordinates to a neighbor in the graph
- The cost of traveling from the current node to the potential next node
- Coordinates to the end node

With the inputs to the NN chosen, implementation could move onto making the NN understand its purpose. This has to be done using a training method. In this case, reinforcement learning was chosen for a couple of reasons:

- Training a NN with a fitness score allows for many key factors to be taken into consideration. Evaluations can be made for specific key factors, and scores can be given based on how well the route adheres to the key factors.
- The possible alternative, supervised learning, has some problems. As specified with the third approach earlier in this subsection, Dijkstra's algorithm cannot train a NN to always find the best path and simultaneously limits the application of key factors.
- The NEAT architecture is created to utilize fitness scores to augment the NNs weights and topology.

Using reinforcement learning for the NN worked well, and the following factors were used to calculate its fitness score:

- Reaching the final node would reward the NN with a great amount of score - To make sure the NN prioritizes routes that reach the end.
- The entire cost of the route would be subtracted from the score - This makes the NN prioritize low-cost routes.
- If the route would randomly go back to the node it came from, the score would be subtracted - This makes sure that the algorithm does not loop between two roads with 0 cost.

This scoring system ensured that the NNs that reached the end, with the cheapest path possible and no backtracks, gained the most score. Thus the NNs were trained to find the most optimal paths.

At this point, the prototype showed signs of giving the intended result, but the data would become "Not a Number" (NAN) at later generations. While troubleshooting, a correlation between specific activation functions built into the neataptic library and the data that turned into NAN was made. By looking into the neataptic issues on GitHub, other people with the same problem were found, and the problem was fixed by looking into which activation functions worked and specifying which of the functions in the library are allowed to augment the NNs structure.

With no noticeable problems left, optimizing and scaling up the solution to more difficult route optimization problems remained. There was, however, only a couple of days left of the current implementation idea to stay on schedule. At this point, there were two potential choices, either moving on or continuing with the NEAT solution. Moving on was eventually decided on, as a couple of limitations were noticed in the NN architecture that was believed not to be fixable and would cause problems in more advanced cases such as TSP. Because of this, the remaining days were spent understanding the issues and documenting them.

The last important decision made during this phase was not to test all key factors in the implementations. The following list, which will be called "Key factor limitations", shows the reasons that led to this:

- Time was limited.
- The gain from using theoretical values as key factors seemed redundant as the ability to utilize them had already been verified.
- Without real data, it would be impossible to verify their application to the algorithms.

The decisions and experience gathered during the implementation of the NEAT solution will be leveraged into the upcoming two weeks of work on the GA with NN solution. Because of this, it is expected that the process will go smoother.

### 3.5.6 GA with NN

As described in subsection 3.5.1, this idea requires a NN to function, and therefore one has to be trained. However, a new NN did not have to be implemented since the idea could be tested with the NN generated by the NEAT algorithm described in subsection 3.5.5.

Before the implementation started, some other parts from the NEAT solution could be copied and modified to save time. The same node network could be used, as well as the HTML page displaying the nodes and eventual path taken by the algorithm.

According to the work structure specified in subsection 3.5.4, the implementation should start by solving a simple route planning problem. Therefore, it was decided that just like for NEAT, the initial goal would be to find the path between two nodes. To accomplish this, a function that utilizes the NN to assign probabilities to all edges in the node network was implemented.

However, when looking at the probabilities generated by the newly implemented function, it was discovered that the NN would assign inaccurate probabilities to the edges in the node network. The probabilities set by the NN did not show any signs of pointing towards an optimal path, but rather, an edge leading directly to the destination node often had a lower probability than an edge leading in the other direction. As a result of this, it was decided to entirely rule out the NN solution and instead focus on a solution using only a GA.

Ruling out NNs from the solution was not without drawbacks. By using a NN, the goal was that a more general path towards the destination would be found. Compared to using solely a GA solution, more combinations had to be tested, and thus more iterations had to be done. However, by using a GA solution, an optimal route would be able to be found, which it was concluded that the NN could not guarantee.

Because the initial idea of using a NN to evaluate edges was deemed unfavorable, a new method of selecting routes was needed. When looking at how this was done in other solutions, the standard concept was to do it randomly. This benefits the GA because it allows it to test many more combinations, gradually optimizing the paths and eventually finding the best path. Therefore, it was decided that all edges would have the same probability of being chosen while traversing the node network.

With a solution to the problem found, discussions between the thesis workers ensued regarding every aspect of the GA solution to figure out how every part could be implemented. Finally, deciding that the specific features of the algorithms would be implemented with the following approach.

The general structure of a GA described in section 2.2 was followed. First, a population of trucks was created, and later it was augmented according to the steps described below.

Starting at a start node, the trucks are to traverse the node network, saving their path along the way. At each node, a random edge departing from that node is chosen, with each edge having an equal probability of being picked. This process is repeated until the destination node has been reached.

The trucks' paths to the destination node are then evaluated, and the trucks are scored based upon how well they perform. The truck will gain score if it reaches the destination node and lose score exponentially based on the cost of the path taken. This scoring system will ensure that the trucks that reach the end, with the cheapest path possible, will gain the most score, and thus the most optimal path is found.

For the GA to use the score as efficiently as possible, it has to be normalized before using it in crossover. This is done by adding together all the scores to receive a sum and then divide each score by the sum. The normalization ensures that all scores consist of a value between 1 and 0 while still being proportional. This is important to ensure that scores do not differ significantly between each other since this would greatly favor a high score of being chosen, almost completely negating any truck with a lower score.

After the trucks have acquired a path and a normalized score, crossover is the next step. There was no immediate idea regarding how crossover within a route optimization GA could be implemented. However, generally, crossover is implemented by combining the genes of two different objects. In this case, the path taken by the trucks can be seen as the genes, and thus the implementation of crossover performed the following:

Steps for crossover:

1. Randomly pick two trucks (truckA and truckB) out of the population. A truck's likelihood of being picked is proportional to the score of that truck, whereas a high score relates to a high probability of being picked.
2. Extract all nodes that both trucks have visited in their path.
3. Choose one random node out of the nodes that the paths had in common. If the paths did not have any nodes in common, go back to step 1.
4. Create a new empty path.
5. Extract the nodes from truckA's path, from the starting node up until the common node, and add it to the new path
6. Extract the nodes from truckB's path, starting at the common node up until the destination node, and append it to the new path
7. The resulting path consists of a combination of the two truck paths.

While performing crossover, the split has to happen where a common node is present. This is because if a random spot in both paths were chosen, the odds are that the resulting path would not be a possible path most of the time. This means that the path would consist of a sequence of nodes that cannot be traversed because no edges exist between the nodes at the splitting point.

The problem with crossover is that it only takes into consideration the paths that already exist. This means that no new information would be added to the system after the initial population has generated its path. Hence, mutation was a critical aspect of the algorithm. Mutation made sure that new paths could be found and introduced more randomness to the system, which raised the likelihood of finding the optimal path. The implementation of mutation performed the following:

Steps for mutation:

1. Randomly pick trucks based upon a mutation rate. The mutation rate decides the percentage chance of a truck being chosen to be mutated.
2. Choose a random node somewhere in the truck's path traveled.
3. Starting at that node, randomly traverse the node network until the destination is found or the path reaches a specific length.

After implementing the crossover and mutation, the algorithm was finished and ready to be tested. The results were reviewed and more functionality was to be added. Therefore, the ability to solve routes where multiple points had to be visited was added. This required a few minor changes and additions to the algorithm:

- The ability to input an array of nodes that should be visited was added.
- Modified the scoring method, making score be gained for each node visited.
- Score was also gained if the last node visited was the same as the start node. This decision was made since the delivery truck most likely wants to return to the depot after being done delivering in a real-life scenario.

The algorithm showed promising results, and therefore the node network was enlarged to make it harder for the algorithm and make the network more similar to a real-world scenario. The node network was changed from 5x5 to 10x10, quadrupling the number of nodes in the node network.

The algorithm was still performing well; however, performing many generations took a long time. The code contained many necessary intertwined loops. Therefore, to optimize the code, the functions that were in the most inner loops were looked at first. It was discovered that some of the functions from the imported libraries were slower than anticipated.

The functions in question served to retrieve data stored in objects that were constructed in the libraries. The way to optimize this was to retrieve all the required information once when the algorithm was first run and store it locally in the client. This small change made the algorithm run on average 24 times faster.

As the algorithm now ran faster than before, it could be tested with larger populations and more generations.

The next step would have been to incorporate load planning and implement the solution for this. However, as the end of the prototype phase was coming, only enough time to devise some general ideas on the subject was possible. The consensus was that load planning could easily be incorporated by dividing which nodes to visit across several trucks. A couple more specific implementation ideas were made on this topic, but since it was not possible within the time frame of the thesis work, the prototype phase was deemed over, and work on the thesis would begin.

## 3.6 Considerations

Several choices were made to limit certain aspects of the thesis. These choices were done for different reasons, either because they would take too much time or because different problems would accompany their usage.

The first choice on this matter was not to use OSM. OSM was intended to be used to create node networks based upon actual roads in cities. This would provide the ability to understand the implementations in the prototype better and figure out their real-world application. This was decided against after looking into how to do this, as an extensive amount of time would have to be dedicated to understanding APIs, translating and retrieving this data, as well as there being no direct examples of how this could be done from what could be found.

Not using the data supplied by Amazon was the next of these choices. Amazon was going to provide real data of different routes that trucks have taken in the USA. However, when this data was released, it only consisted of point 1, point 2, and how long it took between them. This was deemed unusable for the thesis work, as it would limit the usage of key factors.

Trafikverkets API was also intended to be used but was decided against for the same reason as OSM. Much time would have to be dedicated to figuring out how to collect and apply their data.

No interviews were conducted during the thesis work, even though this was a step in the timeline. The plan was to contact different delivery companies and ask them how they plan routes today. However, getting a hold of this information was more challenging than expected. Most delivery companies that were looked into would advertise that they were using software to plan routes, but getting a hold of specific information regarding this software was impossible. The problem stems from two facts.

1. Companies have no reason to share how they optimize their routes
2. Only developers at the company, who cannot be contacted easily, understand how the software was created.

Because of these problems, it was decided that the question would be answered by gathering information online instead of conducting interviews as initial search results showed promise. Therefore this was instead done during the interview step of phase one.

## 3.7 Source Criticism

Many sources were used in this thesis, and to prove their validity, this chapter will motivate why they were chosen.

Sources [1], [2], [3], [7] ,[13], [20], [24], [30], [38],  [44], [45], and [46] were all chosen because they provide information from well known companies. These companies would receive serious backlash if they provided false information and are therefore likely to release correct information.

Similarly to the sources from the companies [40], [41], and [42] contain information from JavaScript and Node.js. JavaScript is a worldwide programming language, and Node.js is a popular runtime in the software development community.

The sources [26] and [27] are libraries used while implementing the prototype and were chosen because they are well documented and have seen many downloads and much use.

The books [4], [8], [18], [19], and [32] were chosen to be trustworthy as they have all gone through a process to be published. Generally, this process weeds out any misinformation. Furthermore, the messages of the books also aligned with what was learned during the thesis work.

The academic texts [5], [9], [11], [14], [17], [25], [33], and [39] are sources taken from academic journals. These can be trusted since all material published in these journals are thoroughly examined and approved by experts before being published.

Source [10] was chosen as it is from the Conference on Machine Learning and Cybernetics. To become approved for a conference like this, proof of information validity is required.

The thesis [43] has been published and thoroughly explains the process by which it got to its result, as well as providing information on how the result was achieved.

The blog posts [6], [16], and [21] were deemed as trustworthy and accurate after cross-referencing the information gathered with other sources to validate its authenticity. Furthermore, the blogs have many followers, and the articles have many likes and comments, which further adds to their credibility.

Book [12] is self-published. However, it is still deemed trustworthy after cross-referencing the information read and since the author has much knowledge within the field of which the book is written. This knowledge stems from the fact that he has published hundreds of videos of himself coding in which he references and explains his process of doing so.

The video [34] is published by the same person as [12]. The video shows an entire implementation process of neuro-evolution, all code and the result can be seen in the video, making it credible.

The online book [15] was used to create a youtube video series on Neural Networks by 3Blue1Brown. The video has been recommended by several professors and has seen its use as an introduction for deep learning in several courses. Based on the video's approval by many professors, the book’s eligibility can also be approved. The author also has a Ph.D. which means he has a reputation to uphold and much experience in research.

The blog in source [22] is run by a person who has a Ph.D. in artificial intelligence, and therefore, the information regarding AI and ML found in the blog is deemed credible.

Source [23] is an article about the NEAT algorithm and neuro-evolution written by the person who invented the algorithm.

The PDF linked in [29] is created and used by a coach who has been teaching agile work methods for 12 years.

The waterfall model, as described in [28], aligns with what the thesis workers previously learned about the model, making it credible for its use.

The sources [35], [36], and [37] are route planning software that had approval on the web.

The thesis plan in [31] is the plan followed while doing this thesis work and is therefore regarded as highly credible.

# 4. Results

This chapter contains the result received from following the methodology. The chapter touches on subjects such as how route optimization is done today, how key factors can be collected and used, as well as the result of the prototype.

The sections are categorized with the problem description in mind, apart from section 4.2, which applies to the thesis but does not directly correlate to a question in the problem description.

## 4.1 Route Optimization and the Supply Chain

Companies plan routes in very different ways. For example, if a company delivers packages to the same place for an extended period, they only have to plan the best route once and then use that route for all future deliveries. However, for companies that handle deliveries to new locations every day, route planning software becomes more beneficial as it can improve the efficiency of the deliveries. On the web, many different companies provide route planning solutions of this type [35][36][37].

For companies that handle point-to-point travel, Google Maps is a valid choice. This is because Google Maps finds routes with the use of Dijkstra's algorithm but also makes logical choices based upon real-time data [44].

Route optimization is not completely limited to optimizing which roads to travel. Optimizing parts of the supply chain can show effects on the logistics of route optimization and save fuel usage [3].

Google is actively working with UPS, the world's largest package distribution company, by helping them optimize their supply chain. This project is a part of Google Cloud and covers supply chain logistics. Google optimizes several parts of the supply chain using AI, thereby showing its potential in route optimization.

It is interesting to note that many areas within the supply chain can still be optimized as it consists of many different parts and many specifics that can be looked into. [45]

## 4.2 Route Planning Incorporation in Chosen ML Algorithms

While working on the prototype, in subsection 3.5.5 and 3.5.6, knowledge regarding how the NEAT and GA algorithms adapt to route planning problems was obtained. With this knowledge, information regarding what the algorithms can do, what they cannot do, and what they are suited for was documented. This information will be presented in this section even though it does not directly correlate to the questions in the problem description, as this information could be used to draw conclusions later in this thesis.

A visualization of node networks will be presented in subsection 4.2.1 and 4.2.2. To understand how the node network is visualized, the following knowledge is required:

Table 2. Description of the colors in the figures in section 4.2

| Color | Meaning |
|---|---|
| Red node | Start point, also end point in multi-stop graphs. |
| Purple node | Nodes that should be visited. If there is only one, then it is also the final node. |
| Blue edge | Edge traversed by the algorithm. |
| Green edge | Edge traversed by Dijkstra’s algorithm. |

If both algorithms traverse the same edge, then the edge will be colored blue in the figure.

### 4.2.1 NEAT

The NEAT algorithm initially showed that it could solve simple route planning problems when the node network described in subsection 3.5.5 contained 25 nodes.

Dijkstra path is green, cost: 33

NEAT path is blue, cost: 33

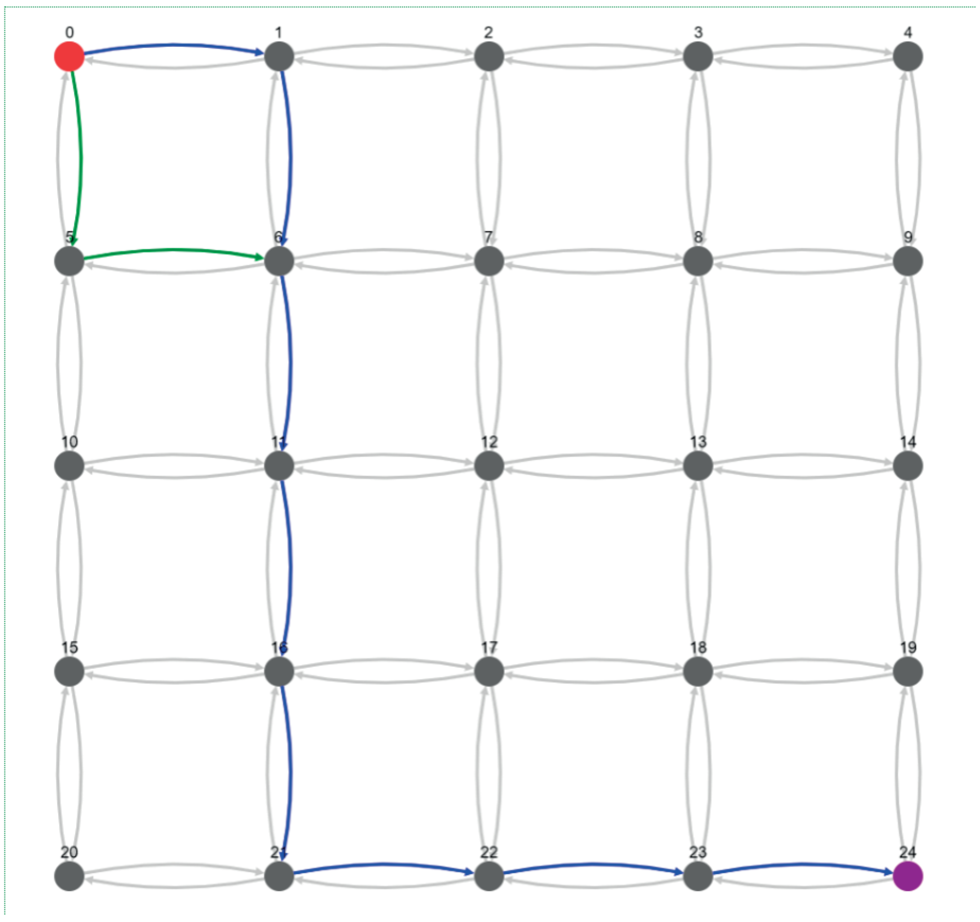
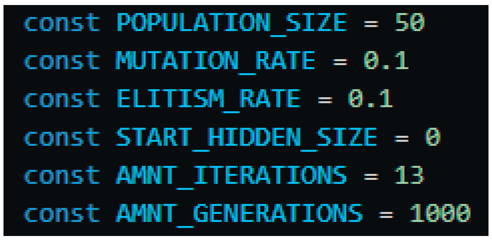


Fig. 8. Unbiased and average path in 5 x 5 network calculated by NEAT.

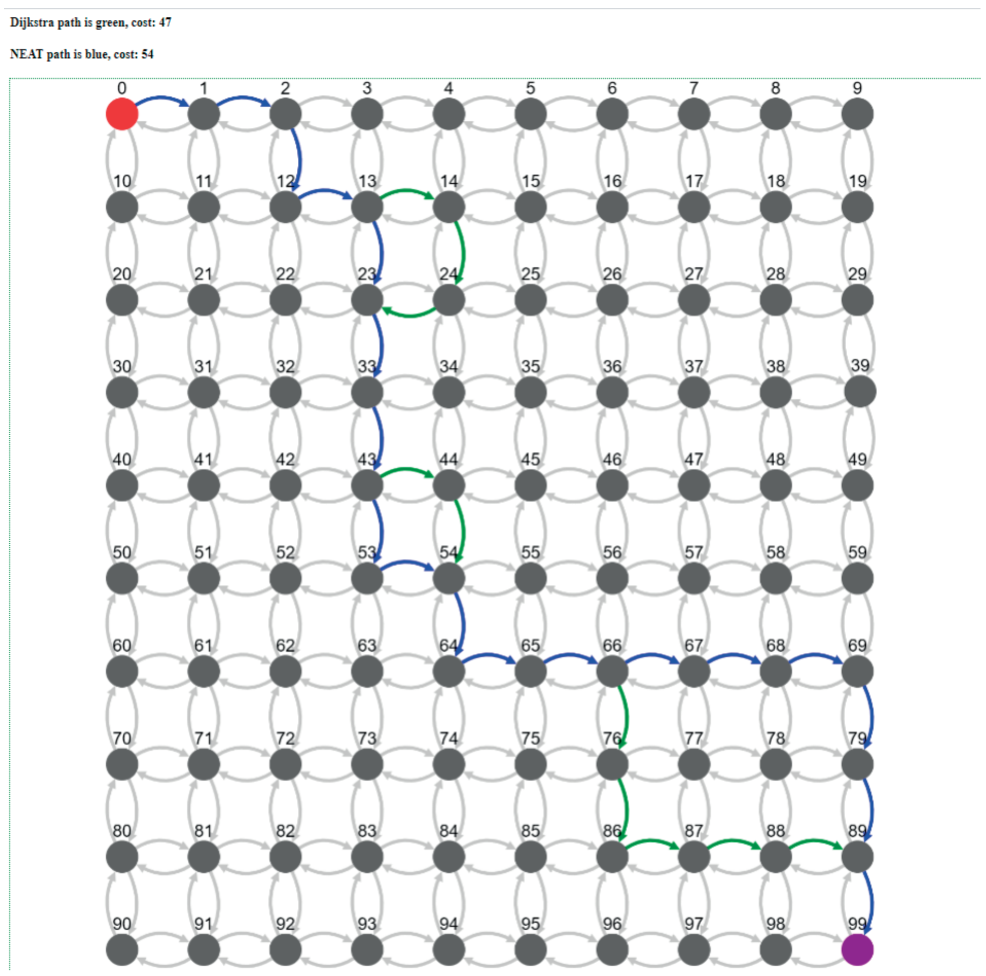
Fig. 8. shows a path generated by the NEAT algorithm. Fig. 8. is proclaimed to be unbiased and average as it was the first route the algorithm generated specifically for this use. When running the algorithm in a new randomly generated node network, with the same start and end node 10 times, the best path was found 5/10 times. Although, when the best path was not found, there were only slight differences in cost between the path generated by Dijkstra's algorithm and the path generated by NEAT.



```
const POPULATION_SIZE = 50
const MUTATION_RATE = 0.1
const ELITISM_RATE = 0.1
const START_HIDDEN_SIZE = 0
const AMNT_ITERATIONS = 13
const AMNT_GENERATIONS = 1000
```

Fig. 9. Settings for the generated path in Fig. 8.

If the settings in Fig. 9. would be changed by adding more generations, or if the initial population would be increased, the probability of finding the best path would increase at the expense of increasing how long the algorithm takes to run. However, training and running the network took roughly a minute using these settings, which seemed like an appropriate breakpoint, as waiting excessive time per path to potentially receive a higher probability of finding the best path seemed redundant. It is also worth noting that major differences in efficiency can be made by finding the best combination of settings. A large part of training a NN is making sure that the settings are tailored for the problem they are used to solve. For this specific problem, the settings in Fig. 9. are considered optimized.


Dijkstra path is green, cost: 47

NEAT path is blue, cost: 54

Fig. 10. Path in 10 x 10 network calculated by NEAT

When the node network is scaled up to containing 100 nodes, the probability of finding the best route diminishes completely. The route in Fig. 10 was the first path generated when using a 10x10 node network. Ten more routes were created with similar settings, as shown in Fig. 9. apart from generations and iterations being changed to 500 and 50, respectively. After 10 paths had been generated using the same node network with the same start and end node, 0/10 paths found the same path that Dijkstra's algorithm generated, as well as there being significant differences in total cost. However, the path still reached the end node every time.

Based on these tests, it was concluded that the implementation and utilization of NEAT would not be useful for planning routes. As this was explored, as specified at the end of subsection 3.5.5, two conclusions to the problem were made.

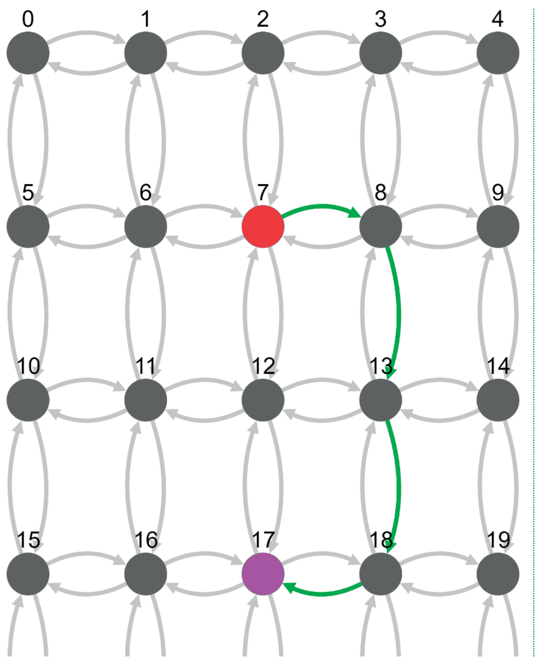
The first conclusion is that NEAT, and specifically NNs, make predictions based on their input data. When the number of predictions the NN has to make increases, the overall probability of making a wrong decision also increases. This leads to apparent differences in the overall cost of the path taken in the network.

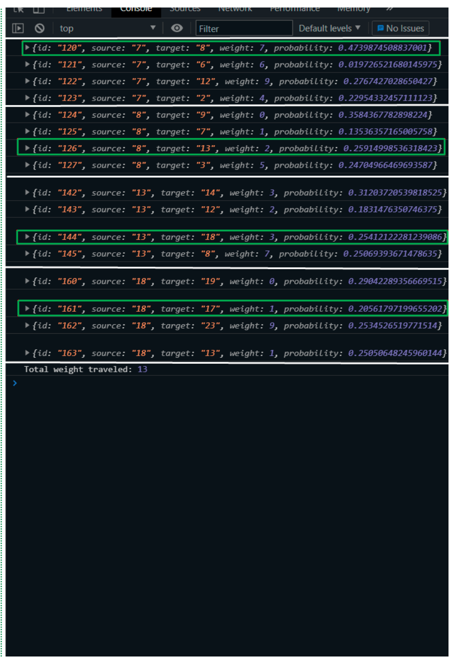
The second conclusion is that the NN prefers to make logical choices, making it not select potential routes that could present an overall lower cost. For instance, when moving towards the end node, the NN is reluctant to make choices that could provide a worse score. This can be seen in Fig. 10. as described below.

The path 13 - 14 - 24 - 23 adds three more nodes instead of going directly 13 - 23. Dijkstra finds the path containing more nodes to be optimal, as it always finds the cheapest path. However, the NN does not recognize the path 13 - 14 - 24 - 23 as a valid option. This is likely because the NN assumes that 14 - 24 and 24 - 23 should grant an average cost of 5 each, as every edge has a cost between 0 and 10. Instead of gaining a potential extra cost of 10, it should simply go directly to 23 from 13 and gain a cost that is on average lower than making the detour. However, it is possible that 14 -24 and 24 -23 both cost 0, making it a way more optimal path.

## 4.2.2 GA with NN

As described in subsection 3.5.6, the intended GA with NN solution had problems.





```
{id: "120", source: "7", target: "8", weight: 7, probability: 0.4739874508837001}
{id: "121", source: "7", target: "6", weight: 6, probability: 0.019726521680145975}
{id: "122", source: "7", target: "12", weight: 9, probability: 0.2767427028650427}
{id: "123", source: "7", target: "2", weight: 4, probability: 0.22954332457111123}
{id: "124", source: "8", target: "9", weight: 0, probability: 0.3584367782898224}
{id: "125", source: "8", target: "7", weight: 1, probability: 0.13536357165005758}
{id: "126", source: "8", target: "13", weight: 2, probability: 0.25914998536318423}
{id: "127", source: "8", target: "3", weight: 5, probability: 0.24704966469693587}

{id: "142", source: "13", target: "14", weight: 3, probability: 0.31203720539818525}
{id: "143", source: "13", target: "12", weight: 2, probability: 0.1831476350746375}

{id: "144", source: "13", target: "18", weight: 3, probability: 0.25412122281239086}
{id: "145", source: "13", target: "8", weight: 7, probability: 0.25069393671478635}

{id: "160", source: "18", target: "19", weight: 0, probability: 0.29042289356669515}

{id: "161", source: "18", target: "17", weight: 1, probability: 0.20561797199655202}
{id: "162", source: "18", target: "23", weight: 9, probability: 0.2534526519771514}

{id: "163", source: "18", target: "13", weight: 1, probability: 0.25050648245960144}
Total weight traveled: 13
>
```

Fig. 11. Node network with a path generated by Dijkstra's algorithm. The console on the right-hand side contains data regarding the edges in the node network. The "source" attribute corresponds to what node the edge starts in. The "target" attribute corresponds to what node the edge ends in. The "weight" attribute corresponds to the cost of traveling the edge.

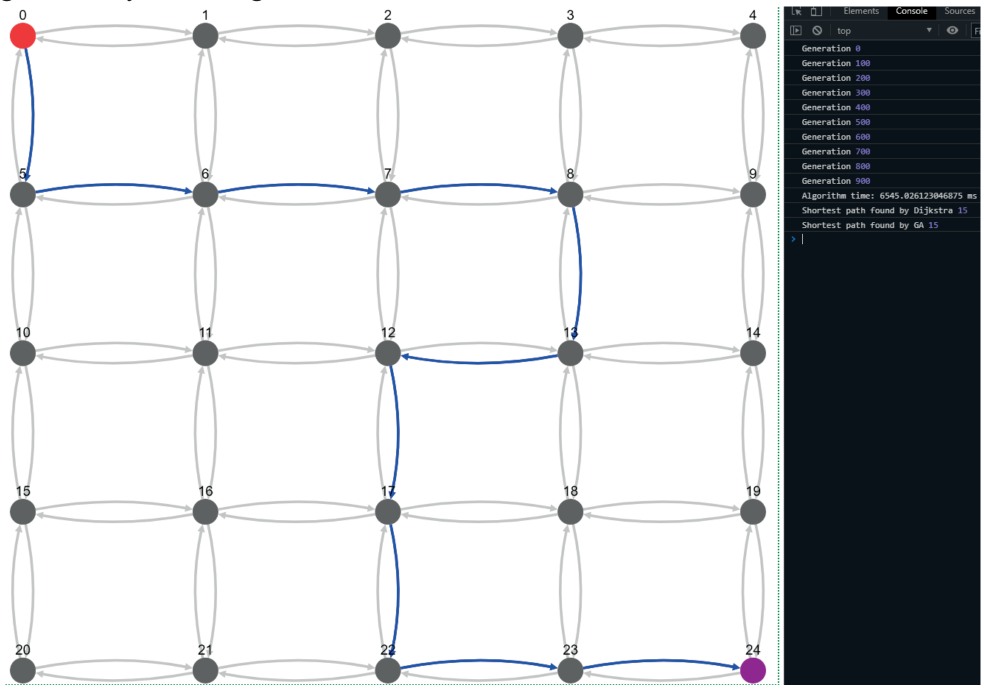
The cheapest path, as calculated with Dijkstra's algorithm, is marked with green edges in Fig. 11. In the console in Fig. 11., all the outgoing edges from a specific node are confined within white boxes. Within the white boxes, each green box corresponds to the green edges in the node network.

The solution did not work as expected, which will be explained using Fig. 11. Since the goal was to find the cheapest path, the probabilities generated by the NN should correspond to the green path. Meaning that the edges within the green boxes in the console should have the highest probability of all outgoing edges from a particular node. As can be seen, this is not the case. Rather, the best edge is only chosen one time out of four. Additionally, the edge connecting node 18 to node 17, the final node, has the lowest probability of all the outgoing edges from node 18.

Similar results to the ones seen in Fig. 11. was seen when using several other examples. Therefore, it was decided that giving all edges an equal probability of being picked was more advantageous since the calculated probabilities were wrong and would have led to worse routes being generated by the GA.

The main problem with the "GA with NN" solution is that in order to work, the NN used must be thoroughly trained. If the NN does not understand its purpose, and when used on its own, cannot solve routes, as seen in subsection 4.2.1, it cannot be used effectively in cooperation with a GA either.

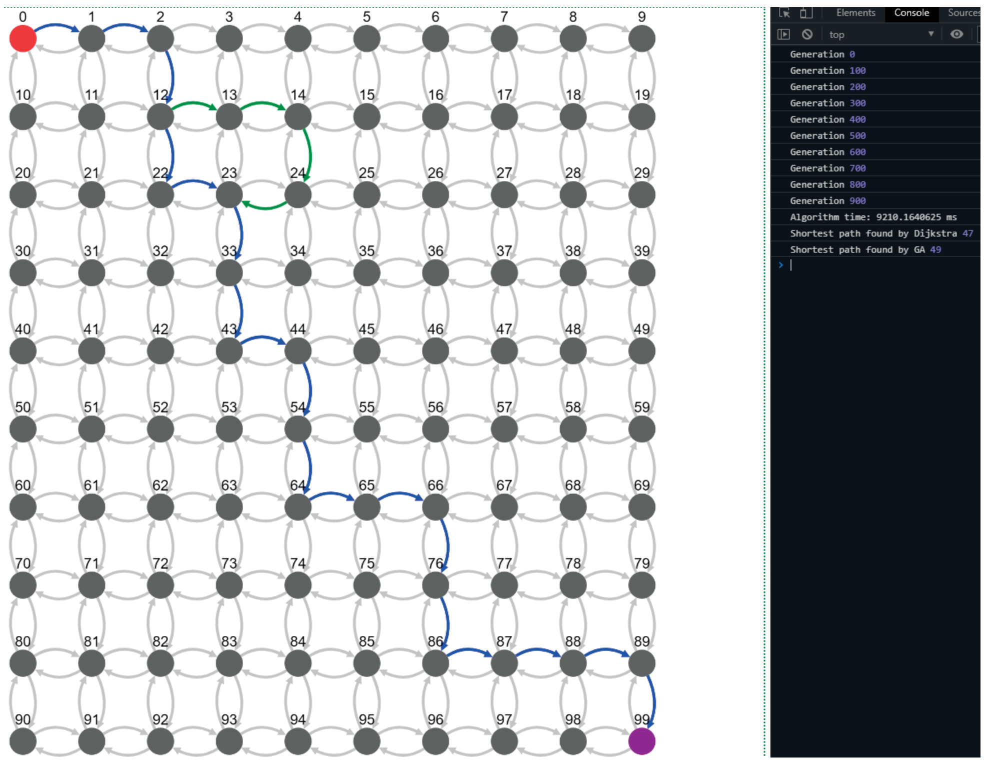
The GA solution that was implemented to utilize the equal probabilities showed great ability in solving routes.



```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Algorithm time: 6545.026123046875 ms
Shortest path found by Dijkstra 15
Shortest path found by GA 15
>
```

Fig. 12. Path generated by the GA in a 5x5 node network.

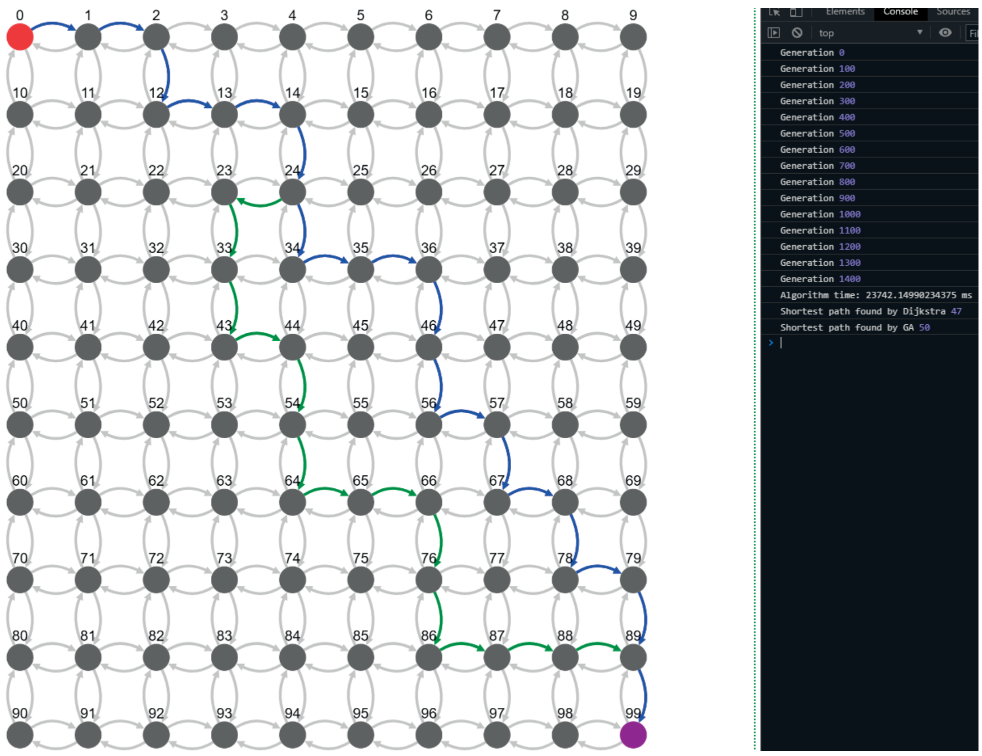
As shown in Fig. 12., the GA finds its way to the goal node and chooses the cheapest path there, as confirmed by comparing with paths generated by Dijkstra's algorithm. When running the algorithm in a new randomly generated node network, with the same start and end node 10 times, the best path was found 10/10 times. Thus, showing more promise than the NEAT algorithm in success rate while taking 4.5s compared to the NEAT solution, which took 60s, making it about 13 times faster.



```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Algorithm time: 9210.1640625 ms
Shortest path found by Dijkstra 47
Shortest path found by GA 49
```

Fig. 13. Path generated by the GA in a 10x10 node network

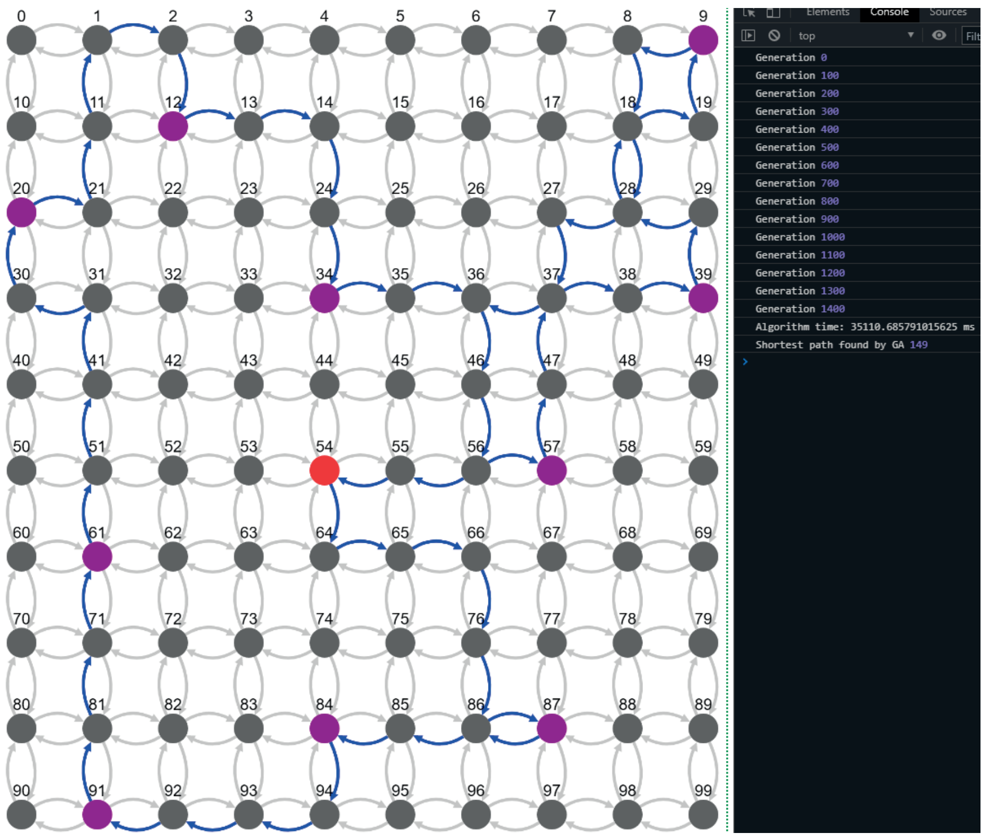
With the larger node network in place and using the same settings (amount of generations, iterations, and population size) that were used with the smaller node network, the algorithm found its way to the goal, but not with the best path. This is displayed in Fig. 13.



```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Generation 1000
Generation 1100
Generation 1200
Generation 1300
Generation 1400
Algorithm time: 23742.14990234375 ms
Shortest path found by Dijkstra 47
Shortest path found by GA 50
> |
```

Fig. 14. Path generated by the GA in the same 10x10 node network as in Fig. 13.

After increasing the number of generations, iterations, and population size, the path shown in Fig. 14. was generated. The algorithm was still not finding the best path every time. When running the algorithm in the same node network, with the same start and end node 10 times, the best path was found 2/10 times. However, a near-optimal path was found every time, with an average deviation from the cheapest path being a cost of 3, which corresponds to the GA finding paths that are on average 6% slower than the ones calculated using Dijkstra's algorithm.



```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Generation 1000
Generation 1100
Generation 1200
Generation 1300
Generation 1400
Algorithm time: 35110.685791015625 ms
Shortest path found by GA 149
>
```

Fig. 15. Multi-point path generated by the GA in the same 10x10 node network and with the same settings as earlier.

After adding the ability to find paths featuring multiple stops in the 10x10 node network, Fig. 15. was generated. As shown in Fig. 15., the algorithm visits all the nodes and returns to the starting point. The paths generated by the GA always find their way around all nodes and back to the red node again when testing the algorithm. This is a positive finding since a working path will always be generated for a truck to traverse when making deliveries.

Each time after running the algorithm in the same node network, with the same nodes to visit, a different resulting path is returned. This is due to the fact of how the GA is built upon randomness. With a large enough population size, amount of generations, and high enough mutation rate, the same best path will always be found. However, this is not realistic to test as it could take several days to generate one path. Therefore, it cannot be known whether these paths are the paths with the lowest possible cost or not.

Table 3. shows the result of running the algorithm in the same node network, with the same start and end node ten times. The best out of the paths generated is shown in Fig 13. More can be found in the appendix under section 8.2.

Table 3. A table showing the results from running the multipoint GA ten times.

| Iterations | Path cost |
|---|---|
| 1 | 171 |
| 2 | 160 |
| 3 | 185 |
| 4 | 185 |
| 5 | 190 |
| 6 | 177 |
| 7 | 154 |
| 8 | 185 |
| 9 | 149 |
| 10 | 187 |

The shortest path that was found had a cost of 149, and the average cost of the paths in Table 3. is 174.3, which is 14.4% more than the cheapest found. However, as mentioned above, this does not mean that 149 is the cheapest path possible for that specific node network.

It can be concluded that the final GA implementation performs well and finds decent paths.

Another conclusion can be made while comparing paths generated using the GA in the large node network. Because of the randomness that lies in the nature of the GA, the more possible paths that the GA can choose from, the more deviations there will be between the paths generated.

## 4.3 Key factor utilization

The application of key factors within machine learning is a wide area. It is possible to apply key factors in many different ways and the ML algorithms that were surveyed in this thesis work do not have any limitations on their potential usability. The key factor list in subsection 3.4.3 specifies some of the initial key factors that were going to be tested in the prototype. Out of those factors, not all were tested, as explained in subsection 3.5.5 in the key factor limitation list. Instead, only the following were tested in the prototype:

- Cost (A value used to provide the algorithms with a sense of how long an edge takes to travel)
- Start node
- Nodes to visit
- End node

Other than the factors mentioned in the key factor list, coordinates of the nodes were also tested to provide the NEAT algorithm implemented in subsection 3.5.5 with a general idea of direction.

By working on the prototype and assessing the key factors usability in route optimization, it was concluded that the only limitation on which key factors that can be used is if the key factors affect the time it takes to traverse a route. This means that the key factors mentioned in the key factor list definitely can be used in route optimization and, more specifically, for route planning. It would also be possible to utilize factors such as traffic, vehicle or driver information. Therefore, it is more interesting to think about the key factors in the sense of their application to route planning solutions using ML algorithms.

The key factors that were tested in the prototype worked well, and it is presumed that applying any of the other key factors in the key factor list in subsection 3.4.3 would be possible because of the following reason: The two algorithms that were implemented, described in subsections 3.5.5 and 3.5.6, utilize a fitness score. By using a fitness score, the problem of applying key factors to the ML model directly correlates to how the fitness function is implemented and which data is available in the function. Given enough data and analysis, it would be possible to apply fitness correctly and make the routes selected be fine-tuned to any key factors.

Key factors can be used in any machine learning algorithm that can be used to plan routes. However, the application is different for every algorithm. With the result retrieved in this thesis work, only ML algorithms that utilize fitness (GA and

NEAT) or algorithms that use a NN to make decisions can be confirmed as their ability has been tested. NNs ability was confirmed as they have many different learning methods, reinforcement learning being one of them, which makes it possible to apply fitness scores to them. Therefore, the problem is not necessarily “which ML algorithms can use key factors” but rather how the key factors can be utilized effectively.

In subsection 4.2.2, it was confirmed that the GA could find paths with multiple stops in a node network. To do this, the GA needed a node network to traverse and edge costs associated with the edges in the network. With these two factors the GA could find routes. Theoretically, the only difference that has to be made to make the GA find more efficient routes would be to revise the edge costs in the system to take key factors into consideration and correspondingly use more precise edge costs than previously.

Based on the information retrieved and the experience obtained from the prototype, the best ML algorithm for evaluating edge cost would also be the best ML algorithm to utilize key factors.

At this point, the main problem is that the cost associated with an edge is hard to calculate. Many probability-based key factors have to be applied to calculate the actual cost, and this would require a thorough study of each key factor and its application. However, NNs are useful for analyzing data and making predictions, given the proper training. Reinforcement learning would not work for this, as this would lead to equally complex solutions as previously mentioned, however, supervised learning might.

The solution to this problem comes from a conclusion drawn after working with the ML algorithms, learning about them, and thinking about their application to route optimization.

The conclusion is that a NN could likely be trained to assess how long a certain edge takes to travel. If the NN is trained using supervised learning, it would be able to connect certain aspects such as traffic density and time of day, along with other aspects that affect time spent traveling. If this is implemented, it could increase the efficiency of the routes generated by Dijkstra’s algorithm as the costs in the system would be more precise, compared to only using factors such as speed and distance. Although, a problem with this solution is that the key factors have to be retrieved in real-time using a method, such as control towers.

While working on the prototype, an issue within the problem description regarding key factors was found. During the writing of Chapter 1, it was expected that key factors would easily be tested in the prototype to confirm their application and potential efficiency. However, as implementation started, the idea of verifying key factors in the prototype was seen as impossible without using data from real-world scenarios.

This problem had already been noticed when writing the background as the third limitation specified that data used in the prototype would be theoretical. While it is possible to verify the ability to use key factors using theoretical values, it is impossible to make any direct conclusions on how well they perform. This means that the question in the problem description, “Which key factors have the most significant impact on transport efficiency and utilization of fleets?” cannot be answered.

## 4.4 Key factor collection

Some of the key factors discussed in the key factor list in subsection 3.4.3 will be easily available, such as what time of the day and which day the route is for, as this information can easily be gathered. The same goes for start, end, and nodes to visit since they have to be known to create a route. Policies regarding how often the trucker needs to have breaks might differ between different companies. Thus this information has to be input by the user.

It is worth noting that the time of day will vary depending on how far into the route that has been traveled. Although this can be calculated since each edge should have a factor regarding how long it takes to travel, and thus it would always be known at what time each edge is traversed.

As for the other key factors in the key factor list, namely road length, speed limit, road availability, road conditions, weather, temperature, and road angle, these can be acquired for swedish roads from Trafikverkets API. More can be read regarding Trafikverkets API and its documentation here [38].

Another solution that was found for this problem was using control towers. Control towers within the supply chain could be used to collect, store, and utilize key factors in real-time. Information regarding previous routes could be stored to later be utilized as training data for ML algorithms. More can be read about control towers and their features at IBM [46].

## 4.5 Load planning

As mentioned towards the end of subsection 3.5.6, a plan was devised for how load planning could be incorporated in the prototype.

If each truck has an attribute containing its max load capacity, then that attribute can be utilized while calculating the route to travel. Each node that needs to be visited will contain a new attribute corresponding to the amount of load that needs to be delivered at the node. If the truck's maximum load capacity is too small to supply all the nodes, additional trucks will be brought into the calculation.

With this logic added to the previously implemented GA, along with other functionality to support dividing a route across several trucks at once, the GA would be able to find optimal routes for several trucks delivering goods. Each truck would then be packed according to what goods the nodes it has to visit need.

Implementing this would ensure that each route receives an optimal number of trucks and that the trucks always depart with an optimal amount of load. The result of this would be cheaper deliveries and less emissions since potentially less trucks would be on the roads. Additionally, if the routes are planned in such a way that the heaviest load the truck carries is delivered first, it could lower emissions even further.

# 5. Conclusions

Route optimization can be applied to many fields, but route planning was the focus of this thesis work. The GA solution showed potential in finding multi-stop routes and optimizing the routes by using a fitness function. The most critical aspect to increase the overall route efficiency was deemed to be related to edge cost. By calculating edge cost in a more precise manner, further optimizations were deemed possible by utilizing relevant key factors.

NEAT and NNs showed promise in solving simple route planning problems. However, as the problems got more complex, it showed that NNs had some underlying issues making them unfit for route planning. With this in mind, another way to utilize NNs was devised. Although it has not been tested, the potential of using NNs to calculate edge cost using supervised learning is seen.

The GA solution showed that it could often find the best path in point-to-point travel in a node network. When the problem was changed to multi-point, correct routes were always found, but it was impossible to verify if the best route was found. The GA solution showed the most promise in this thesis work. It worked well for traversing node networks and performing route planning. Given enough time, GAs could definitely be useful for problems such as planning routes when there are multiple stops.

Key factors showed great potential when used within a node network to decide edge costs, but more factors have to be further tested and with real data to be confirmed as a valid method.

## 5.1 Answers to the Problem Description

In this thesis work, ML application to route optimization was studied. More specifically, three main parts were looked into.

- ML application in route planning.
- Key factors utilization in ML algorithms
- Incorporating load planning in ML route planning solutions.

Seven questions pertaining to these parts were specified in the problem description. Only “Which key factors have the most significant impact on transport efficiency and utilization of fleets?” was not answered following the reasoning at the end of section 4.3.

***How is route optimization done today?***

Route optimization is not completely limited to finding the most optimal route. Optimizing other parts of the supply chain can have effects on the routes planned. Such as improving the storage capacity of delivery trucks which could ultimately lead to less total routes having to be planned. How route optimization is done today can therefore be seen as optimizing the supply chain as a whole. However, route planning is still the primary concern of this question, as it has the most direct effect on route optimization.

Route planning is done in many different ways. For point-to-point delivery, Google Maps seems like the safest and overall most efficient alternative. However, data gathered from previous routes can still be leveraged to make potential optimizations in point-to-point travel. For the multi-stop problem, utilizing this data becomes more challenging as the number of possible routes that could be planned drastically increases with the number of nodes to visit. Therefore, software that can make efficient routes using algorithms and data analysis becomes more lucrative and seem to be a popular alternative.

***Which key factors can be used in route optimization?***

All key factors listed in the key factor list in subsection 3.4.3 are usable in route optimization. However, theoretically, every factor that has an impact on a planned route could be used. This includes factors such as traffic, vehicle, and worker information.

It is worth noting that the overall usability of the factors is directly correlated to their application, meaning that they would have to be individually tested and evaluated to see if they can be incorporated in route optimization.

***How can data relevant to these key factors be collected?***

Two different methods of gathering data relevant to key factors were discovered. First, a control tower can be used to collect, store, and use data in real-time while vehicles are in use. Alternatively, they can be retrieved for swedish roads by using the API supplied by Trafikverket. As mentioned earlier Trafikverket only has this available for Sweden, which means that if the key factors are needed in other countries, a different solution has to be found.

***Which ML algorithm can utilize these key factors?***

Any ML algorithm that can be used to plan a route is able to handle key factors as well. This stems from the fact that most, if not all, key factors can be applied directly onto the node network instead to generate edge costs. However, this does require implementation of how the edge cost should be calculated.

***Which ML algorithm can utilize the key factors the best?***
For route optimization, this thesis showed that both NEAT and GAs could be used in route planning. GAs showed promise in solving multi-stop route planning problems and, given enough optimization, could be utilized for problems of this type. NEAT showed that it could be used to plan routes but severely lacked efficiency compared to routes planned using Dijkstra's algorithm.

The most beneficial way to apply key factors was deemed to use them as a way to evaluate edge cost. This would improve the GA solution and Dijkstra's algorithm's ability to find the best routes. Theoretically, this could be done by training a NN using supervised learning. Then, given enough real data, the network could start making logical predictions taking into account any factors the network was trained with.

***How can load planning be incorporated to optimize routes further?***
Load planning can be incorporated by adding attributes to the nodes in the network corresponding to the load size to be delivered and to the trucks to show how much load they can carry. By thereafter planning routes while considering these attributes, each truck will be assigned a route with an optimal amount of goods to deliver.

## 5.2 Fulfillment of Purpose, Goal, and Motivation

With the result presented in chapter 4, deductions in the area of route optimization can be made. Two algorithms have been tested and had their abilities verified in route planning scenarios. GAs showed promising results in solving complex route planning problems with efficiency and great potential for improvement.

However, it was not possible to verify that the conclusions made in this thesis fulfill the initial purpose, which is that goods would arrive on time, at a lower cost and energy expenditure than today. Instead, the potential of these conclusions lay in their ability to be worked on in the future, given more testing and better implementation. With this, the purpose of the thesis, presented in section 1.3, is considered to be partly fulfilled.

Hopefully, Capgemini will be able to utilize the result and studies done in this thesis to optimize the supply chain, and with that, further, fulfill the motivation described in section 1.6 of this thesis.

## 5.3 Ethical Aspects

The social benefit from this thesis work is significant. Since the thesis supplies knowledge on how routes can be optimized both looking at environmental and efficiency aspects, the routes found could lead to less vehicles on the road overall. This, in turn, helps the environment by lowering emissions and benefits the population with faster deliveries. However, if the improved delivery service leads to more people shopping online, it can harm the local shops already struggling in smaller cities.

An ethical dilemma that posed during the thesis work was that since it took place during a global pandemic, it had to be decided that all work and meetings would take place from home. This included meetings with the supervisors both at Capgemini and the university.

## 5.4 Future Work

This section covers the parts that can be expanded upon, such as, optimizing the algorithms, verifying certain conclusions, and figuring out real-world applications of the result discovered in this thesis.

The NEAT solution did not present a satisfying result. However, given more thought into certain aspects of the solution, it might be possible to improve the overall ability of the algorithm implementation. The aspects and the reason why they could be improved will be explained in the following list:

Mutation rate - With a higher mutation rate, it is possible that the algorithm would have an easier time finding weights better suited to the NN. It was found that using a low mutation rate worked best for the current solution. However, if the mutation rate was increased and the method of applying mutation to the network was changed, there might be some potential for optimization.
Mutation elitism - Every generation, a couple of the best networks are added to the new generation to ensure that no potential is lost going from one generation to another. However, an underlying problem with the library neataptic is that it applies mutation after these networks have been added to the next population. This means that there is a probability that all the best networks can be mutated, and progress could be lost.
Fitness calculation - The fitness calculation for the route planning implementation did not take many factors into account. Given a better method of rating the networks, it is possible that a better network, more suited to the problem, could be generated.

Normalization - NNs tend to work better depending on how the data is normalized. Therefore, it could be worthwhile to look into other methods of normalization.

Instead of creating a NN using NEAT, the RNN structure might be better suited for finding routes in a node network as RNNs work well with problems of sequential nature. When working with a RNN, it would also be possible to utilize TensorFlow, an optimized and up-to-date library for machine learning that companies like Google utilize for their ML needs.

Given that the GA solution could solve TSP and VRP in node networks, certain aspects would have to be optimized to make it more applicable to actual real-life route planning scenarios.

The overall efficiency of the implementation of the GA could be surveyed. The most costly operations were optimized, but certain algorithms in the implementation could probably see a reduction in run time, given a more thorough investigation.

As specified in the methodology, crossover and mutation were challenging to implement. If the two functions were given more focus, it could be possible to find better ways to implement them, which could show an increased efficiency in routes found. A potential improvement that could be tested is making crossover only select common nodes within the middle nodes of the truck's paths. This could lead to better paths since crossover generally is done by combining equal parts from both parents.

The idea of applying probabilities to how the edges are selected when the GA algorithm operates is still valid, even though there were complications to how the neural network was utilized for this problem. If the probabilities in the node network could be tailored to decrease the probability of picking roads that lead in wrong directions, breakthroughs in efficiency and the overall routes found could be made.

The problem of calculating edge cost would require much research. Time would have to be dedicated to implementing a fitness function in one of two ways. Either by analyzing data, making predictions, and testing the function. Alternatively, by retrieving data, applying it to a neural network, and testing its ability to predict edge cost.

An idea regarding load planning was devised in section 4.5, but was never implemented. Therefore, trying to implement and test this idea could be a valid step in order to test and verify its usefulness.

# 6. Terminology

Description of technical terms:
**AI** - Artificial intelligence, the simulation of human intelligence in machines.
**Control tower** - A central hub for data and metrics regarding the supply chain. Provides end-to-end visibility across the full supply chain. Utilizes technologies such as ML to function.
**Fleets** - Groups of delivery trucks all owned by the same company.
**Heuristic search -** A search strategy which finds approximate solutions, which are iteratively improved until a "good enough" solution is found. This is often used when classic search algorithms cannot find a solution.
**Key factors** - The machine learning algorithm will have to look at different values in order to be able to make decisions. We call these values "key factors" and these can be something like a speed limit for a road, how often the road is closed down for various reasons, how much of a slope the road has, how long the road is, how often the trucker needs breaks, where the trucker can take breaks, e.t.c.
**Load planning** - Describes the planning going into loading delivery trucks. Specifically optimizing what cargo goes in which truck. By doing this fewer trucks can be used and more optimal routes created.
**ML** - Machine learning, a subset of AI. With the specialization of analyzing data and creating analytical models from said data.
**Route optimization** - The route will be optimized in a way that reduces transportation costs, CO2 emissions and time spent delivering goods. It should also reduce the time real people have to spend on planning routes.
**Topology -** In the regards of neural networks, the topology describes how the network is structured, which includes the neural network's size, amount of hidden layers, and connections between the nodes.
**Transport efficiency** - The efficiency at which an independent truck or a fleet of trucks operate at. The factors we incorporate into the efficiency are transportation cost, CO2 emissions, and time spent on the road.

# 7. References

[1] https://routingchallenge.mit.edu/about-the-challenge/ Accessed 2021-03-04

[2] https://cloud.google.com/solutions/supply-chain-logistics?hl=bg
Accessed 2021-03-05.

[3] https://www.youtube.com/watch?v=QCm6Vx-eogc&ab_channel=GoogleCloud
Accessed 2021-03-05

[4] S. Even, "*Graph Algorithms*,"United States of America, Computer Science Press, ISBN: 978-0-521-51718-8, 2012.

[5] D. R. Lanning, G. K. Harrell, J. Wang. "Dijkstra's algorithm and Google maps," in the *proceedings of the 2014 ACM Southeast Regional Conference (ACM SE '14)*, ISBN: 978-1-4503-2923-1, Article 30, pp. 1–3, New York, NY, USA, march 2014.

[6]
https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 Accessed 2021-03-04

[7] https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html
Accessed 2021-03-04

[8] M. Melanie, "*An Introduction to Genetic Algorithms*", Cambridge, MA: MIT Press, ISBN 9780585030944, 1996.

[9] H.-S Hwang, "An improved model for vehicle routing problem with time constraint based on genetic algorithm,", Computers & Industrial Engineering, Volume 42, Issues 2–4, pp 361-369, ISSN 0360-8352, 2002.

[10] L-Y. Wang, J. Zhang, H. Li , "An Improved Genetic Algorithm for TSP", in the International Conference on Machine Learning and Cybernetics, Machine Learning and Cybernetics, ISBN: 978-1-4244-0972-3, pp. 925–928, Hong Kong, China, August 2007.

[11] M. A. Mohammed, M. K. A. Ghani, R. I. Hamed, S. A. Mostafa, M. S. Ahmad, D. A. Ibrahim, "Solving vehicle routing problem by using improved

genetic algorithm for optimal solution", *Journal of Computational Science*, Volume 21, Pages 255-262, ISSN 1877-7503, 2017.

[12] D. Shiffman, *"The Nature of Code",* **self published**, ISBN 978-0985930806, 2012.

[13] https://www.ibm.com/cloud/learn/neural-networks Accessed 2021-03-11

[14] M. Olazaran, "A Sociological Study of the Official History of the Perceptrons Controversy", *Social Studies of Science*, vol 26, no 3, pp. 611–659. ISSN 0306-3127, Aug. 1996

[15] "*Neural Networks and Deep Learning*"
http://neuralnetworksanddeeplearning.com/chap1.html Accessed 2021-03-11

[16]
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 Accessed 2021-04-01

[17] P. Ramachandran, B. Zoph, and Q. V. Le. "Searching for activation functions". arXiv preprint arXiv:1710.05941, 2018.

[18] L.R. Medsker and L.C. Jain "*Recurrent neural networks. Design and Applications"*, Washingtron, D.C., CRC Press, ISBN: 0849371813, 2001

[19] T. O. Ayodele, "*Types of Machine Learning Algorithms, New Advances in Machine Learning*", Yagang Zhang (Ed.) InTech, ISBN: 978-953-307-034-6, , 2010.

[20] https://www.ibm.com/cloud/learn/neural-networks#toc-neural-net-u3voPJVU Accessed 2021-03-08

[21]
https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063 Accessed 2021-03-08

[22]
https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/ Accessed 2021-03-09

[23]
https://www.oreilly.com/radar/neuroevolution-a-different-kind-of-deep-learning/
Accessed 2021-03-09

[24] https://eng.uber.com/deep-neuroevolution/ Accessed 2021-03-09

[25] K. O. Stanley, R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies", *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, ISSN: 10636560, summer 2002.

[26] https://js.cytoscape.org/
Accessed 2021-06-06

[27] https://wagenaartje.github.io/neataptic/docs/
Accessed 2021-06-06

[28] https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm
Accessed 2021-05-03

[29]
https://blog.crisp.se/wp-content/uploads/2013/08/20130820-What-is-Agile.pdf
Accessed 2021-05-11

[30] https://discord.com/
Accessed 2021-05-03

[31] Thesis Timeline
Accessed 2021-06-06

[32] G. Bonaccorso, "*Machine Learning Algorithms*", Birmingham, Packt Publishing Ltd., ISBN: 978-1-78588-962-2, 2017

[33] M. Dorigo, M. Birattari and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28-39, ISSN 1556-6048, Nov. 2006

[34]
htt4s://www.youtube.com/watch?v=c6y21FkaUqw&list=PLRqwX-V7Uu6Yd3975YwxrR0x40XGJ_KGO&index=3 accessed 2021-05-02

[35] https://optimoroute.com/ accessed 2021-05-06

[36] https://gsmtasks.com/ accessed 2021-05-06

[37] https://onfleet.com/ accessed 2021-05-06

[38] https://api.trafikinfo.trafikverket.se/API/Model accessed 2021-04-16

[39] M. Tabassum, K. Mathew. “Genetic Algorithm Analysis towards Optimization solutions” *International Journal of Digital Information and Wireless Communications,* vol. 4, pp. 124-142, Jan. 2014.

[40] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript accessed 2021-04-29

[41] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference accessed 2021-04-29

[42] https://nodejs.org/en/about/ accessed 2021-04-30

[43] https://kth.diva-portal.org/smash/get/diva2:1249111/FULLTEXT01.pdf accessed 2021-05-12

[44] https://www.google.com/intl/eng/maps/about/#!/accessed 2021-05-07

[45] https://www.ibm.com/topics/supply-chain-optimization accessed 2021-05-07

[46] https://www.ibm.com/topics/control-towers accessed 2021-05-06

# 8. Appendix

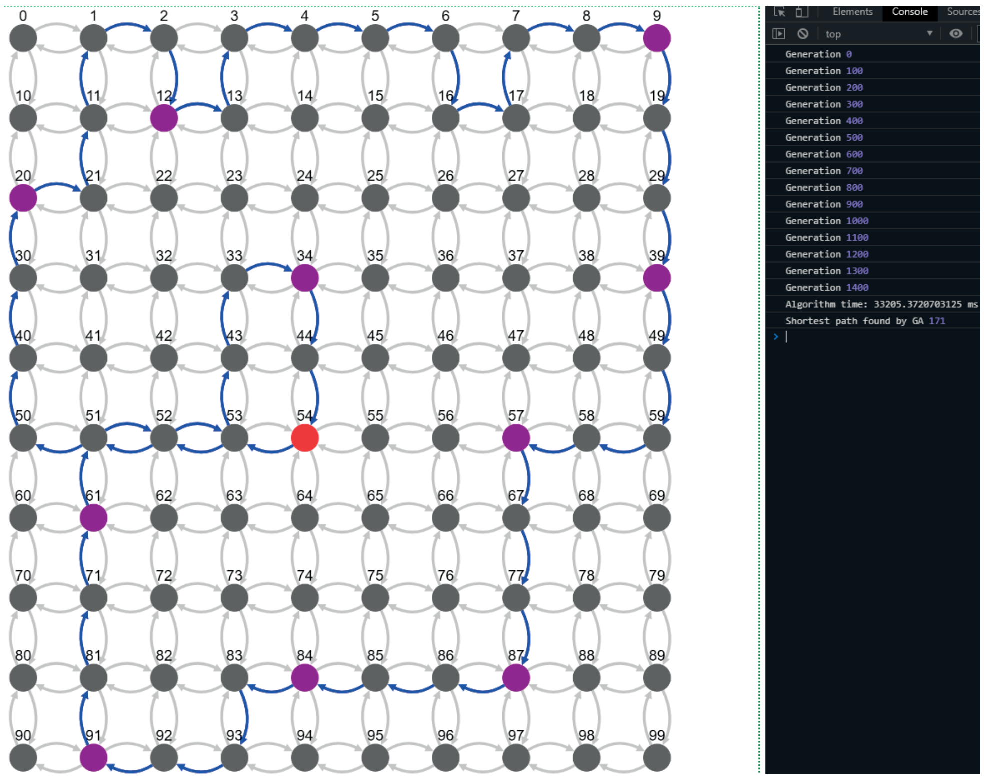
Contains lists and pictures that would take up to much space in the other chapters in the thesis.

## 8.1 List of Acronyms

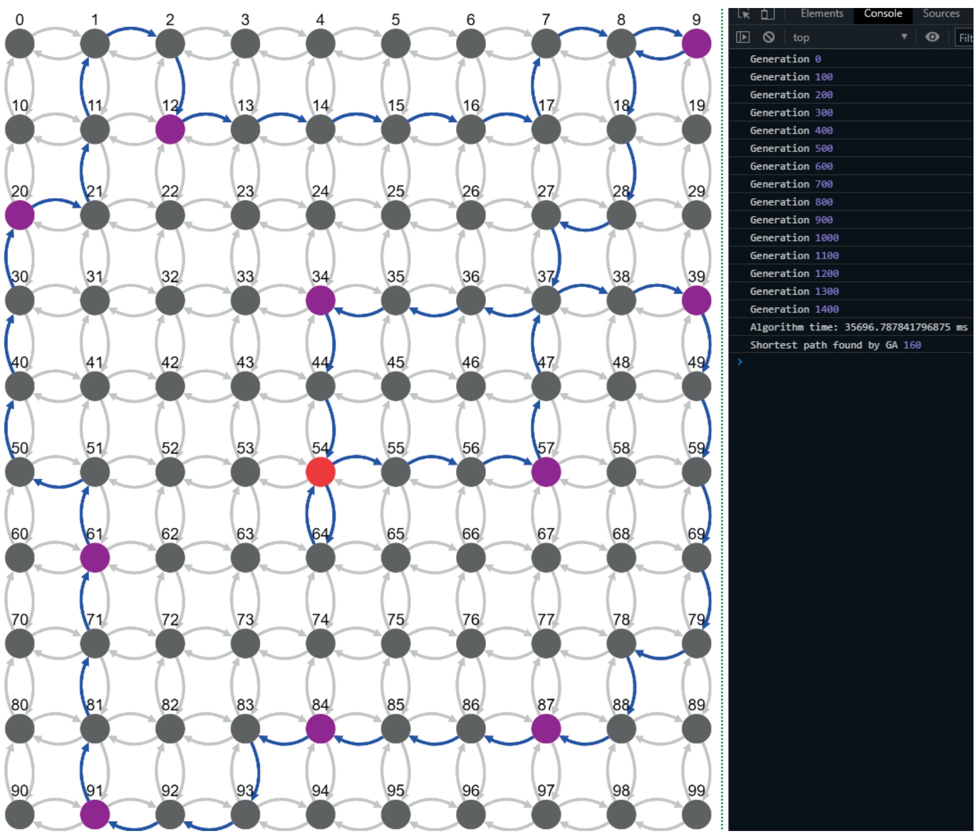
AI Artificial Intelligence
API Application Programming Interface
DL Deep Learning
DNN Deep Neural Network
GA Genetic Algorithms
GPS Global Positioning System
HTML HyperText Markup Language
IBM International Business Machines Corporation
IDE Integrated Development Environment
LTH Lunds Tekniska Högskola
MIT Massachusetts Institute of Technology
ML Machine Learning
NAN Not A Number
NEAT Neural Evolution of Augmenting Topologies
NN Neural Networks
OSM Open Street Map
RNN Recurrent Neural Networks
TSP Travelling Salesman Problem
UPS United Parcel Service
VRP Vehicle Routing Problem
VSC Visual Studio Code

## 8.2 Figures relating to Table. 3.

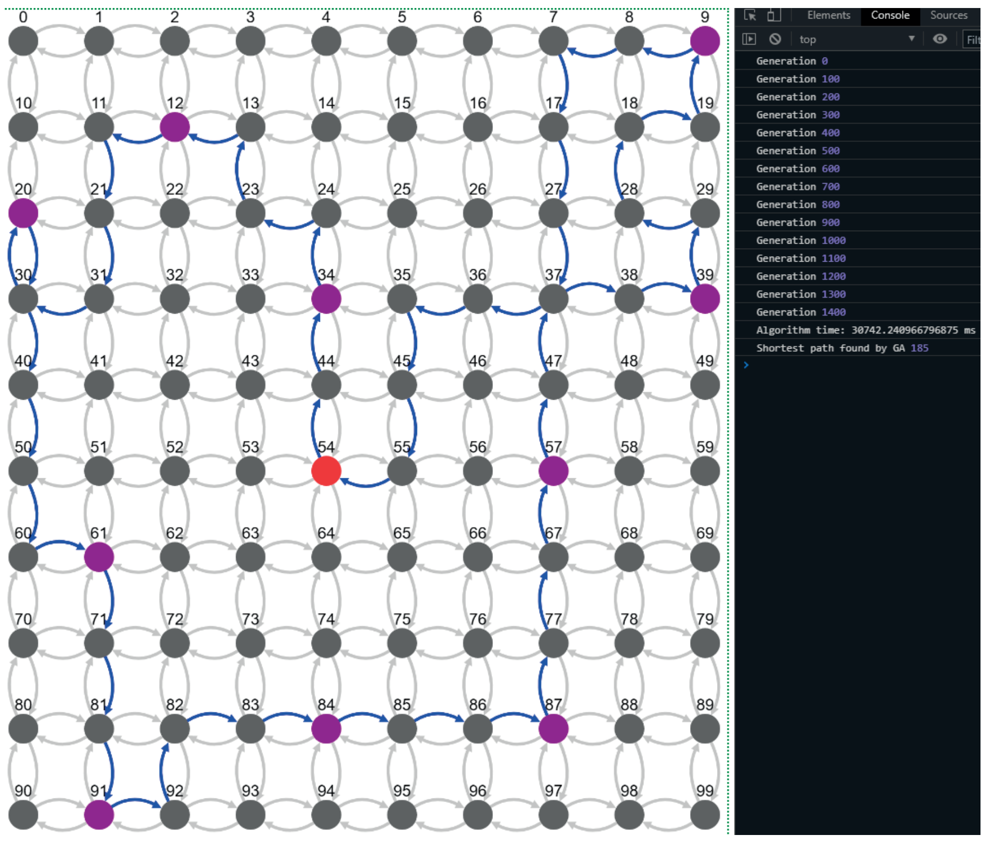
This section contains more multipoint routes created by the GA in subsection 4.2.2. The figures relate to entries in Table. 3. with corresponding shortest paths found.



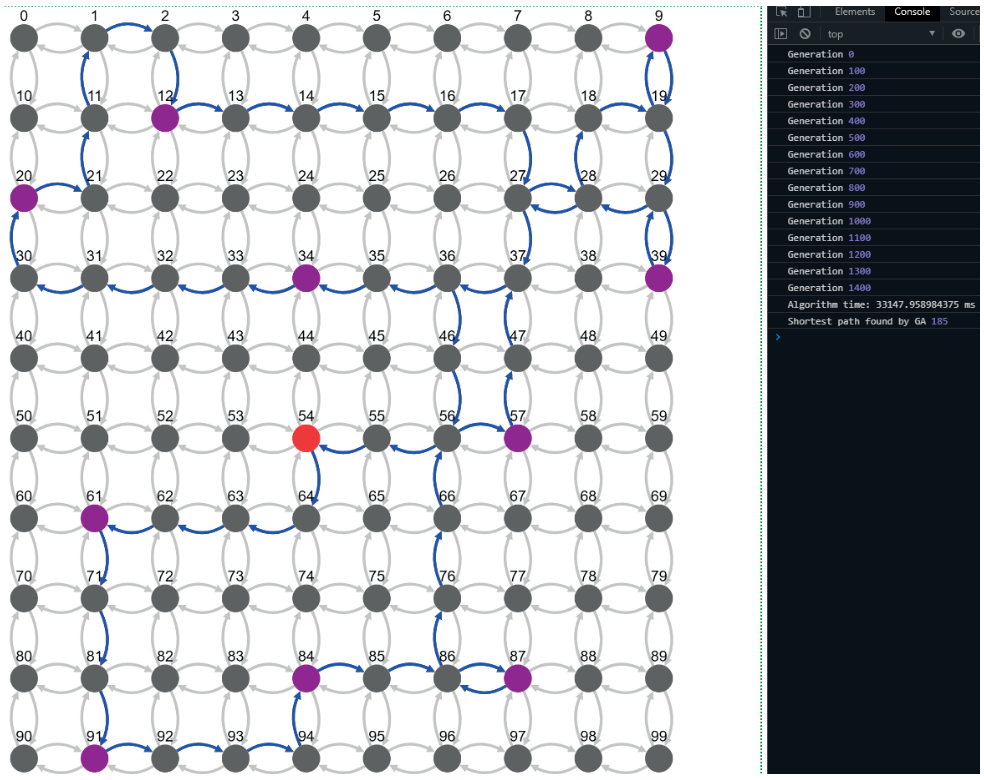
```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Generation 1000
Generation 1100
Generation 1200
Generation 1300
Generation 1400
Algorithm time: 33205.3720703125 ms
Shortest path found by GA 171
> |
```



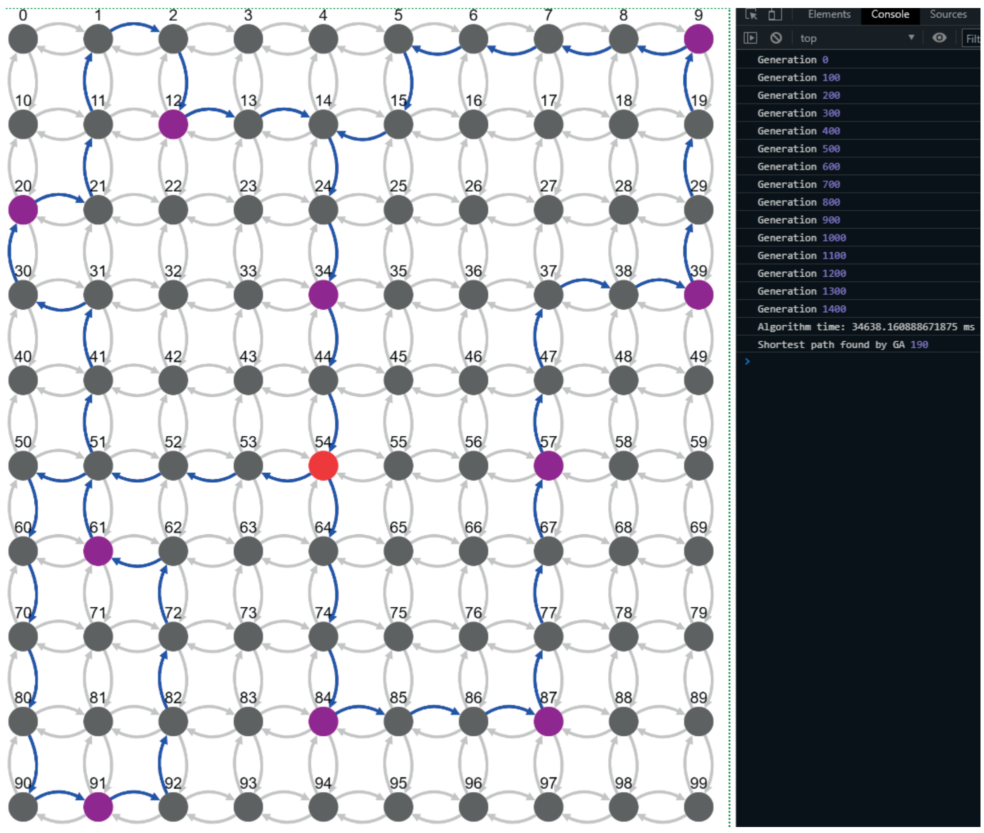
```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Generation 1000
Generation 1100
Generation 1200
Generation 1300
Generation 1400
Algorithm time: 35696.787841796875 ms
Shortest path found by GA 160
>
```



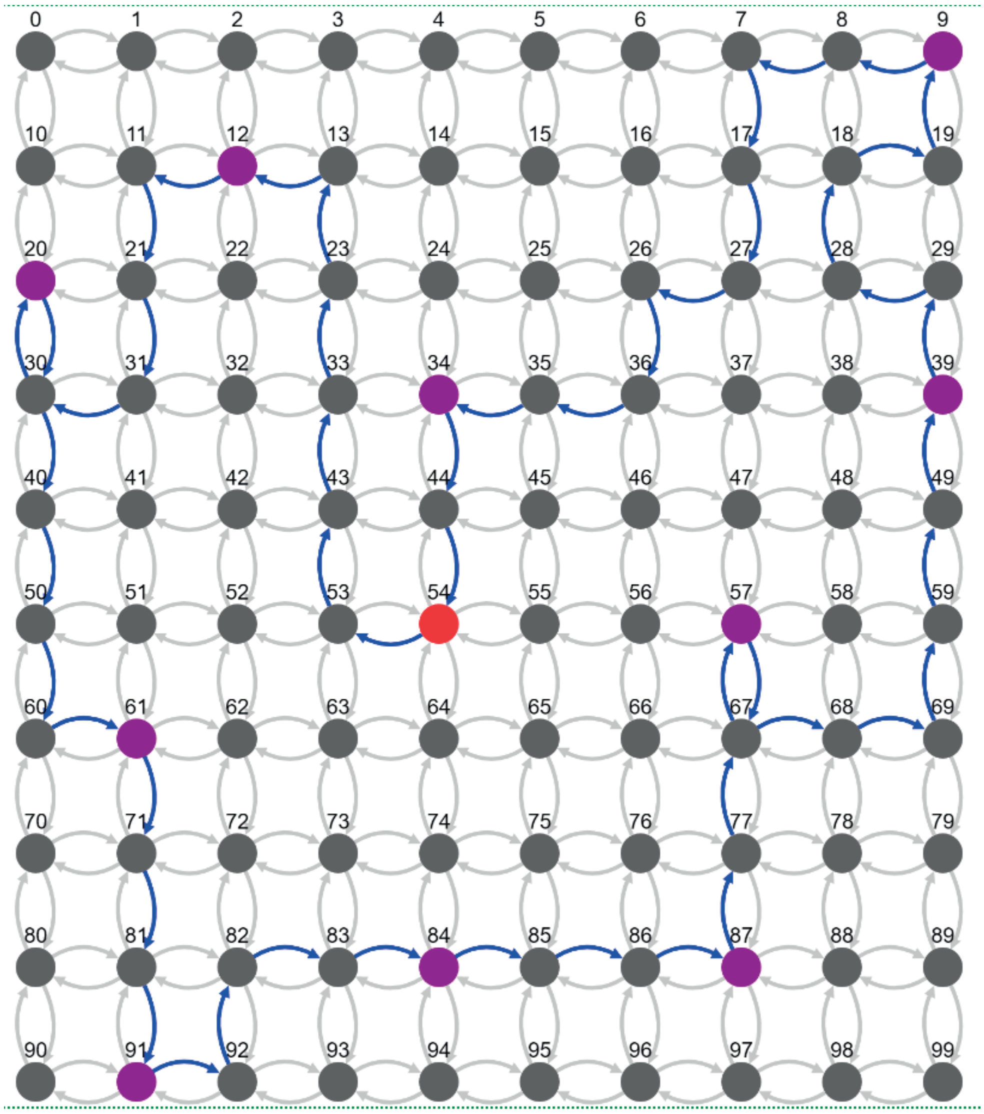
```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Generation 1000
Generation 1100
Generation 1200
Generation 1300
Generation 1400
Algorithm time: 30742.240966796875 ms
Shortest path found by GA 185
```

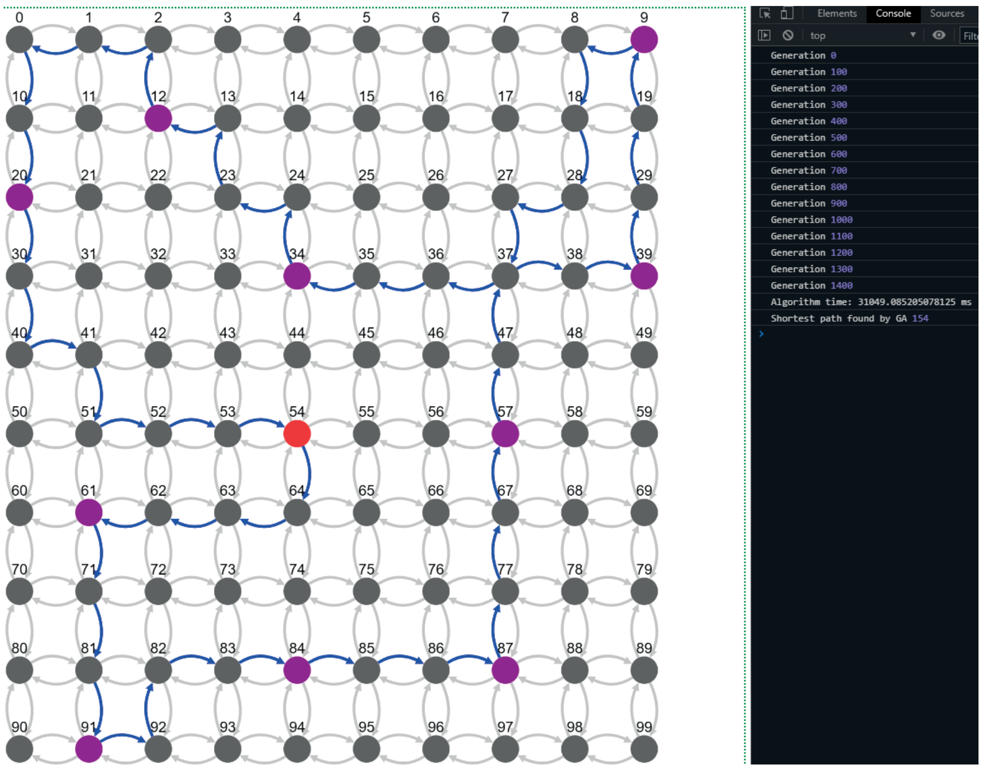


```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Generation 1000
Generation 1100
Generation 1200
Generation 1300
Generation 1400
Algorithm time: 33147.958984375 ms
Shortest path found by GA 185
>
```



```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Generation 1000
Generation 1100
Generation 1200
Generation 1300
Generation 1400
Algorithm time: 34638.160888671875 ms
Shortest path found by GA 190
>
```


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99



```
Generation 0
Generation 100
Generation 200
Generation 300
Generation 400
Generation 500
Generation 600
Generation 700
Generation 800
Generation 900
Generation 1000
Generation 1100
Generation 1200
Generation 1300
Generation 1400
Algorithm time: 31049.085205078125 ms
Shortest path found by GA 154
>
```

Printed by Tryckeriet i E-huset, Lund 2021

Series of Bachelor’s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2021-818
http://www.eit.lth.se