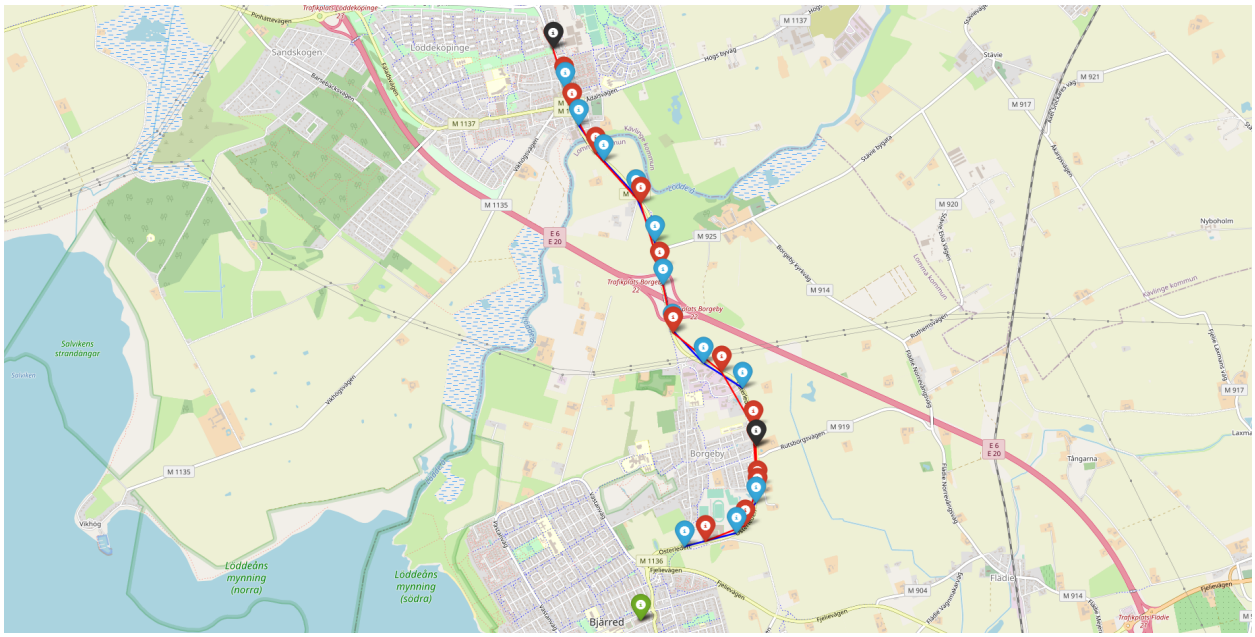
CHALMERS
UNIVERSITY OF TECHNOLOGY



Optimizing Road Network Weight Calculation for Emergency Vehicles

Predicting Emergency Vehicles Routes and Locations to Send Preemptive Warnings to Road Users

Master's thesis in Physics

VIKTOR WIKLUND
ANTON ROSENBERG

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Optimizing Road Network Weight Calculation for Emergency Vehicles

Predicting Emergency Vehicles Routes and Locations to Send Preemptive Warnings to Road Users

VIKTOR WIKLUND
ANTON ROSENBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

Optimizing road network weight calculation for emergency vehicles
Predicting emergency vehicles routes and locations to send preemptive warnings to road users
ANTON ROSENBERG
VIKTOR WIKLUND

Supervisor: Jakob Hendén, Carmenta Automotive
Examiner: Giovanni Volpe, Physics

Master's Thesis 2023
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telefon +46 31 772 1000

Cover: Example of emergency vehicle distress call, the actual vehicles position updates are in red and the predicted positions from the position estimator described in chapter 2 are blue. Furthermore the green and black point represent the target and start position respectively.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Optimizing road network weight calculation for emergency vehicles
Predicting emergency vehicles routes and locations to send preemptive warnings to road users
ANTON ROSENBERG
VIKTOR WIKLUND
Department of Physics
Chalmers University of Technology

Abstract

Correctly calculating routes and the future positions of emergency vehicles is important for traffic safety and can reduce response time during distress calls. Carmenta Automotive is a company that supports the EU-funded NordicWay3 project, which aims to create a safer traffic environment. Their objective in the project is to anticipate the path and location of emergency vehicles with the goal of sending warning messages to road users.

This thesis is a continuation of a previous thesis in which an algorithm for predicting the routes and positions of emergency vehicles was implemented. However, no data or metric was used to validate the developed algorithm's performance. In this thesis the goal was to develop validation metrics for this algorithm and try to optimize the cost of selecting each road, called its weight, in the road network. To achieve this an SQL database was created containing the mission data of emergency vehicles, which was used for validation purposes. Two validation metrics were also developed, the first uses the mean distance between the actual and predicted positions in a shorter time frame, the second focusing on more extended missions and selecting correct routes. Furthermore, parameters were optimized for three different weight equations, which are used to calculate the weight of each road using information about its speed, size and distance. The parameters were optimized using Competitive Particle Swarm Optimization with respect to one validation metric at a time. Through this process a maximum improvement of 4% and 7% compared to the previous thesis weight equation with respect to the first and second validation metrics was found respectively. Finally, the conclusion reached was that the goal of the thesis was accomplished and in order to further improve the algorithm's performance, a weight calculation considering more aspects of the road than just its size, length, and speed is required.

Keywords: algorithm, PSO, CPSO, A-star, stochastic, optimization, route, prediction.

Acknowledgements

We would like to thank our supervisor Jakob Hendén at Carmenta Automotive for all his help and insightful feedback during the development and thesis writing. We would also like to thank our examiner Giovanni Volpe for useful feedback and for making the thesis examination process easy and without trouble. We also thank our opponents, Jesper Jäghagen and Carl Hjalmarsson, for their thorough and thoughtful feedback. Finally, we would like to extend our thanks to the entire team at Carmenta Automotive for helping us with questions regarding code and the software environment.

The authors, Gothenburg, 05 2023

Contents

List of Figures

List of Tables

1

Introduction

Calculating the optimal route when traversing from point A to B is a common problem regarding transportation which has become increasingly more complex with more traffic and routes available. This problem has a tremendous economic impact on a societal level for several reasons. Efficient routes lead to less fuel consumption and faster delivery times when transporting goods [2]. Optimized routes are also important for public transportation and ensuring Emergency Vehicles (EVs) are able to arrive on time during distress calls which could save lives.

1.1 Algorithmic Approach

A computer science approach to deal with this problem is by developing algorithms well suited for finding optimal paths, such as Dijkstra's and A-star algorithm [3] [2]. These algorithms work by calculating the path with the lowest cost in weighted graphs. In road-based vehicles, getting from A to B can easily be represented as traversing a weighted graph [4]. This is done by letting the start and end of each road be represented as a node, while edges represent the roads connecting the nodes. Using this, the known algorithms can find the shortest paths given the cost of each road. However, calculating this cost is more complex since simply using the distance of each road as its weight ignores many factors, such as the traversability and speed limit of the road. Furthermore, the cost also depends on what you want to optimize; for example, if the most fuel-efficient route from A to B is desired, the weights will differ from those that yield the fastest route. Several models are developed for calculating how costly taking a particular route is [5]. These are, however, adapted for passenger cars. Moreover, it's exceedingly difficult to predict how the weight calculation algorithm affects the outcome, this makes any optimization method that uses properties about the objective function unusable. For example, whether the function is differentiable is unknown, so using the derivative to optimize may not yield correct results. This paper focuses on optimizing the calculation of weights for a directed graph traversal algorithm applied to EVs during distress calls. Since EVs don't follow the same restrictions, such as speed limits or illegal U-turns, as passenger cars, the road network needs to consider these possibilities. Furthermore, this emphasizes why a new model is required since models fitted to passenger cars won't work for an EV with different traffic rules and route possibilities.

1.2 Aim and Limitations

This thesis treats route calculation regarding emergency vehicles (EVs), specifically how route calculation can be used to predict an EV's position. This is a continuation of a previous thesis [1] where a position was predicted using the A* algorithm to predict the route for an EV. In this thesis, the weight calculation in the road network was done by manually testing different dependencies and parameters on a small sample set of distress calls. The aim is to optimize the weight calculation of this road network, which in the future could enable a warning system for road users on the predicted EV route to ease the process of paving way for the EV. The optimization method used was competitive swarm optimization (cpso) which is an stochastic optimization method [6]. However, the optimization results are limited by the road network data available since it only contains five different categories of road size and five categories of road speed, its not possible to accurately calculate the actual cost of taking each road by using five discrete values for speed and size. This is due to the fact there is a much wider variety of roads than can be categorized by these two discrete variables, for example, some roads may have different degrees of curviness, which affect how fast they can be traversed, and there are more than five different speed limits and sizes of roads in the entire Swedish road network.

An SQL database of missions was used where start position, target position, and all available positions between these points are stored. Due to the running time of the algorithm implemented in the previous thesis, a limited number of missions and iterations were done when optimizing the parameters. This could impact the performance of the resulting algorithm since some cases may be over/underrepresented in the data set, such as distress calls inside/outside large cities. The performance metric used to evaluate the weighting was the root mean square value of the average difference in meters between the predicted position at a given time and the actual position. Furthermore, only polynomial size, speed limit, and length dependencies were considered when calculating the weights.

1.3 Definition of the Project's End Goal

This thesis aims to validate and improve the service developed by Alfred and Jakob. Specifically, the goal is to obtain the optimal weighted distance calculation that results in the best predictions for most probable route and for the EVs future positions along this route during distress calls.

2
Background

The purpose of this chapter is to review the previous thesis [1] work to provide context to this thesis. The earlier thesis was authored by Alfred Arvidsson and Jacob Hendén, who were pursuing master's degrees in data science and AI. Their report outlined a problem formulation consisting of two primary objectives. The first objective was to devise an algorithm that generates a set of routes with a specified probability and predicts the current position of the EV along the most probable route, given an emergency vehicle's starting position and destination. The second objective was to integrate the algorithm and visualization into Carmenta's Trafficwatch framework, which would be available as a service called the Route Probability Service.

2.1 Previous Work

The foundation of this thesis is built on the prior research work conducted by Alfred and Jacob on the "route probability service" implemented in the Carmenta Trafficwatch system [1]. This service has been designed to determine the most probable route an emergency vehicle would take during a distress call by considering their location and final destination. Their current location and destination would be received from SOS-alarm. To accomplish this task, the researchers employed the A* search algorithm to identify the path in the road network graph that leads to the target node while incurring the lowest possible cost. Since the main objective of the route probability service is to determine the most probable route an EV would take and its position along that route, it is crucial to ensure that the cost of each edge in the graph accurately represents the time cost of a driver choosing the corresponding road.

2.1.1 Road Network Handler

The road network handler is a software function in the route probability service responsible for loading the road network graph data from an SQL database for the route probability service. However, instead of loading the entire Swedish road network, the handler extracts a small relevant subpart of Sweden's road network, which is calculated as a bounding box that includes the start and end nodes, with an additional 0.1 degree of latitude and longitude added to each side. This approach is optimized for memory usage and processing speed, as loading the entire network for each operation is impractical and unnecessary.

To ensure the accuracy of the edge costs in the graph, the edges are assigned weighted distances that are calculated by

$$weightedDist = dist \cdot ((c_1 + c_2 \cdot size) \cdot (c_3 + c_4 \cdot speed)) \tag{2.1}$$

instead of just distances when loading the road network. The parameters had the following values $c_1 = 0.1, c_2 = 0.6, c_3 = 0.1, c_4 = 0.7$. This weighting is critical to the main objective of the route probability service, which is to determine the most probable route an EV would take. Using only distances would result in the A* algorithm only calculating the shortest route, which may not be the most probable route for an EV during a distress call.

The road network handler also incorporates a feature called position estimator, its job is to estimate the position of an EV once a route is aquired. This is done by setting a speed of the vehicle and let it traverse the graph, this speed is set to 20 weight units per second.

2.1.2 Determining the Route Score

The route generated by the A* is referred to as the main or most probable route. Evaluating the probability of the main or most probable route involves comparing its total weight to that of alternative routes. To obtain these alternative routes, nodes are removed randomly from the main route, and the most probable route is recalculated for each resulting configuration. This results in multiple routes with varying route scores, which can then be used to assess the likelihood of the road user selecting the main route. By examining the ratio between the total weight of the main route and that of the alternative routes, one can obtain a more nuanced measure of the probability score for the main route. Below 2.1 is a image displaying the main and alternative route for a EV.

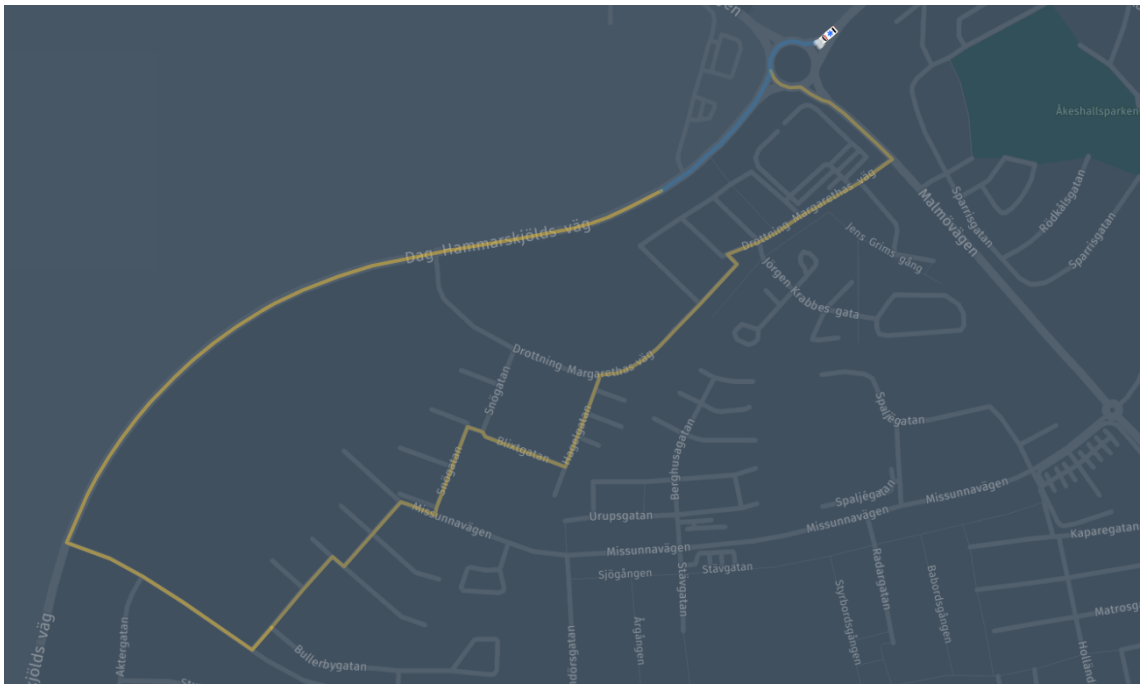


Figure 2.1: Image depicting two predicted routes for EV. The route with a blue segment represents the main route, while the other route is an alternative route with a higher route score. In the image we see a vehicle situated in the top right corner, indicating its current location, while the destination is in the bottom left

2.1.3 Training Data

Previously, the algorithm's validation was deemed unfeasible due to the unavailability of SOS data. In light of this, the algorithm's performance had to be evaluated by providing it with a start and end position and verifying the credibility of the generated route through visual examination. Upon assessing the algorithm's performance, the authors of the thesis deduced that the primary route identified was usually the quickest, with alternative routes often being reasonable options, albeit some of which could be viewed as long detours [1]. This outcome was attributable to the weight equation utilized by the algorithm, which had not been validated or optimized against genuine data. It is worth noting that the objective of an emergency vehicle is not to identify the shortest route, but rather the fastest, which can sometimes entail selecting a longer route with more significant speed limits and road size, resulting in a higher average speed than the shortest route. However, at the time of writing our report, SOS had initiated a partnership with Nordic Way3 that would entail the sharing of live data. This development allowed us to verify the accuracy of the weight equation used by the A* algorithm.

3
Theory

This chapter presents the relevant theory to the methods used in this thesis. This includes the following areas optimization, algorithms, data handling and graph theory.

3.1 Pre-processing

Pre-processing is a fundamental and crucial phase in machine learning and optimization. It involves converting raw and unstructured data into a more refined and organized format to increase training performance and create a more effective model. The pre-processing step also involves identifying and removing data points plagued by errors, inconsistencies, missing values, duplicates, and other unreliable and substandard data sources. In addition, the pre-process will help the learning model identify meaningful patterns and relationships within the dataset [7]. In many instances, datasets may have incomplete or missing values that require attention and remediation, often involving deleting or correcting the data. Another vital aspect of pre-processing is the identification of correlated features and the elimination of redundant information. Failure to pre-process the data can significantly and adversely affect the results obtained from the learning model. For example, splitting the dataset into train and test sets effectively ensures the model is overfitted on a specific data set. Overfitting refers to the situation where the analysis produced, only correlates with a particular dataset, resulting in the model's inability to fit another dataset.

3.2 Graph Theory

Graphs are a handy mathematical tool with many applications in computer science [8]. They can be used to represent many real-life problems and can provide useful insights into problems. An example is the traveling salesman problem: finding the shortest path for a salesman who visits each town once and returns to the starting city. A graph is a finite, none empty set V of elements called nodes and a set of edges E where each edge consists of two elements from V. Each edge $e_j = (v_i, v_j) \in E, \{v_i, v_j\} \in V$ represents a connection between nodes v_i and v_j. We denote a graph with sets V and E as $G(V, E)$ where the number of nodes and edges are written as $|V|$ respectively $|E|$. Another useful feature in many graph problems is to let each edge have a value associated with it, referred to as its weight. The traveling salesman problem can be converted to a graph problem using these

notations. This is done by representing each city as a node and creating edges between all nodes. Furthermore, we let the distance between two cities be the weight of each edge. Now, the problem is equivalent to finding the path with the smallest total edge weights that visits all nodes once and returns to the start node [9]. This is a good representation of the problem because graphs have many known algorithms and theories surrounding their traversal. Hence representing a problem as a graph, known theory can yield useful insights and possible solutions to the problem at hand. Moreover, there are several types of graphs. One which is central to this paper is directed graphs where each edge also has a direction that is edge $e_j = (v_i, v_j)$ implies that one can only move from node v_i to node v_j but not the other way around. This can be used when representing a road network as a graph where each edge is a segment of a road since generally, on roads, only one directional travel for each lane is allowed, and to switch direction, one must switch lanes.

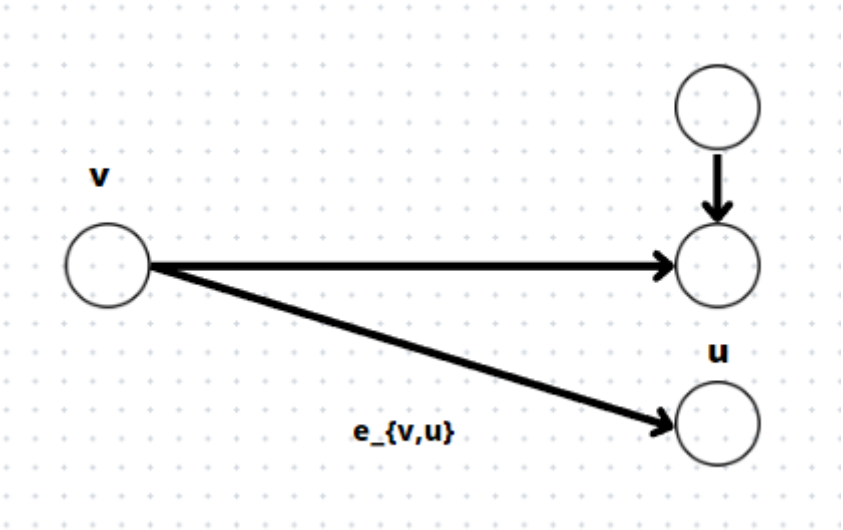


Figure 3.1: Example of a directed graph. Circles represent nodes, while arrows represent directed edges.

This thesis used a graph representation of the road network designed for EVs. In this graph, each node represents a road intersection, and the edges represent the roads between intersections. The edges are directed to indicate the direction in which roads can be traveled. The edges are bidirectional for each road that can be traversed both ways. Furthermore, on longer roads such as highways, there could be gaps in the rail for EVs, which are additional nodes without any intersections. Each edge in this graph gets weight from the algorithm described in section 4.3. This weight represents the traversal time of the EV, and when predicting positions, a constant weight/second speed is assumed. This graph satisfies all requirements for the A* algorithm to be applicable, and since there is always some set of edges connecting any two nodes in the graph, a path from the starting node s to destination t is guaranteed to be found [10]. The guarantee of finding such a path means that the graph is connected and comes from the fact that it represents roads and intersections, and there is always some route from any intersection to any other in the Swedish

road network.

3.3 A* algorithm

A* is a path search algorithm that works for weighted graphs. Given a graph $G(V, E)$, starting node $s \in V$, target node $t \in V$ and edge costs $w_{e_{i,j}}$ for all $e_{i,j} \in E$, it calculates the path with minimum total cost from s to t if such a path exists. The path is calculated by creating a tree of paths from s and adding one edge at a time to a chosen path. The edge added during each iteration is determined by the node n, which minimizes

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost from s to n and $h(n)$ is the heuristic function that estimates the cheapest path from n to t. To guarantee that A* finds the shortest path, an optimistic heuristic is required such that

$$\forall e_{x,y} \in E, h(x) \leq w_{e_{x,y}} + h(y).$$

A* stops when the path it extends goes from s to t or no paths are eligible for an extension, equivalent to no path connecting s to t in G. The time complexity of A* is $O(|E|)$ since every edge is considered once in the worst case. The space complexity of A* is given by $O(|V|)$ since all nodes present in the paths are stored. [11] [3]

3.3.1 Pseudo Code

The A* algorithm and how it works step by step can also be described using the following procedure:

- Set $f(i) = \infty$ for each node $i \in V$. Then set $g(s) = 0$ and $f(s) = g(s) + h(s)$, finally set $S = s$
- Find node $v \in S$ such that $f(v) \leq f(i), \forall i \in S$, set $S = S - \{v\}$.
- If $v = t$ the shortest path has been found, otherwise continue to the next step.
- For all nodes j where $\exists e_{v,j} \in G$, if $g(j) > g(v) + w_{e_{v,j}}$ update $g(j) = g(v) + w_{e_{v,j}}$ then set $f(j) = g(j) + h(j)$ and if $j \notin S$ set $S = S + \{j\}$. After all updates are made repeat the process from the second step.

Once the path from s to t has been found it can be recovered through backtracking. [12]

3.4 Black Box Optimization

Optimization problems can generally be formulated as finding the optimum of an objective function $f(\vec{x}), \vec{x} \in \mathbf{R}^n$ where $f : \mathbf{R}^n \to \mathbf{R}$. This optimum is found by either maximizing or minimizing f. There are several algorithms for solving this type of problem, some examples are gradient descent, the Ellipsoid method, and the Simplex algorithm. Effective optimization algorithms exploit known features of f to find an optimum such as fs differentiability, linearity, or other known properties of f. [13] [14]

However, black box optimization (BBO) involves optimizing an objective function f with no known properties. This means that any method which assumes something about f, such as differentiability, is not guaranteed to work. Moreover, this leads to the fact that only the function's value at different points $\vec{x_0}$ can be used to find the optimum (these values can however be used to try and estimate the derivative of f). Furthermore, BBO problems usually suffer from long computation times as well [13], which is also the case in this paper where each objective function evaluation takes $\approx 20s$ depending on the number of EV missions simulated. Another difficulty associated with BBO is that the number of parameters that should be optimized can also be unknown. Finally, when dealing with BBO, it's difficult to determine whether a solution is local or global optimum of f due to insufficient information about the objective function.

3.5 Stochastic Optimization

There are several approaches to optimize discrete and nonlinear objective functions, one popular approach is stochastic search algorithms. Some stochastic algorithms imitate natural processes seen in nature, such as evolution or swarm behavior, they are generally useful for searching through a parameter space efficiently and yield satisfactory results but are not guaranteed to find a global optimum. These methods are especially well suited for problems where

I The derivative of the objective function is unknown or expensive to calculate.
II The objective function is highly non-linear and likely to get stuck in poor local optima.
III There are few parameters and a finite parameter space to search.

Moreover, one key point about stochastic search algorithms is that they contain some stochastic element, and hence they don't guarantee convergence as opposed to gradient-based search methods. The same optimization may need to be run several times to ensure an optimum has been found. [15]

Let's consider the case described in section 3.4 where an objective function $f(\vec{x})$ is to be optimized, and we need to learn more about f to use any deterministic method. If we can limit the search space, which is defined as a finite subset Ω of parameters where the optimum $\vec{x}^*$ lies

$$\vec{x}^* \in \Omega \in \mathbf{R}^n. \tag{3.1}$$

In this case, a search algorithm can be implemented for finding $\vec{x}^*$, but it's usually not practical to exhaustively search the entire Ω space, especially if it's high dimensional. However, using stochastic search algorithms, one can efficiently search the variable space Ω and thus likely avoid local optimums. In stochastic search algorithms such as swarm optimization or genetic algorithm, the Ω space is searched through random initialization, and the search algorithms don't require the gradients of the objective

function but rather only rely on the actual score/value of the function to guide the search making them suitable for optimizing discrete functions. This is because of population-based approaches such as GA and CPSO, where each iteration consists of several individuals with their own set of parameters and individuals with higher scores are more likely to survive to the next generation. Since the score is the value of the objective function, after several iterations, the population will converge to a set of individuals/parameters, which yield a robust local optimum only using the value of the function. However, there is no simple condition for convergence. As such, the algorithms must run several times, and if no significant change is made for several iterations, the optimum is considered robust. To handle the constraints on variables, for example make sure that $\vec{x} \in \Omega$, penalties to all values that don't satisfy this condition can be applied. [16]

3.6 Competitive Particle Swarm Optimizer

Particle Swarm Optimization (PSO) is an effective stochastic optimization technique that takes inspiration from the collective behavior of natural swarms, such as those exhibited by flocks of birds or schools of fish [6]. The algorithm utilizes a population-based approach, where multitudes of particles traverse through an n-dimensional search space in search of the optimal solution. By mimicking the cooperative behavior of these natural systems, PSO will try to optimize a problem by improving a candidate solution.

A set of parameters represents every n-dimensional particle. In each iteration, PSO evaluates every particle in the swarm, using a loss function that yields a numerical value representing its fitness. The lower the loss value, the closer the particle's parameters are to the optimal solution. Selecting a suitable loss function depends on the addressed problem.

The PSO algorithm begins with a uniform initializing the swarm of particles throughout the n-dimensional search space. Then, each particle iteratively moves through the search space and updates its velocity and position based on its personal best and the global best among all particles in the swarm. To accomplish this, PSO employs two equations that update the velocity and position of each particle. The equations utilize control parameters, including an inertia weight (ω) and acceleration coefficients (c_1 and c_2), to influence how rapidly the particle moves in pursuit of the optimal solution.
The first equation,

$$V_i(t+1) = \omega V_i(t) + c_1 R_1(t)(pbest_i(t) - X_i(t)) + c_2 R_2(t)(gbest(t) - X_i(t)), \quad (3.2)$$

represents the velocity update equation. The movement of a particle is influenced by two components - the cognitive component, which considers the particle's personal best position, and the social component, which takes into account the best position among all particles in the swarm. These components are represented by the equations $c_1 R_1(t)(pbest_i(t) - X_i(t))$ and $c_2 R_2(t)(gbest(t) - X_i(t))$ respectively.

The second equation,

$$X_i(t+1) = X_i(t) + V_i(t+1), \tag{3.3}$$

represents the position update equation. It updates the particle's position based on the updated velocity obtained from the first equation.

Overall, the PSO algorithm effectively harnesses the collective behavior of natural swarms to optimize the search process and efficiently navigate the search space, which has been used successfully in various applications. However, research shows that PSO performs poorly when the problem has many local optima or is high dimension. As a result, there have been many attempts to introduce new variants of PSO to enhance its search performance. However, due to its strong influence on global and personal best, it often suffers from premature convergence [6]. Therefore, a relatively recent evolutionary technique known as Competitive Particle Swarm Optimization (CPSO) has been suggested by researchers, as documented in [17]. This particular method eliminates both the particle best and global best, paving the way for a fresh approach to the problem. This evolutionary framework updates each iteration by performing a competitive mechanism between particles in the swarm. During this competition, the defeated particle will undergo an update by mimicking the winning particle, whereas the winning particle will undergo an update via mutation. By removing the reliance on personal and global influences, this approach effectively eliminates the potential for premature convergence dependent on historical values. Additionally, This method negates the need to record historical values.

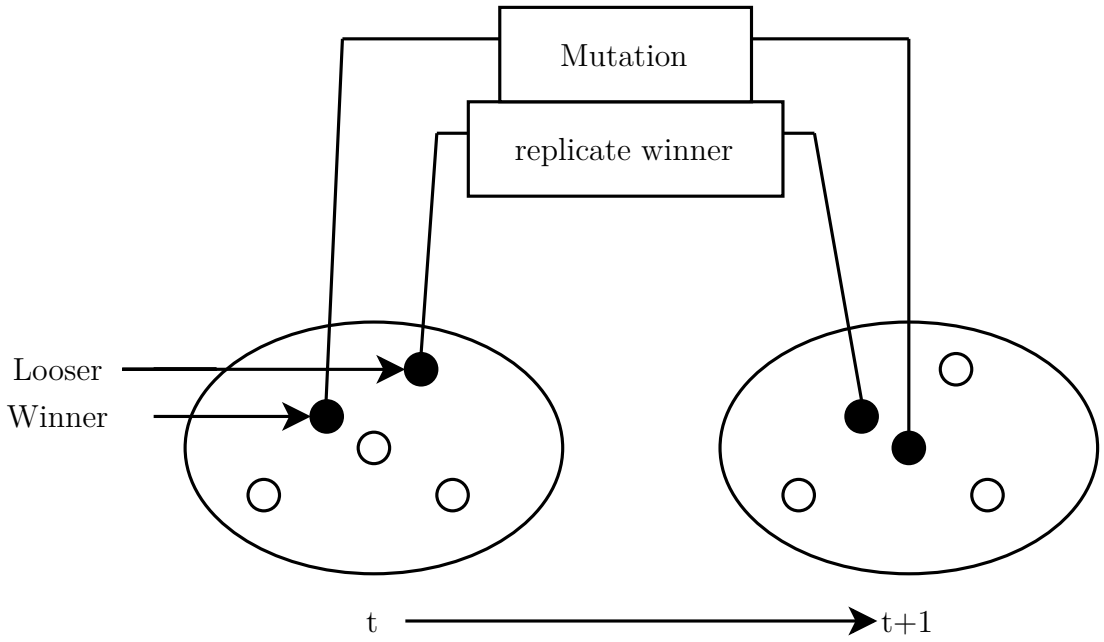


Figure 3.2: This image shows the general updating rule of CPSO where the winner will undergo mutation while the loser replicates the winner.

We can establish the location and speed of the champion in the k-th round as $X_{w,k}(t)$ and $V_{w,k}(t)$ respectively. Likewise, we can determine the position and velocity of the defeated player as $X_{l,k}(t)$ and $V_{l,k}(t)$. After the k-th competition, the velocity of the losing particle updates as follows:

$$V_{l,k}(t+1) = R_1(k,t)V_{l,k}(t)+R_2(k,t)\left(X_{w,k}(t) - X_{l,k}(t)\right)+\varphi R_3(k,t)\left(\bar{X}_k(t) - X_{l,k}(t)\right), \tag{3.4}$$

where $R_1(k,t)$, $R_2(k,t)$, and $R_3(k,t)$ are randomly generated vectors within the range $[0,1]^n$. Additionally, $\bar{X}_k(t)$ represents the average particle position in the k-th competition, and φ controls the influence of $\bar{X}(t)$.
In the update equation 3.4, the first term, $R_1(k,t)V_{l,k}(t)$, serves as an inertia term that ensures the search stability of the algorithm, similar to regular PSO. However, instead of using a fixed inertia coefficient like ω, CPSO will utilize a randomly generated vector $R_1(k,t)$. The second term, $R_2(k,t)\left(X_{w,k}(t) - X_{l,k}(t)\right)$, is called the cognitive term. Unlike regular PSO, the losing particle learns from the winning particle and adapts accordingly rather than relying on the influence of personal best. This mechanism may be more biologically realistic, as the swarm does not require a historical record of the best particles [6]. Finally, the third term, $\varphi R_3(k,t)\left(\bar{X}k(t) - Xl,k(t)\right)$, is known as the social term. Here, the losing particle learns from the mean position of the swarm rather than the global best, which also makes biological sense for the same reason as above. The position of the losing particle will update similarly to regular PSO

$$X_{l,k}(t+1) = X_{l,k}(t) + V_{l,k}(t+1) \tag{3.5}$$

This modification of PSO will retain its algorithmic simplicity where $X_{w,k}$ will be pushed into generation $t+1$ and $X_{l,k}$ will update with equation (3.4) and (3.5). The following pseudo-code shows the simplicity of the algorithm.

$t = 0$
$X_w = \varnothing$
$X_l = \varnothing$
while condition not satisfied **do**
    $U = P(t)$
    $P(t+1) = \varnothing$
    while $U \neq \varnothing$ **do**
        Randomly select two particles X_1 and X_2 from U
        if If the evaluation of X_1 is greater than the evaluation of X_2 **then**
            $X_w = X_1$
            $X_l = X_2$
        else
            $X_l = X_1$
            $X_w = X_2$
        end if
        Push X_w to set $P(t+1)$ and update X_l according to equations (3.4) and (3.5).
        Remove X_1 and X_2 from U.
    end while
    Increment t by 1.
end while

The time complexity of this algorithm is $\mathcal{O}(mn)$, where m is the population size and n is the search dimensionality.

4
Methods

This chapter details the data processing from SOS raw data, modifications made to the existing algorithm and code implementation. Furthermore, the optimization method for minimizing the objective functions and acquiring a candidate set of parameters is described.

4.1 SQL-database

When SOS-alarm receives a distress call, they will initiate a process by transmitting pings containing a timestamp, the current location, vehicle identification, and approximate destination to NordicWay 3. This transmission of pings will be continuous, occurring roughly every 30 seconds until the mission is either completed or aborted. A framework was established to capture and store this transmission of pings in a SQL database 4.1. In total, 200 000 pings were stored, representing approximately 2000 missions.

Table 4.1: Table showing an example snapshot of the SQL database.

id	startLat	startLon	endLat	endLon	positionTime
...-9220	64,59...	18,65...	64,59...	18,65...	2023-01-23 09:10:07
...-9220	64,59...	18,65...	64,59...	18,65...	2023-01-23 09:10:27
...-9410	63,15...	16,27...	63,15...	16,27...	2023-01-23 09:26:42
...-9410	63,15...	16,27...	63,15...	16,27...	2023-01-23 09:27:12
...-9410	63,15...	16,27...	63,15...	16,27...	2023-01-23 09:27:42
...-9410	63,15...	16,27...	63,15...	16,27...	2023-01-23 09:28:12
...-9410	63,15...	16,28...	63,15...	16,27...	2023-01-23 09:28:42
...-9410	63,15...	16,27...	63,15...	16,27...	2023-01-23 09:29:12

The SQL-database shown in Table 4.1 is an example snapshot of the stored pings. The table contains a unique id that includes information about the vehicle and the mission it is currently on. By examining the ending in the vehicle id, we can determine if the vehicle is a helicopter, ambulance, firefighter truck, or other, which can be sorted in the pre-processing step. The startLat and startLon columns contain the vehicle's current location at the given time, while the endLat and endLon columns represent the approximate target destination of the distress call.

4.2 Pre-processing

Before feeding the data into our optimization model, pre-processing steps were conducted to prepare the data. Firstly, a thorough elimination process was undertaken to filter out unsuitable or faulty missions for the optimization model. The elimination involved excluding missions with less than three pings, as this typically indicated shorter distances traveled; by excluding these missions, the analysis could focus on missions that were likely to cover more considerable distances, enabling more meaningful insights and conclusions from the data.

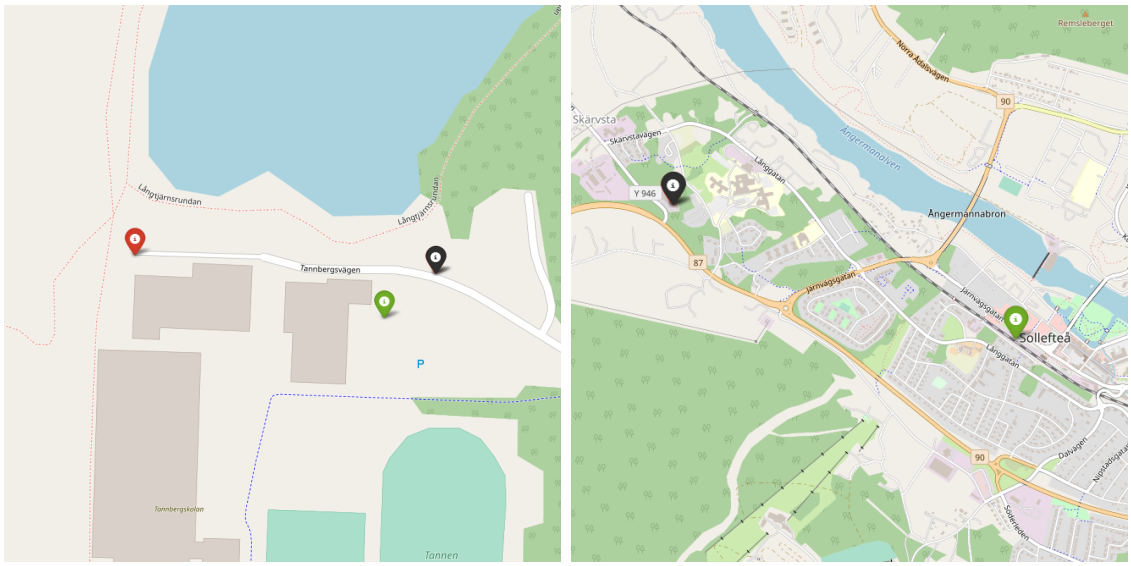


Figure 4.1: The image shows a mission eliminated from the dataset as it contained less than three pings.

Furthermore, we decided to exclude missions exhibiting clear faults, such as those featuring helicopter routes or rescue operations in mountainous regions without roads.

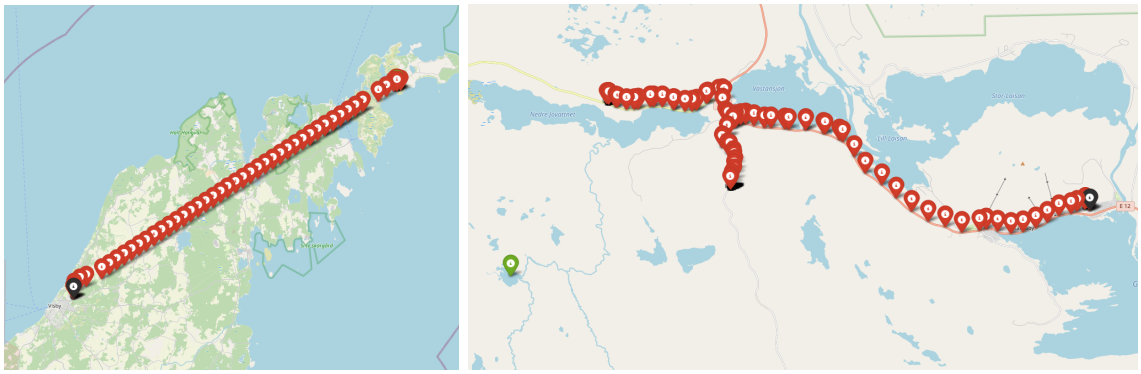


Figure 4.2: The displayed images illustrate missions that possess evident faults. The image on the left displays a mission conducted by a helicopter. In contrast, the image on the right showcases a rescue mission executed in a mountainous region inaccessible by road users.

These types of missions presented significant anomalies that could have had a detrimental impact on the accuracy of the analysis. As a result, removing them from the dataset was considered necessary. Additionally, the missions were trimmed by excluding the pings before the vehicle moved a threshold of 500 meters.

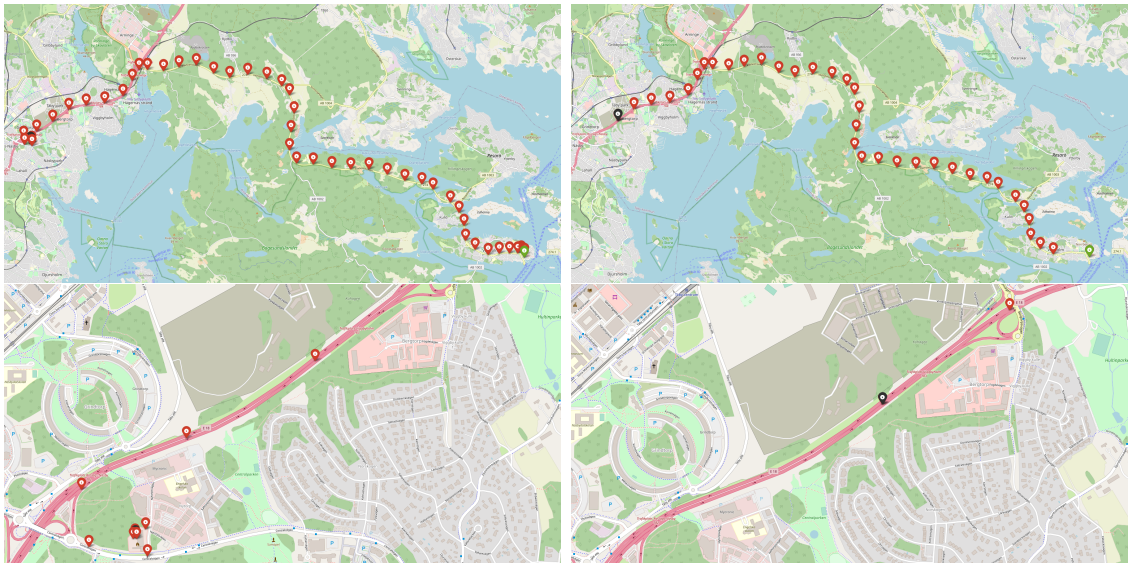


Figure 4.3: The four images illustrate the missions after the preprocessing stage, where pings that would cause noise in the dataset have been removed. The image on the right shows the mission after preprocessing, and the lower image depicts a zoomed-in view of the beginning of the mission.

Doing so ensured that the analyzed data represented the vehicle's actual movement and progress toward its destination. Furthermore, any pings before this threshold would have only added unnecessary noise and confusion to the analysis.

The mission data used was split into training and test data. This is done to validate how well the optimized parameters in the weight equations, described in section 4.3, perform on unseen data. The prepossessing selection of the 2000 missions in the SQL database yielded 500 missions, of which 300 were used for optimization and 200 for testing.

4.3 Road Network Handler

The road network handler's function is to retrieve a segment of the entire road network of Sweden by querying the Traffic Watch cloud-based SQL server. To expedite each query, enhancements were made by downloading the SQL database locally and modifying the road network handler to query the local database instead. An illustration of the query and the table extracted from the SQL database is presented below.

```
"DECLARE @polygon GEOMETRY = geometry::STGeomFromText('
POLYGON((f"{fromLon} {fromLat}, {fromLon} {toLat},
{toLon} {toLat}, {toLon} {fromLon}, {fromLat}"))', 4326);
SELECT * FROM RoadNetworkWithDists
WHERE GEOM.STIntersects(@polygon) = 1";
```

Table 4.2: Table showing an example snapshot of the road network.

id	refin	nrefin	speed	size	dir	geom	ilat	ilon	olat	olon	dist
...4	85...	85...	7	5	B	0xE6...	59,5...	18,0...	59,5...	18,0...	10,7...
...6	85...	85...	7	5	B	0xE6...	59,5...	18,0...	59,5...	18,0...	99,3...
...7	85...	85...	7	5	B	0xE6...	59,5...	18,0...	59,5...	18,0...	261,4...
...8	85...	85...	7	5	B	0xE6...	59,5...	18,0...	59,5...	18,0...	417,3...
...9	85...	85...	7	5	B	0xE6...	59,5...	18,0...	59,5...	18,0...	44,8...
...0	85...	85...	7	5	B	0xE6...	59,5...	18,0...	59,5...	18,0...	683,1...

The data extracted from a sample query is presented in Table 4.1, where each road segment is displayed as a separate row. The 'id' column denotes a unique road identifier, while the 'refin' and 'nrefin' columns indicate the connecting node IDs of the road's edge. The 'size' column represents the road's size, ranging from 1 to 5, while the 'speed' column denotes the speed category of a road. The speed category is not solely based on a particular road's speed limit, as additional factors such as speed bumps influence the category and can lower it. Since the edges are directional, 'ilat' and 'ilon' represent the latitude and longitude of the source node, respectively. In contrast 'olat' and 'olon' represent the sink node. 'dist' denotes the distance of each road, which is already precomputed, thus eliminating the need for additional computations.

The road network handler is responsible for calculating the weighted distance of each road segment, which represents the likelihood that a user will choose that particular road. In the previous thesis [1], a linear equation was used to calculate the weight (2.1). Several different weight equations have been experimented with to determine the best fit for the data. These equations are called weight equation one, weight equation two, and weight equation three.

Weight equation one

$$weight = a_1 + (a_2 * speed + a_3 * speed^2) + (a_4 * size + a_5 * size^2) \tag{4.1}$$

Weight equation one is a polynomial function of degree 2. Therefore, it requires optimization of five parameters, denoted as a_1 to a_5.

Weight equation two

$$weight = (b_1 + b_2 * speed + b_3 * speed^2) * (b_4 + b_5 * size + b_6 * size^2). \tag{4.2}$$

Weight equation two is similar to the previous equation with one additional term and allows for cross-terms between 'size' and 'speed.' The equation has six parameters, denoted as b_1 to b_6.

Weight equation three

$$weight = (d_1 + d_2 * speed) * (d_3 + d_4 * size). \tag{4.3}$$

In contrast, weight equation three is a more straightforward equation that only includes a set of four parameters, denoted as d_1 to d_4. In addition, this equation removes the second-order terms present in the previous equations. As a result, it is anticipated to expand each parameter's search space, yielding better calculation process efficiency.

The weight structure of the weight equations Once the weight of each road segment has been determined, it is multiplied by the road segment's distance to calculate the weighted distance. This process is represented by Equation 4.4. The result is a more accurate measure of the likelihood that a user will choose a particular road, enabling better route planning and optimization,

$$weigthedDist = weight * dist. \tag{4.4}$$

Since the speed of the EV is set to 20 weights units per second, the optimized road network weights will be fitted to this speed.

4.3.1 Bounding Box

The route calculator incorporates a feature known as the bounding box, which loads a reduced section of Sweden's road network to improve efficiency and memory usage. In the earlier algorithm, this feature was defined by utilizing the starting and ending positions to establish the bounding box's corners and then extending it by 0.1 units of longitude and latitude. However, this method may only sometimes be optimal since specific routes may extend beyond the defined geographical limits.

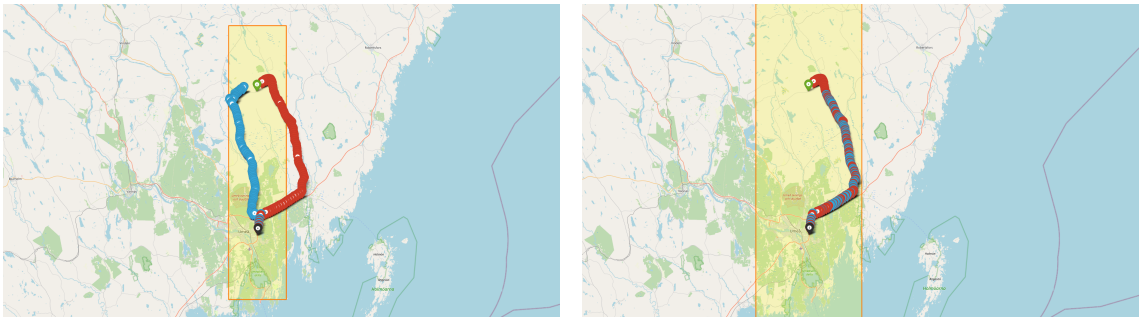


Figure 4.4: The image shows a bounding box highlighted in a bright yellow color. The left image displays an implementation of the bounding box, which involved adding 0.1 units of longitude and latitude to the starting and ending positions. The image displays a modified approach on the right-hand side, adding 0.15 units to the coordinates. The blue markers represent the estimated position, while the red markers represent the actual pings of the emergency vehicle. When the bounding box is expanded, the route calculator selects a different route that aligns with the EV's actual movement.

Interestingly, the image 4.4 displays that by increasing the size of the bounding box, the route calculator selects a different route, which aligns with the actual movement of the EV. A modified version addressed this issue by bounding the box around the EV's actual pings instead of relying solely on the starting and ending positions. This was accomplished by taking the minimum and maximum values of latitude and longitude of the sequence of pings to define our bounding box. In addition, we added 0.1 units to the box to ensure the entire route was included.

4.4 Objective Function

This thesis aims to optimize the weights of the road network graph used for predicting the routes and positions for EVs. Specifically, a weight calculation algorithm was to be optimized where the weight of each road was calculated according to (4.1), (4.2), and (4.3). Since the output from the modified algorithm described in section 4.5 outputs a series of positions in latitude and longitude values for each mission, a good starting point would be to consider the distance between these predicted points and the actual positions of the ambulance during the mission. The actual position relayed through pings is also in latitude and longitude coordinates; a simple conversion to meters was made according to

$$a = \sin^2(\Delta\Phi/2) + \cos(\Phi_1) \cdot \cos(\Phi_2) \cdot \sin^2(\Delta\lambda/2) \tag{4.5}$$

$$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a}) \tag{4.6}$$

$$d = \text{R} \cdot c \tag{4.7}$$

Here Φ represents the latitude, λ the longitude and R the earths radius ($\approx$ 6,371 km). Note that both the longitude and latitude must be in radians which could be easily fixed by multiplying them by $\pi/180$. Using the distance between actual and predicted positions considers both the route selected and the estimated position since if the wrong route is selected, the function will increase due to all estimated positions being on the wrong route which in turn yields a large output of the objective function. Furthermore, since we are measuring the distance between the predicted and actual position, it's clear that optimizing this function will yield better position prediction since a smaller distance between the predicted and actual position implies a better prediction.

Two objective functions utilizing the distance between prediction and actual positions were implemented. The first version, f_1, use the mean of the 10 first pings, and the other f_2 uses the sum of the 40 first pings. The first function f_1 was made to ensure that the function emphasizes predicting correctly in the near future, hence each consequent prediction was decreased by a factor $i \cdot 0.1 + 1$ where i is the ping number. The second function f_2 was implemented using the sum of the 40 first pings to emphasize correct road selection. This was done since selecting the wrong route has a larger impact when looking at more pings, and using the sum makes it so that selecting the wrong route is heavily penalized. Furthermore, the route-selecting process is more complex when looking at more pings since more decisions must be made. For the first objective function f_1, the mean distance between predicted and

actual positions was used to represent the average error between each mission's estimated position and ping. Finally, for both f_1 and f_2, the root mean square error was taken as the function's output.

The objective functions mathematical representation is shown below in equations (4.8) and (4.9) where f_1 uses the mean of 10 pings and f_2 the total error of 40 pings for each mission, respectively.

$$f_1 = \sqrt{\frac{\sum_j^m \left(\frac{1}{n_1} \sum_i^{n_1} \frac{d(X_i^j, Y_i^j)}{0.1 \cdot i + 1} \right)^2}{m}}, \tag{4.8}$$

$$f_2 = \sqrt{\frac{\sum_j^m \left(\sum_i^{n_2} d(X_i^j, Y_i^j) \right)^2}{m}} \tag{4.9}$$

Here X_i^j, Y_i^j represents the ith predicted and actual position of mission j respectively, n_1, n_2 are the maximum number of pings looked at in each mission and have the values $n_1 = 10$ and $n_2 = 40$ finally m represents the total number of missions used in the optimization. Note that $d(X_i^j, Y_i^j)$ is the distance between X_i^j and Y_i^j in meters.

4.5 Code Implementation

Several changes to the algorithm's structure were needed to optimize the algorithm's output devised in the previous thesis [1]. First, the algorithm needs to take the parameters as input and the starting and ending positions of the EV's journey to provide an estimated route. This was achieved by transforming the original code for the algorithm into an executable file, which can be run in Python by providing a command as an argument. This file is designed to execute a set of commands obtained from preprocessing. For instance, consider the following command:

```
command = "XX;XXXXXXX;X;XXX–XXXX,
63.17218,17.21252,63.092608,16.357664
30.0,60.0,90.0,120.0"
```

The command, which represents a mission for an EV, comprises three rows of information. The first row specifies the mission ID, including details about the vehicle. The second row contains the start and target destination coordinates. The third row determines the interval at which pings are received. This interval was then passed to to the position estimator which returned the predicted positions at the given time intervals. By passing a folder containing all missions that are to be used by the optimization algorithm, they can be executed, and an estimated position for each ping of each mission can be returned. Finally, the executable needs to be passed an argument containing a set of parameters that are to be tested, such as:

```
modelParameter = f"{a1},{a2},{a3},{a4},{a5}"
```

We can then run the function along with the set of parameters using the following code:

```
os.system(path + modelParameters)
```

Where path is the absolute path to the directory containing all missions/commands, this function estimates the vehicle's position at specified intervals. By comparing these positions to the actual positions in the commands, we can assess the performance of each set of parameters through the objective function.

4.5.1 Flowchart

Optimizing the parameters can be explained through a flowchart where the optimizer is the process where each individual in the CPSO algorithm passes its parameter values to the algorithm and gets predicted positions for each mission, which are then used to get a score. The process of the flowchart 4.5 occurs for each individual in each generation.

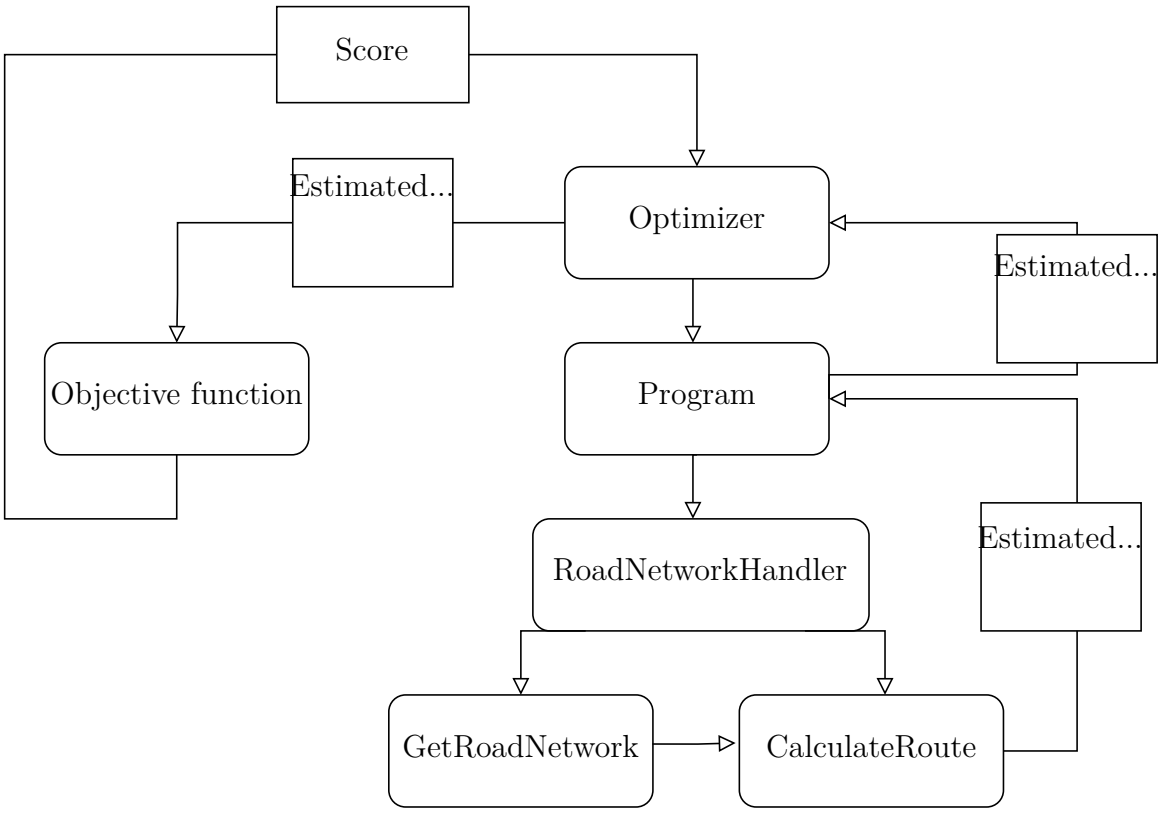


Figure 4.5: Flowchart showing the process and role of each component in the optimization algorithm. Estimated... are the estimated positions, Optimizer is the CPSO algorithm which first calls Program and then gets back estimated positions and calls the objective function to get a score for the parameters, Program is the modified algorithm from [1] which returns estimated positions, RoadNetwork handler utilizes the road network graph and the route calculator to estimate positions, GetRoadNetwork uses the road network SQL database and weight calculation algorithm to construct road network graph, CalculateRoute uses the mission information and the road network graph to calculate the route and estimate positions and ObjectiveFunction compares the estimated positions to the actual positions to return a score.

4.6 Competitive Particle Swarm Optimization

The training was done by implementing competitive particle swarm optimization (CPSO). The parameter space consisted of four to six variables depending on which weight calculations from equations (4.1), (4.2), or (4.3) were used. With respective search bounds;

$$\{a_1, a_2, a_3, a_4, a_5\} \in [0, 3]$$

$$\{b_1, b_2, b_3, b_4, b_5, b_6\} \in [0, 3]$$

$$\{d_1, d_2, d_3, d_4\} \in [0, 3]$$

Everything is analogous for using all three weight equations; therefore, only the example with five parameters attained from equation (4.1) will be used. Each particle p in the swarm consists of a set of parameters $[a_1^*, a_2^*, a_3^*, a_4^*, a_5^*]$ which are passed to the route calculation algorithm and scored using the objective function described in section 4.4. The bounds were set using the fact that the parameters must be positive since negative values could result in negative weights for certain roads, and as the weights indicate travel time, they must be strictly positive. For the upper bounds, it was found by trial that optimizing with values above 3 for any of the parameters yielded poor results. Hence three was set as an upper bound to ensure all reasonable values were in the parameter space.

The optimization was done using a population size of 100 and a maximum number of iterations set to 100. The control parameter $\varphi = 1$, and the initialization of the parameters is uniformly distributed. The reason for selecting a maximum number of iterations is that if the CPSO has yet to converge after 100 iterations, it's likely gotten stuck around some local optima. Therefore, the running time would be too long for a maximum larger than 100 iterations. A stopping criteria were also used, which required an improvement of at least 10 meters for f_1 and 100 meters for f_2 to continue. Furthermore, a constraint on the particles was enforced where any particle outside the set bounds would be placed on the edge of the parameter bounds by shrinking its velocity vector. This ensures that only feasible solutions are considered. Finally, the score and parameter values of the best particle for each iteration were recorded during simulations, and the all-time best score and parameters were also saved/updated during the optimization.

5

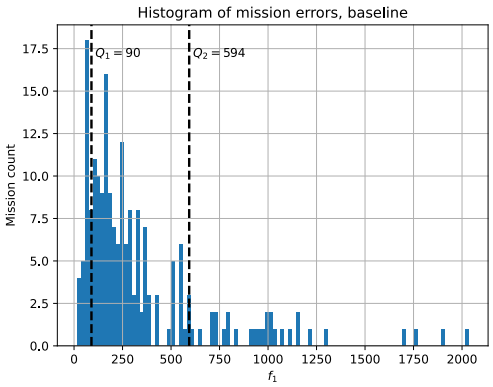
Results

This chapter will present the results achieved using the Competitive Particle Swarm Optimization (CPSO) method for various weight equations and objective functions. In the optimization of the weight equation, three different weight equations were used (4.1), (4.2), (4.3). These equations were multiplied by the distance of the road to obtain the final weight value, known as the weighted distance (4.4). In addition, for each weight equation, two objective functions, namely f_1 (4.8) and f_2 (4.9), were employed, as explained in section 4.4. In observing the progression of the optimization process, it becomes evident that the improvement of the best particle discovered in each generation gradually diminishes as the generation count increases. This decline continues until it reaches a predefined threshold upon which the training is stopped. The threshold for f_1 was set to 1 m and for f_2 it was 20 m which corresponds to roughly 0.2% of the respective baseline scores derived from the previous parameters.

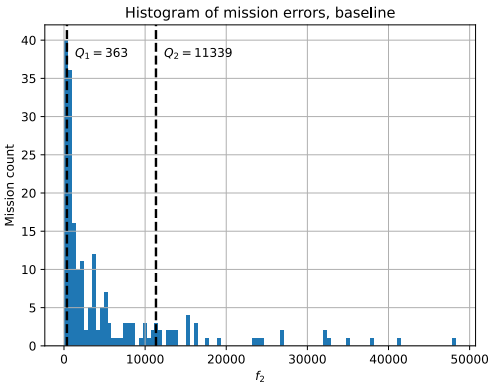
5.1 Baseline

To assess the effectiveness of the optimization results, a comparison was conducted with a baseline parameter set derived from a previous thesis [1]. The purpose of the baseline is to have a reference score against the optimization scores and to validate the algorithm developed in the previous thesis [1]. The weight equation in the baseline has the variables set to $c_1 = 0.6, c_2 = 0.1, c_3 = 0.7, c_4 = 0.1$, and uses the same equation structure as weight equation three 4.3. The baseline score was computed using the test dataset, and the results can be found in table 5.1. In our pursuit of minimizing the objective functions, we will present a score quota between the optimization score and the score of the baseline, a value greater than 1 corresponds to an improvement of the algorithm's performance. Consequently, larger values correspond to more significant improvements.

Moreover, the mission score distribution of the baseline parameters is analyzed, focusing on their error scores for objective functions f_1 (4.8) and f_2 (4.9). These distributions are visually presented in figure 5.1. The comparison of mission score distributions aims to determine whether the optimization has led to a general improvement in accuracy across all missions or if its impact is primarily focused on reducing scores by minimizing errors for outlier missions.



(a) Results obtained from objective function f_1



(b) Results obtained from objective function f_2

Figure 5.1: The histograms show the distribution of individual mission errors using the baseline weight equation (2.1), with evaluations performed on both objective functions f_1 (5.1a) and f_2 (5.1b). The quantiles Q_1 (below 15%) and Q_2 (above 85%) are use to categorize bad and good scores.

The table 5.1 showcases the error obtained from running objective functions f_1 and f_2 on the test dataset. These scores, labeled as the baseline scores for f_1 and f_2, provide a reference point for comparison.

Table 5.1: The table presents the results obtained from the parameters used in the previous thesis [1], evaluated on objective functions f_1 and f_2.

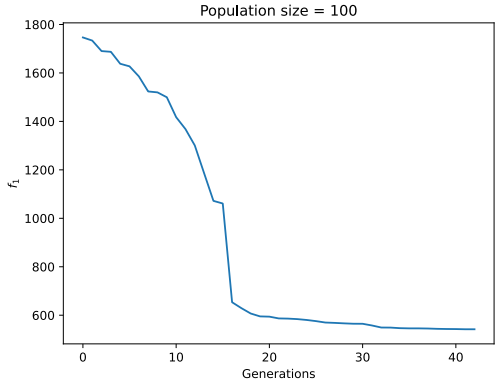
Obj	c_1	c_2	c_3	c_4	Baseline score
f_1	$60 \cdot 10^{-2}$	$10 \cdot 10^{-2}$	$70 \cdot 10^{-2}$	$10 \cdot 10^{-2}$	490
f_2	$60 \cdot 10^{-2}$	$10 \cdot 10^{-2}$	$70 \cdot 10^{-2}$	$10 \cdot 10^{-2}$	9838

5.2 Weight Equation One

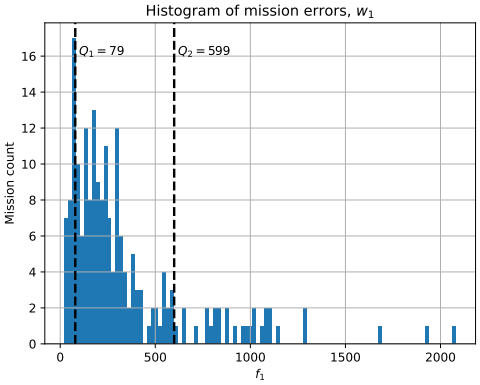
This section presents the achieved outcomes by optimizing parameters $a_1 - a_5$ in weight equation one (4.1). Within each iteration of the CPSO algorithm, individual particles undergo evaluation using either objective function f_1 or f_2. The results of utilizing the different objective functions will be separately presented to ensure clarity.

5.2.1 Optimizing Results Using Objective Function f_1

These results are derived from optimizing weight equation one (4.1), in conjunction with the objective function f_1. Figure 5.2 illustrates the optimization scores attained for various generations. Moreover, a histogram is provided to visually represent the distribution of missions based on their error scores (5.2b). Additionally, table 5.2 presents the final parameter values and their corresponding scores for both objective functions f_1 and f_2.



(a) Graph of the progression of the best particle score achieved through the CPSO algorithm over successive generations. The x-axis represents the number of generations, while the y-axis denotes the corresponding best particle score. The population size for this analysis is set to 100.



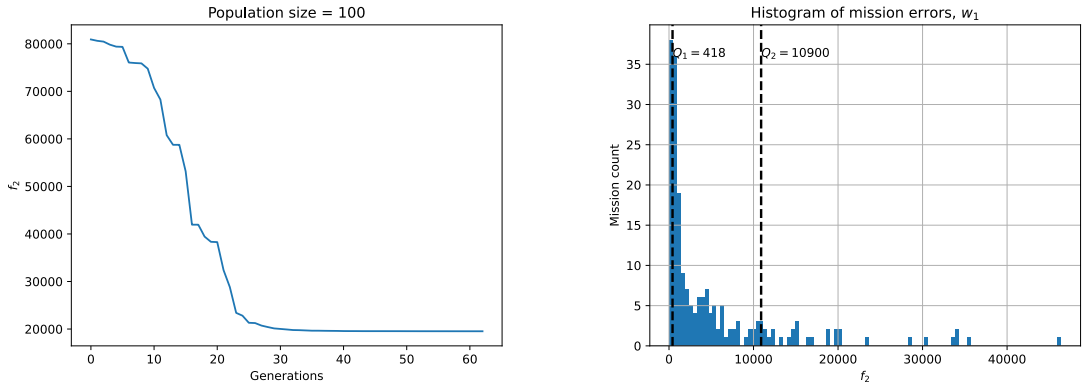
(b) The histogram presents the distribution of individual mission errors. These errors were evaluated using the best parameters identified within the training set, and the histogram specifically showcases the performance of these parameters evaluated on the test dataset. The y-axis is the mission count, and the x-axis is the error value.

Figure 5.2: Within the histogram 5.2b, two quantiles, denoted as Q_1 and Q_2, serve as thresholds to classify scores as good or bad. To elaborate, missions with scores below Q_1 (less than 15%) are categorized as having good scores, while missions with scores above Q_2 (greater than 85%) are deemed to possess bad scores. Overall, the figures provide an overview of the best particle score evolution and the distribution of mission errors, illustrating the performance analysis of the CPSO algorithm applied to objective function f_1 using weight equation one (4.1).

In observing the progression of the optimization process, it becomes evident that the improvement of the best particle discovered in each generation gradually diminishes as the generation count increases. This decline continues until it reaches a predefined threshold of 1 meters, upon which the training is stopped. Figure 5.2a visually represents this trend. It is noticeable that the objective function experiences a rapid decrease during the initial generations and subsequently converges after approximately 20 generations. The most favorable score achieved throughout this optimization was $f_1 = 484$. Interestingly, this outcome is nearly identical to the baseline score, showcasing a marginal improvement of 1%. By comparing the histograms displayed in Figure 5.1a and Figure 5.2b, one can see that the differences in distribution are also minimal. This is a distinct set of parameters that differs from the baseline yet performs equally as the baseline.

5.2.2 Optimizing Results Using Objective Function f_2

Similar to the results in figure 5.2, the corresponding outcomes obtained by employing objective function f_2 with the same weight equation are showcased in figure 5.3.



(a) Graph of the progression of the best particle score achieved through the CPSO algorithm over successive generations. The x-axis represents the number of generations, while the y-axis denotes the corresponding best particle score. The population size for this analysis is set to 100.

(b) The histogram presents the distribution of individual mission errors. These errors were evaluated using the best parameters identified within the training set, and the histogram specifically showcases the performance of these parameters evaluated on the test dataset. The y-axis is the mission count, and the x-axis is the error value.

Figure 5.3: The graphs show both the best particle score from CPSO after several generations (5.3a) and the distribution of individual mission errors (5.3b), using the f_2 objective function and the first weight equation (4.1).

To optimize weight equation parameters using the objective function f_2, a slightly higher stopping criteria was set compared to f_1. This is because f_2 summarizes the errors from 40 pings, resulting in generally larger values. The stopping criteria were set at 20 meters. Additionally, a histogram similar to the baseline distribution is presented for analysis. The best particle obtained achieved a score of $f_2 = 9172$, which was 7% better than the baseline score. This improvement indicates that the optimization process effectively enhanced the performance of the weight equation in predicting routes/positions. The histogram depicting the distribution of errors also provides valuable insights. Comparing it to the baseline histogram, we can observe that the optimized weight equation shifted toward lower error values. This shift indicates a reduction in higher error missions, leading to a more accurate prediction of routes/positions.

The table 5.2 summarizes the results from optimizing using weight equation one. We see a slight improvement of 1% and 7% using f_1, respectively f_2.

Table 5.2: Table of results form the first weight equation (4.1) using both objective functions.

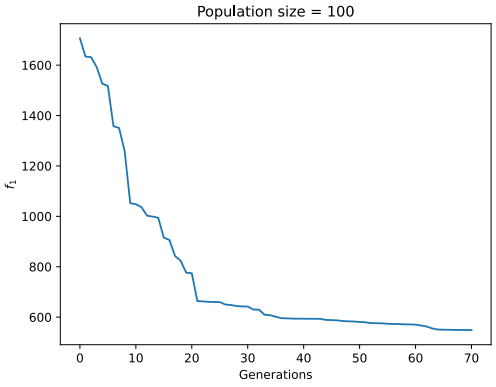
Obj	a_1	a_2	a_3	a_4	a_5	Score	Baseline/Score
f_1	$40.25 \cdot 10^{-2}$	$74.2 \cdot 10^{-3}$	$52.15 \cdot 10^{-5}$	$36.06 \cdot 10^{-3}$	$19.06 \cdot 10^{-3}$	484	1.01
f_2	$34.534 \cdot 10^{-2}$	$73.3 \cdot 10^{-3}$	$63.98 \cdot 10^{-4}$	$54.79 \cdot 10^{-5}$	$14.86 \cdot 10^{-3}$	9172	1.07

5.3 Weight Equation Two

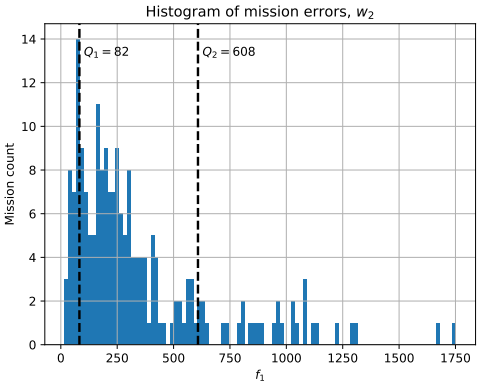
After analyzing the results for weight equation one, we explore the outcomes for weight equation two. Similar to the previous weight equations, it was multiplied by the distance of the road to obtain the weighted distance. The optimization process was performed using both objective functions f_1 and f_2, and the results for each objective function are presented separately. The optimization scores and parameter values for weight equation three are shown in table 5.3. Furthermore, the distribution of mission errors on the test data for functions f_1 and f_2 are shown as histograms in figures 5.4b and 5.5b.

5.3.1 Optimizing Results Using Objective Function f_1

The optimization focused on weight equation two and the objective function f_1. By utilizing the CPSO algorithm, we aimed to enhance the performance of this specific weight equation.



(a) Graph of the progression of the best particle score achieved through the CPSO algorithm over successive generations. The x-axis represents the number of generations, while the y-axis denotes the corresponding best particle score. The population size for this analysis is set to 100.

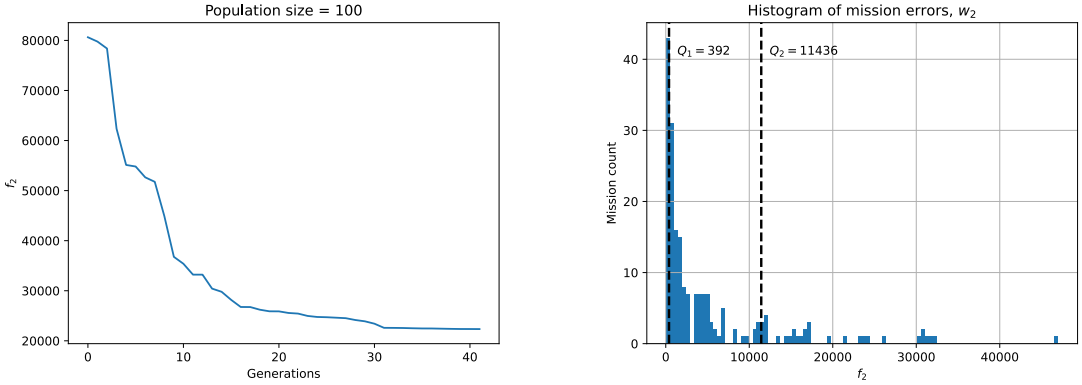


(b) The histogram presents the distribution of individual mission errors. These errors were evaluated using the best parameters identified within the training set, and the histogram specifically showcases the performance of these parameters evaluated on the test dataset. The y-axis is the mission count, and the x-axis is the error value.

Figure 5.4: The graphs show the best particle score from CPSO after several generations using the f_1 objective function and weight equation (4.2). Q_1 and Q_2 are quantiles same as before, $Q_1 < 15$ represents the good scores and Q_2 >85% represents the bad scores.

5.3.2 Optimizing Results Using Objective Function f_2

Similarly to the previous section the optimization results of objective function f_2 done on weight equation two are presented in figure 5.5 and table 5.3.



(a) Graph of the progression of the best particle score achieved through the CPSO algorithm over successive generations. The x-axis represents the number of generations, while the y-axis denotes the corresponding best particle score. The population size for this analysis is set to 100.

(b) The histogram presents the distribution of individual mission errors. These errors were evaluated using the best parameters identified within the training set, and the histogram specifically showcases the performance of these parameters evaluated on the test dataset. The y-axis is the mission count, and the x-axis is the error value.

Figure 5.5: The graphs show the best particle score from CPSO after several generations using the f_1 objective function and weight equation (4.2).

Table 5.3 summarizes the results obtained for the second weight equation. The findings are similar to those observed for weight equation one, as both the scores and distributions are near the baseline score. This suggests that the optimization process for weight equation two has not significantly deviated from the baseline performance.

Table 5.3: Table of results from the second weight equation (4.2) using both objective functions

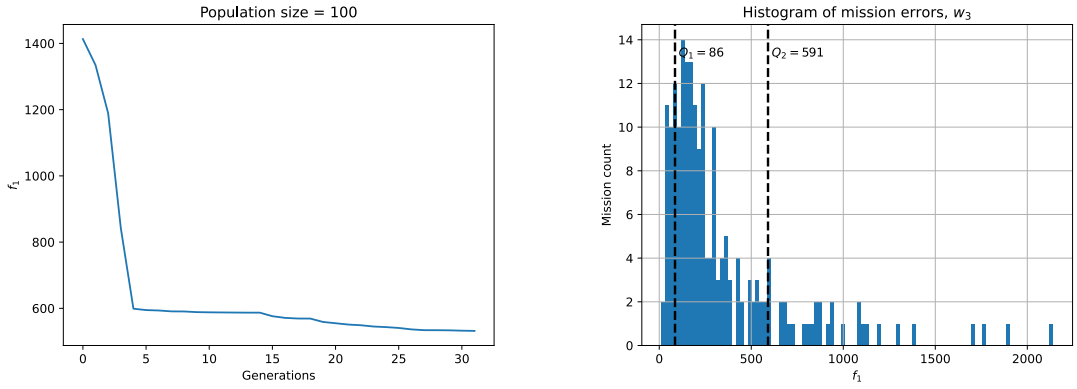
Obj	b_1	b_2	b_3	b_4	b_5	b_6	Score	Baseline/Score
f_1	$17.38 \cdot 10^{-2}$	$16.11 \cdot 10^{-4}$	$87.77 \cdot 10^{-5}$	$29.54 \cdot 10^{-1}$	$79.53 \cdot 10^{-2}$	$19.23 \cdot 10^{-6}$	469	1.04
f_2	$12.8 \cdot 10^{-1}$	$11.3 \cdot 10^{-1}$	$12.5 \cdot 10^{-3}$	$13.1 \cdot 10^{-2}$	$37.7 \cdot 10^{-4}$	$80.3 \cdot 10^{-19}$	9287	1.06

5.4 Weight Equation Three

Finally, weight equation three was used when optimizing f_1 and f_2 which yielded the results presented in figures 5.6, 5.7 and table 5.4.

5.4.1 Optimizing Results Using Objective Function f_1

When using f_1 as the objective function the following scores were obtained for the optimization on the training set see figure 5.6a and on the test set the score distribution shown in figure 5.6b was attained.



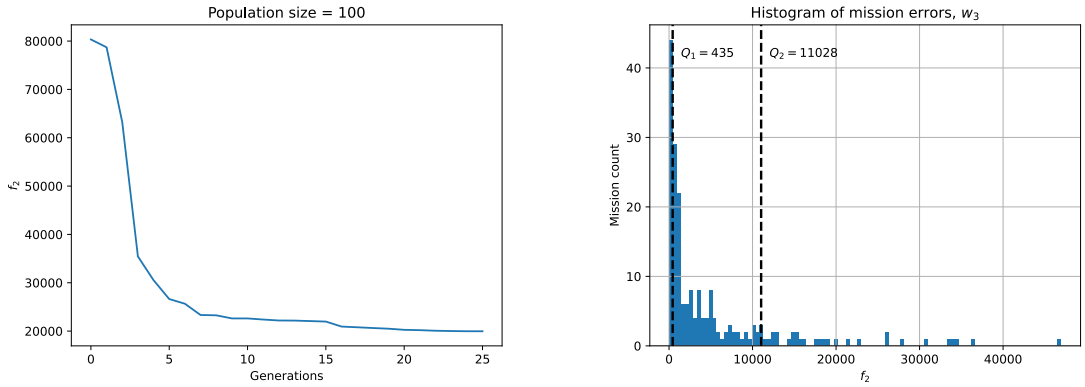
(a) Graph of the progression of the best particle score achieved through the CPSO algorithm over successive generations. The x-axis represents the number of generations, while the y-axis denotes the corresponding best particle score. The population size for this analysis is set to 100.

(b) The histogram presents the distribution of individual mission errors. These errors were evaluated using the best parameters identified within the training set, and the histogram specifically showcases the performance of these parameters evaluated on the test dataset. The y-axis is the mission count, and the x-axis is the error value.

Figure 5.6: The graphs show the best particle score from CPSO after several generations using the f_2 objective function and weight equation (4.2).

5.4.2 Optimizing Results Using Objective Function f_2

The same procedure as for f_1 with weight equation (4.3) was repeated using f_2 as the scoring function. The results from using the training data to optimize are shown in figure 5.7a and using the optimized parameters on the test data the score distribution in figure 5.7b was acquired.



(a) Graph of the progression of the best particle score achieved through the CPSO algorithm over successive generations. The x-axis represents the number of generations, while the y-axis denotes the corresponding best particle score. The population size for this analysis is set to 100.

(b) The histogram presents the distribution of individual mission errors. These errors were evaluated using the best parameters identified within the training set, and the histogram specifically showcases the performance of these parameters evaluated on the test dataset. The y-axis is the mission count, and the x-axis is the error value.

Figure 5.7: The graphs show the best particle score from CPSO after several generations using the f_2 objective function and weight equation (4.2).

After evaluating all the weight equations, it was found that number three obtained the lowest score.

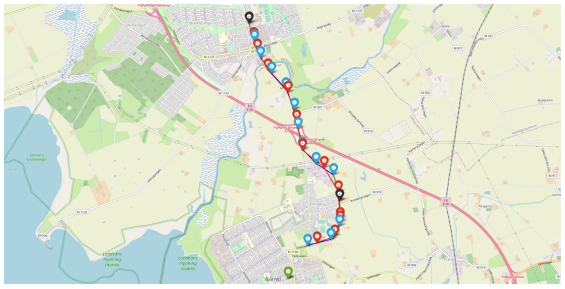
Table 5.4: Table of results from the second weight equation (4.3) using both objective functions

Obj	d_1	d_2	d_3	d_4	Score	Baseline/Score
f_1	$29.87 \cdot 10^{-1}$	$11.48 \cdot 10^{-1}$	$87.29 \cdot 10^{-4}$	$96.11 \cdot 10^{-3}$	486	1.01
f_2	$20.37 \cdot 10^{-2}$	$77.27 \cdot 10^{-3}$	$16.19 \cdot 10^{-2}$	$12.71 \cdot 10^{-1}$	9462	1.04

5.5 Examining Good and Bad Scores

The utilization of quantiles Q_1 and Q_2, as depicted in the previous histograms, aids in establishing the distinction between good and bad scores. Plotting these quantiles enables us to examine the characteristics that differentiate them. Below are examples of a typical mission with a good score and a typical mission with a bad score.

(a) A typical route coming within the Q_2 quantile. Here we see the first 40 predictions (blue markers) choose a different path than the actual route (red markers)



(b) A typical route coming within the Q_1 quantile. Here we see the first 40 predictions (blue markers) choose the same path as the actual route (red markers)

Figure 5.8: The following images visually represent two missions with error scores falling within the quantiles Q_1 and Q_2 as observed in Figure 5.2b. These predictions were generated using the weight values from Table 5.2 and objective function f_1. In the images, the black markers indicate the initial position of an EV, the green markers represent the destination, the red markers indicate the actual pinged positions of the EV, and the blue markers represent the predicted positions.

In the images above 5.8, we see a typical snapshot of missions in the Q_1 and Q_2 quantiles, respectively.

6
Discussion

This chapter provides some possible explanations for why the parameter optimization was unable to improve the previous algorithm with more than 4% and 7% for objective functions f_1 and f_2, respectively. It also provides more context to as why CPSO was selected and analyzes the running time of the developed optimization algorithm. Finally, it also discusses the possibilities of future improvements and limitations in the data used.

6.1 Optimizer Selection

There exists an abundance of optimization methods which each have its perks and drawbacks. Hence it becomes an important choice to pick the most suitable method for the problem. A good starting point is to look at the function which is to be optimized. In this thesis, it's the objective functions f_1 and f_2 described in section 4.4. The following arguments hold true for both f_1 and f_2 as such the objective functions will be referred to as f. Since the functions calculate the RMS value of the distance between each predicted position of the EVs and the actual positions for each mission and the fact that the A^* algorithm will always find a route for each mission as explained in section 3.3, this implies that for all parameter values $\vec{x}$, $f(\vec{x})$ will have some value. Furthermore, the route is calculated in order to predict the positions, and a small weight change may change the route selected for several missions, thus affecting the value of f by a substantial amount. In contrast, other large changes may not affect the routes and thus not f by as much. This implies that f will be nonlinear and the derivatives of f with respect to the parameters are impossible to calculate. This makes gradient-based methods ill-suited for optimizing f. The fact that some thresholds of the parameters will change the routes chosen for several missions also makes it so that f has several local minima since each parameter set that yields more correct routes than the parameter values in its neighborhood will form a local minimum. The parameter space can also be specified since all edge weights must be positive in the road network graph. Using this, and the knowledge that higher speed and size should increase the weight, it's reasonable to assume that all parameters should also be positive.

$$x_i \geq 0, \quad \forall x_i. \tag{6.1}$$

Where x_i are the parameter elements in $\vec{x}$. An upper bound can also be estimated by trying values and seeing where the bound of each parameter is by looking at when the weights are consistently too large. This is seen by the predicted positions

always being clustered at the start of the missions since the EVs are assumed to drive with a constant weight per second. Finally, due to the running time and the fact that each weight only depends on five discrete values for road size and speed, a low number of parameters were used when optimizing, making it so that all three criteria in list 3.5 are satisfied for the objective function hence a stochastic search method seems to be well suited for the optimization.

There are several different stochastic search methods e.g. GA, PSO, and CPSO. However, since PSO performs poorly in the presence of multiple local minima [18], it was not considered for our purpose. Instead, CPSO was chosen. However, GA might have worked well too. Due to the running time of each objective function evaluation, a random or exhaustive search through the parameter space wasn't a viable option.

6.2 Insufficient Data

The optimization results are severely limited by the road network data since there are only five discrete values for road size and speed, along with the distance of the road, to determine its weight. For this reason, many aspects of each road might be overlooked. To illustrate this, consider two roads of equal size, speed, and length, where one contains many curves and the other is straight. The first road should have a smaller weight than the second due to its geographical properties. Still, due to the architecture of the algorithm, these roads will be assigned equal weight regardless of which parameter values are used, meaning that it's impossible to correctly weigh the network through the optimization given such an instance. With more complex road network data, this algorithm could yield substantially better results at the cost of increasing time complexity.

Moreover, there is data available regarding the road names. There was an attempt to use this data by separating the roads in too four different categories; EU roads, larger national highways, rural roads, and others. Each road type got its parameter associated with them, representing its priority with a value in the interval $[0.5, 1]$, where the weight of the related road type was multiplied by the respective priority value. However, this did not yield satisfactory results, and the rural roads got higher priority than EU roads which shouldn't happen since EU roads are faster and broader. This could be due to the data set of missions having an over-representation of some road types compared to others which in turn means that giving them high priority could yield a good score from the objective function. This was not included in the result since only one optimization was made due to time constrains. This would be included if a comprehensive optimization was made including all weight equations and objective functions

Another area for improvement with the database for the road network and the ambulance position is that there needs to be height data available. This lack of this causes issues each time a mission is processed since the closest node in the road network graph, which is used as the start position, might be located on e.g. a bridge

above or below EV's current position, making the algorithm unable to predict correctly. These missions have been removed by hand in the pre-processing. As such, they are absent in the optimization, but when the algorithm is used in real-time, this may lead to incorrect position/route predictions. This problem is explained more thoroughly and illustrated in [1].

6.3 Running Time

Another critical aspect of the algorithm is how the road network graph is created, more specifically, how the road network is taken from the SQL database. It severely affects running time if too much of the road network is loaded each time. In the previous thesis [1], a constant value of 0.1 degrees was added to the start and destination of the mission to create a bounding box to determine how much of the road network was to be fetched from the database. As mentioned in section 4.4, changes were made to look at the pings from the mission to fetch a smaller portion of the road network. This not only ensures that enough of the road network is loaded to enable correct route prediction but also speeds up running time by adapting how much data is loaded.

The running time of the route calculator also posed a limiting factor to the optimizations since the CPSO optimization was limited by the population size of 100. While the population size and number of generations were capped to reduce the running time of the optimization, which was around 40-70 hours, using a higher cap the same optimization setup might have achieved better results.

The algorithm's time complexity depends on multiple factors. Firstly, the route and position prediction and the error are calculated once for every mission, which is done for every individual of each generation. This yields

$$\mathcal{O}(i, n, m, |E|, p) = (in) \cdot m \cdot (|E|^2 + p + p). \tag{6.2}$$

The in represents the running time of CPSO, which is the maximum number of iterations i times the population n, the m represents the number of missions, $|E|^2$ comes from the running time of going through the SQL road network database once which contains $|E|$ edges/roads and for each time this is done the $A*$ algorithm is applied which takes $|E|$ time. Finally, $p + p$ comes from predicting the number of pings p positions and calculating the error of each class for each mission for each individual.

Several design choices were made to speed up the running time of this algorithm. A critical feature of stochastic population algorithms is that each individual does not depend on any other individual's score from the same iteration. As such, when implementing the CPSO algorithm, the evaluation of individuals can be run in parallel. This is an essential feature of the optimization method that affected the algorithm loop design shown in figure 4.5. By utilizing parallelism, each iteration got cut in time complexity by a factor $\frac{1}{\#CPU}$, where $\#CPU$ is the number of available cores

on the system's CPU.

6.4 Results

The final score results presented in tables 5.2, 5.3, and 5.4 are all within 7% performance of the original parameters. After evaluating all the weight equations, it was found that weight equation three obtained the worst score. Initially, this equation was believed to yield better results due to its broader parameter exploration with fewer parameters. However, it performed worse, indicating the significance of the second-order terms that is included in weight equation one and two. Based on this information, the parameters used in the previous thesis [1] to compute weights using speed, size, and length data are nearly optimal. However, it may be possible to find better parameters by increasing the weight dependency on the geometry of the road, where the weight depends on both the curviness and length of the road. This could help improve performance since roads with severe curves are more challenging to traverse quickly.

One possible reason why the CPSO cannot make significant improvements to the baseline is that the mission data is based on actual ambulance data, which is subject to human error. In some cases, drivers may choose a less optimal route. This can result in a higher score for the objective function. Moreover, the missions where drivers select sub-optimal routes and the assumption that the EVs travel at a constant weight per time speed could explain why no parameter set managed to yield more than 7% improvement for any of the objective functions. It is also possible that the CPSO may miss some optima due to the population size of 100 which along with the number of parameters defines the resolution of the search space.

A false-positive rate for route selection would be required to further validate the parameters obtained through optimization, and those used in [1]. This, coupled with the error scores presented in the results, would indicate the probability of selecting the correct route and how well positions along that route are predicted.

7
Conclusion

The results from graphs 5.2, 5.3, 5.4 5.5, 5.6 and 5.7 indicates that the CPSO optimization finds some minimum to the objective functions which are significantly better than any of the initial values. Looking at all three weight equations used, weight equation two (4.2) had the most improvement of f_1 with 4% and weight equation one (4.1) had the largest improvement on f_2 with 7%. Unlike weight equation three (4.3) both (4.1) and (4.2) use second order coefficients for speed and size of the road which indicates that a quadratic model is slightly better than the first order dependencies used in the previous thesis [1]. However, when comparing the scores obtained for using f_1 and f_2 in tables 5.2, 5.3 and 5.4 to the baseline score in table 5.1, a key observation is that the optimized scores are as mentioned within 4% and 7% of the baseline scores for f_1 and f_2 respectively. This and the fact that all parameter sets from the optimizations are different and use different weight equations would suggest some inherent limitation in the model, which prevents the error score from going significantly below the baseline score. Such an assumption could be that the weight only depends on the road's speed, size, and distance. These results also provide a metric for validating the algorithm developed in thesis [1] since the score demonstrates how close the predicted positions are to the actual positions. Furthermore, since none of the tested weight algorithms could significantly improve, a more nuanced weight calculation that depends on more than the road's size, speed, and length is necessary to find a better minimum.

If Carmenta decides to try to further improve the algorithm, our suggestion is to focus on three main areas: acquiring new data with more speed and size categories for each road, revising the weight equation to account for additional aspects of the road and finally refining the model used to simulate how EVs traverse the road network. Rather than adjusting the parameters of the current weight equation.

Bibliography

[1] A. Arvidsson and J. Hendén, "Real-time route prediction of emergency vehicles," *Master's thesis in Mathematics Chalmers*, vol. 1, 2022.

[2] Z. H. L. S. Wang, X., "Path planning of scenic spots based on improved a* algorithm," *Sci Rep*, vol. 12, 2022.

[3] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: the case for a*," *International Journal of Geographical Information Science*, vol. 23, no. 4, pp. 531–543, 2009.

[4] R. Yuster, "Approximate shortest paths in weighted graphs," *Journal of Computer and System Sciences*, vol. 78, 2011.

[5] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1274–1285, 2019.

[6] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Transactions on Cybernetics*, vol. 45, no. 2, pp. 191–204, 2015.

[7] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised leaning," *International journal of computer science*, vol. 1, no. 2, pp. 111–117, 2006.

[8] G. G, *"Graph Theory"*. "Dover Publications", 2012.

[9] M. Jünger, G. Reinelt, and G. Rinaldi, "Chapter 4 the traveling salesman problem," in *Network Models*, vol. 7 of *Handbooks in Operations Research and Management Science*, pp. 225–330, Elsevier, 1995.

[10] C. Wang, L. Wang, J. Qin, Z. Wu, L. Duan, Z. Li, M. Cao, X. Ou, X. Su, W. Li, Z. Lu, M. Li, Y. Wang, J. Long, M. Huang, Y. Li, and Q. Wang, "Path planning of automated guided vehicles based on improved a-star algorithm," in *2015 IEEE International Conference on Information and Automation*, pp. 2071–2076, 2015.

[11] K. Khantanapoka and K. Chinnasarn, "Pathfinding of 2d 3d game real-time strategy with depth direction a algorithm for multi-layer," in *2009 Eighth International Symposium on Natural Language Processing*, pp. 184–188, 2009.

[12] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: the case for A*," Apr. 2009.

[13] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel, "Two decades of blackbox optimization applications," *EURO Journal on Computational Optimization*, vol. 9, p. 100011, 2021.

[14] J. Martins and A. Ning, *Engineering Design Optimization*. 10 2021.

[15] R. Wildman and A. Gaynor, "11 - topology optimization for robotics applications," in *Robotic Systems and Autonomous Platforms* (S. M. Walsh and M. S.

Strano, eds.), Woodhead Publishing in Materials, pp. 251–292, Woodhead Publishing, 2019.

[16] J. Leng, "6 - optimization techniques for structural design of cold-formed steel structures," in *Recent Trends in Cold-Formed Steel Construction* (C. Yu, ed.), pp. 129–151, Woodhead Publishing, 2016.

[17] R. Cheng, C. Sun, and Y. Jin, "A multi-swarm evolutionary framework based on a feedback mechanism," in *2013 IEEE Congress on Evolutionary Computation*, pp. 718–724, 2013.

[18] K. Luu, M. Noble, A. Gesret, N. Belayouni, and P. Roux, "A parallel competitive particle swarm optimization for non-linear first arrival traveltime tomography and uncertainty quantification," *Computers Geosciences*, vol. 113, 04 2018.

A

Appendix 1 - Code for route calculator

```
totWeightedDist = 0.0;
var backtrack = new Dictionary<int, int>();
var queue = new PriorityQueue<(Node, double),
                double>(new Comparator<double>());
var startDist = GetHeuristicDistance(startNode, endNode);
queue.Enqueue((startNode, 0.0), startDist);
var foundRoute = false;

while (queue.Count > 0)
{
    var (node, distToHere) = queue.Dequeue();
    if (node.id == endNode.id)
    {
        foundRoute = true;
        totWeightedDist = distToHere;
        break;
    }

    foreach (var (outNode, dist) in node.outNodes)
    {
        if (!backtrack.ContainsKey(outNode.id))
        {
            backtrack.Add(outNode.id, node.id);
            var distToEnd = GetHeuristicDistance(
                    outNode, endNode);
            var distToOutNode = distToHere + dist;
            queue.Enqueue((outNode, distToOutNode),
distToOutNode + distToEnd);
        }
    }
}

nodeIds = foundRoute ? Backtrack(backtrack, startNode.id,
endNode.id) : new List<int>();
return foundRoute;
```

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY