Optimal routing and charging procedures for electric buses

Kseniia Masliakova
Master Thesis, Computer Science
Department of Computer Science and Computational Engineering
UIT - The Arctic University of Norway,
Narvik,
Norway,

July 7, 2016

Master thesis project 2016
for
540659 Kseniia Masliakova

Optimal routing and charging procedures for electric buses

This project should be based on previous MSc work on Ant Colony Optimization (ACO) undertaken by Kristoffer Tangrand in 2015 and the present ACO program developed in Pyhton in the FlexChEV project. References should also be made to similar projects such as IVIsion of Fraunhofer and the Helsinki el-bus project supported, among others, by VTT. Data and specifications will be gathered from Troms fylkeskommune's el-bus project in 2016.

The goal of the project is to use Ant Colony Optimization techniques with Genetic Algorithm(s) to design and analyze routes and recharging schedules for electric buses. Two systems will be analyzed. One system will use "en route" charging where the bus applies a kind of pantograph at bus stops to recharge the bus batteries. This implies more frequent, but short recharging intervals. Another will address "depot recharging". This means that the buses will be recharged when they park at depots for the night. The latter will usually require larger batteries that again will require more space and impose greater weight loads for the bus. Simulations should take into account temperature, congestions, road conditions, traffic incidents and topography. Inclusion of topography is important as up-hill driving will require more power, especially during winter time when it can be very slippery. Down-hill driving opens-for quick charging by means of brakes and braking. Some buses have this opportunity.

Consequently, the student should:

1. Apply the present ACO program to investigate a selection of bus routes in Tromsø where real tests of different types of electric buses might be performed. The selection process should be made together with the transportation department at Troms Fylkeskommune (TF). Two routes should be selected and modeled for more detailed analysis. The aim is to make up-front analyses and prognoses of how different bus and recharging technologies could behave when serving the two bus routes selected by TF.

2. In face of the challenges confronted with in #1 above the student should recommend improvements to the present program and implement these. This should be done together with TF, the advisor and Mr. Tangrand.

3. Particular focus needs to be directed towards bus technologies using a kind of pantograph system. The student should create a GA that together with the ACO designs a recharging system along the two routes. The aim is to minimize

investments, costs of operation and time of travel for passengers. Other relevant constraints should be defined too if necessary. The system should thus define the number of charging stations for the pantograph, the distance between them and the stops required at each one for 365 days of operation under variable conditions. The GA will suggest alternative solutions and the ACO will rank them.

The student should familiarize herself with the problem and related literature and create a work plan for the project. The work plan should be due after three weeks. A final report should be prepared and delivered at the end of the project together with the program developed. The report should follow the standards applied to regular scientific articles describing the background for the project, state-of-the-art in research and transport, the definition of the problem undertaken, assumptions and limitations, the method applied to solve the problem, an extensive description of different results, a discussion of such results and a final conclusion. A proper reference list is required. Besides the report a system primer is required that could help users apply the program in a fruitful way.

Supervisor UIT - Navik: Professor Bernt A. Bremdal

Abstract

The goal of the project is to use Ant Colony Optimization techniques with Genetic Algorithm to design and analyse routes and recharging schedules for electric buses. One system so-called "en route" charging was analysed where the bus applies a kind of pantograph at bus stops to recharge the bus batteries. This implies more frequent, but short recharging intervals. The program offers different variants of placing for charging station with different characteristics and amount. Also there is possible to predict the best battery capacity for electric bus. The application was implemented for five different cases: if we don't want to place new bus stops, we use homogeneous charging stations and fixed battery capacity; if we don't place new bus stops, we use inhomogeneous charging stations and fixed battery capacity; if we want to place new bus stops, we use homogeneous charging stations and fixed battery capacity; if we want to place new bus stops, we use inhomogeneous charging stations and fixed battery capacity; if we don't want to place new bus stops, we use homogeneous charging stations and variable battery capacity. In the presented project we have the main focus on the cases where we don't place the new bus stops, because we analyse only one route. Also we focus on the case where we use variable battery capacity, in this case we can analyse if there is possibility to reduce the battery capacity for the bus.

Preface

I want to thank my supervisor Bernt A. Bremdal for providing useful insight and experience in researched field, and also for giving me opportunity to work in an interesting project. I want to thank Kristoffer Tangrand for consultations according Ant Colony Optimization algorithm and Thomas Martinsen for giving me the information about the problem area. I want to thank Liv Cecilie Evenstad from Troms fylkeskommune and Stig Normann from Troms fylkestrafikk for providing the necessary data to me. I also want to thank my family and friends for supporting me throughout this Master's Degree and Master Thesis.

Contents

Chapter 1

Introduction

The master's thesis work presents the machine learning approach including Ant Colony Optimization (ACO) techniques with Genetic Algorithm (GA) to solve the Vehicle Routing Problem. The goal of the project is to use ACO with GA to design and analyse routes and recharging schedules for electric buses. The task with corresponding data was given by Troms fylkeskommune. Their el-bus project was started in 2016. The problem is to find the optimal solution for placing charging stations along one bus route in Tromsø.

Electric vehicles have become very popular the latest time and there is many reasons why, the main is that it is ecological transport. In Norway electric transport is especially popular. Of course for comfortable using such type of transport we need to develop necessary infrastructure, like charging stations for electric vehicles. We need to explore possible placement of stations, and the important point is to reduce the cost of this process.

Electric vehicles are a very good solution for public transport, for example electric buses. The present project is focused on developing the infrastructure for charging process of elecric buses. Much attention was paid to analyse the topography, because of mountainous areas.

1.1 FlexChEV project

The Flexible Electric Vehicle Charging Infrastructure (FlexChEV) project is a project that has been granted funds by EU's SmartGrids ERA-NET program. The project funding encompasses UIT - campus Narvik, Aalborg University and University of Zagreb Faculty of Electrical Engineering and Computing [18]. The project is focusing on several areas of research, some of them [8]:

- To integrate the energy storage system in distribution grids as an imortant part

of smart charging station;

- To find optimal storage technology and capacity;
- To control the design of charging stations and allow widely use of the electric vehicles;
- To coordinate concept between the constructed flexible electric vehicle charging infrastructure.

The vision of the Flexible Electric Vehicle Charging Infrastructure project is to develop an integrated concept of smart charging stations.

The goal of the project is to develop a new concept for flexible electric vehicles charging stations. Emerging technologies and control methods deployed in distributed generation systems and microgrids are used to reach this goal. A new distributed coordination strategy between the grid and energy storage systems converters will be implemented. This strategy will provide expandability and robustness of the system.

The focus of the project is also on a new generation of fast hybrid electric vehicle charging stations, on theoretical development and experimental verification of such type of charging stations. We can say with confidence that flexible fast charging stations will be an important part of the future intelligent power systems [8].

1.2 Troms fylkeskommune's el-bus project

In our thesis we will focus more on another project. Troms fylkeskommune's el-bus project started in 2016. Nowdays, many countries (for example, Finland, Germany, Netherlands, Sweden...) start to use electric buses for public transport. And it is clear, the electric vehicle is ecologic transport, electric vehicles are much more energy efficient, maintenance cost, including fuel cost, is much lower with an el-bus [20].

Norway has probably the world's best incentives for Zero Emission Vehicles, and a correspondingly the world's highest number of electric cars per capita by a wide margin [6].

In 2016 the project connected with el-bus was started in Tromsø. In the beginning of the project it appears many important questions for analysing [7]:

- Battery range in Tromsø: topography, long winters, a lot of snow and demanding winter maintenance of roads, chain use, much cold and wet precipitation, and high requirements for the use of energy for air circulation in the bus.
- Recommended charging infrastructure: night charge, charging on bus stops or mix?

- How to prepare for changes in batteries in advance?
- Best routes: short, slightly flat, densely populated due to noise, possible location of any pantograph.
- etc ...[7]

This work will help to analyse charging infrastructure system for electric buses in Tromsø.

1.3 Previous Work

This project should be based on previous MSc work on Ant Colony Optimization (ACO) undertaken by Kristoffer Tangrand in 2015 and the present ACO program developed in Pyhton in the FlexChEV project [18]. Data and specifications will be gathered from Troms fylkeskommune's el-bus project in 2016. The goal of the presented MSc work of work Kristoffer Tangrand was to implement ACO approach for electric vehicles and simulate the traffic flow. The results and basic development of ACO approach with corresponding classes from this thesis are used in present work.

1.4 Contribution

The main goal is to improve presented implementation of ACO approach and use this approach in application with genetic algorithm. The GA approach was implemented in presented work, which with ACO helping to find optimal placing for charging stations. ACO is actually a part of GA. ACO ranks result, which the GA generates.

1.5 Plan of work

The first thing that we did was writing a detailed report with stepwize instructions and comments what should be done during the project. The plan consists of main research and implementation steps.

1. General preparation:

1.1 Read the literature with relevant material

1.1.1 Read general theory about Ant Colony Optimization algorithm.

The first question is to learn the general theory about algorithms: Genetic algorithm and Ant Colony Optimization algorithm. Ant Colony Optimization algorithm

applied in different spheres. I concentrated on the book of Marco Dorigo and Thomas Stützle: Ant Colony Optimization [4].

1.1.2 Read general theory about Genetic algorithm [9].

1.1.3 Read theory about how to apply these algorithms in optimal routing problem.

The big problem now is to figure out how to use GA in our routing problem, for example we don't know which fitness function will we use. Here we need to read literature about already solved routing problems and examples.

1.2 Familiarization with Python programming language (books, internet materials, internet courses).

2 Preparation according the given task:

2.1 Apply the present ACO program:

2.1.1 Familiarization with ACO program developed earlier by Kristoffer Tangrand: create a test network, test application with this network.

2.1.2 Write a specification for implementation, where describing the problem and what needs to be taken into account when it relates to routing of electric buses.

2.1.3 Find the theoretical ground for analyzing the routes according to the data in account (formulas).

2.1.4 Collect the data about routes, types of electric buses (get data from Troms fylkeskommune and Troms fylkestrafikk).

2.1.5 Apply ACO program for two types of electric buses in two selected routes (was selected only one type of buses - pantograph system and one route was analysed).

2.2 Make up-front analysis and prognoses of how different bus and recharging technologies could behave when serving the bus routes selected by Troms fylkeskommune.

2.3 In face of the challenges confronted with in 2.1 and 2.2 above, recommend improvements to the present program and implement these. This should be done together with Troms fylkeskommune, the advisor and Kristoffer Tangrand.

2.3.1 Recommend improvements to the present application according to the new case with electric buses and discuss these improvements with other participants.

2.3.2 Implement the improvements.

2.4 Implementation of new solution of the problem using Genetic Algorithm together with Ant Colony Optimization algorithm. The GA will suggest alternative solutions and the ACO will rank them.

2.4.1 Implement Genetic algorithm in general case.

2.4.2 Apply GA for routing problem, create the application on Python.

2.4.3 Connect GA and ACO together in one solution.

Here I want to answer the question how to use both algorithms together, explain and analyse results, describe advantages (or disadvantages) of method.

3 Write final report:

3.1 Describe all obtained results.

3.2 Write conclusion.

3.3 Collect all literature which has been used.

3.4 Prepare CD, attachments and other extra materials.

Chapter 2

Preliminaries

2.1 Vehicle Routing Problem

The target of vehicle routing problem is to minimize the total cost of the routes of the vehicles, serving a number of clients. The problem can be presented as a complete weighted graph $G = (V, A, d)$, where the vertices are represented as $V = (v_0, v_1, ..., v_n)$, and the arc as $A = \{(v_i, v_j), v_i! = v_j\}$. The central transport base, which is the beginning and end of each route, represented by the vertice v_0, the other vertices represent N clients. For arc (i, j) is given the destination d_{ij}. For the customer i set to positive demand q_i, capacity is limited by number Q of each vehicle. Also, the following constraints are specified for the task [3][11]:

- each customer must be visited only once;
- the beginning and end of all routes of the vehicles are in the transport base v_0;
- total demand served by each vehicle should not exceed Q.

A special case of VRP problems, when the demand of each customer is 0, is the traveling salesman problem. The task of routing is to transport combinatorial discrete optimization problem in which the number of permissible routes increases exponentially with increasing number of customers, and belongs to the class of complexity NP. For such problems, the use of heuristic algorithms seems reasonable, for example to use the algorithm of ant colonies [11].

2.2 Ant Colony Optimization algorithm

2.2.1 General description of ACO

Ant colony algorithm is part of a large class of group intelligence algorithms, in which for solving the problem it is used behaviors of social insects such as bees and ants.

Ant colony algorithm is based on ants ability to find the shortest path to a food source, and to adapt to changing conditions. Ants communicate with each other via the external secretions - pheromones, a fixed amount which they leave in their path. An ant, who has found a shortest route, will upgrade more intensely the pheromone trail. By choosing the direction of motion ants consider intensity of the pheromone trail, thus they will choose the shorter path. The evaporation of the pheromone and the element of randomness in choosing the path, allow to avoid the local optima and find shorter routes [11].

2.2.2 Applying ACO for Vehicle Routing Problem

When we solve the problem of transport routing ants imitate vehicles, routes are being built step by step by selection of the next client until you have served all customers. In the beginning, the ant starts on the base and list of customers, included in his route, is empty. The ant chooses the next customer from the list of clients and updates current load before selecting the next one. The ant returns to the depot when reached maximum load capacity, or all customers already are visited. The total length L_k is calculated as the value of the object function of full route of k-th ant. Ant colony algorithm builds a full route for the first ant before the second begins the movement. It continues as long as a certain number of ants m will build a full route [11].

Selection of the next customer is done randomly, based on a probability formula:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \eta_{il}^\beta}$$

if $j \in N_i^k$, otherwise 0.

τ- the amount of pheromone on the path between the current position i and possible position j;

$\eta_{ij} = \frac{1}{d_{ij}}$- reverse value for the destination between positions i and j;

the parameters α and β, respectively, represent the relative importance of the number of pheromone and distance when selecting the next customer.

When $\alpha = 0$ the algorithm degenerates into a greedy, when you select the closest city, excluding the amount of pheromone.

N_i^k- all possible vertices to which we can go from vertex i (neighbouring nodes) [18].

In order to achieve the improvement of future routes it is necessary to update the pheromone trail, depending on the quality of the solutions. This process is a key element of adaptive technology training algorithm of ant colonies, and helps to improve the subsequent decisions. Updating the pheromone trail consists of two

stages: simulating pheromone evaporation, and the next update, depending on the quality of the solutions. Evaporation of pheromone produced is represented by the following formula:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \sigma$$

ρ- parameter controlling the evaporation rate of pheromone;

σ- setting guarantees a minimum concentration of pheromone on the edge, which is arbitrary, sufficiently small value.

For arc (i, j) and route T_k pheromone is defined like this:

$$\triangle\tau_{ij,k} = \begin{cases} \frac{O}{L_k}, & (i,j) \in T_k \\ 0, & (i,j) \notin T_k \end{cases}$$

where O - value of optimal length of all routes, for the edge (i, j) the total amount of pheromone is determined according to the following formula:

$$\triangle\tau_{ij} = \sum_{k=0}^{m} \triangle\tau_{ij,k}$$

where m - the number of ants in the colony.

Thus, the sum of the two steps of the amount of pheromone given by the formula:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \sigma + \triangle\tau_{ij}$$

2.3 Genetic Algorithm

2.3.1 General description of GA

The genetic algorithm (GA) is an optimization and search approach, which is based on the rules of genetics and natural selection. During the work, the genetic algorithm generates many populations with big amount of individuals, these populations become more "fit" for the present problem after using selection rules, i.e. we will minimize the cost function step by step in every new population. This method was developed by John Holland over the course of the 1960s and 1970s, and finally used and popularized by one of his students, David Goldberg. David solved a difficult problem connected with the control of gas-pipeline transmission for his dissertation [9].

The main steps of the algorithm are presented on the Fig 2.1 [9]:

- Selection of the variables (chromosome), variable encoding and decoding

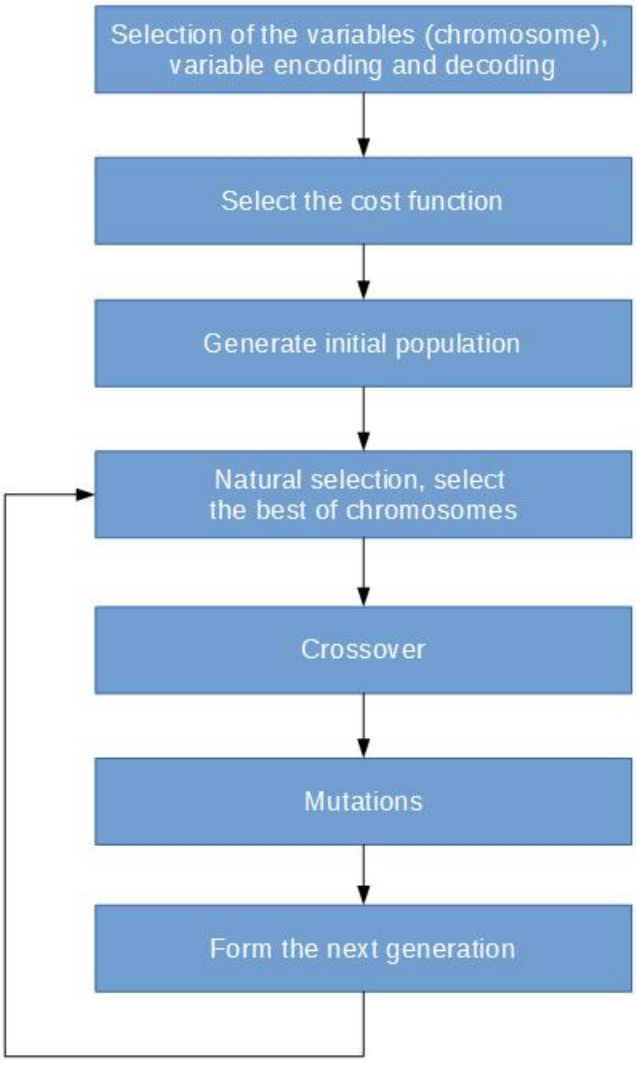

Figure 2.1: Steps of the genetic algorithm.

Genetic algorithm starts by defining a chromosome or an array of variable values which we need to optimize. If the chromosome has n variables given by $a_1, a_2, ..., a_n$, then the chromosome is written as an n element row vector. We can use binary representation of the chromosome:

$$chromosome = [a_1, a_2, ..., a_n]$$

$$chromosome = [1, 0, 0, 1, 1, 1, 0]$$

The variable values are represented in binary, so there must be a way of converting continuous values into binary, and visa versa. Sometimes it is difficult to represent the present data into binary. If we can not represent the data in binary, we can use continuous values, but for convergence of algorithm we must have enough initial material, a big amount of initial data for producing offsprings that will pass the fittest.

- Select the cost function

The cost function generates an output from a set of input chromosomes. The cost function can be a mathematical function or an experiment. We use the cost function to estimate the fitness of our set of chromosomes.

- The Population

The population is a group of chromosomes, for example the population has p chromosomes.

- Natural selection

Now we need to select the best chromosomes in our population to reproduce new offsprings. The fittest discards the chromosomes with the highest cost. First, the p costs and associated chromosomes are ranked from lowest cost to highest cost. Then, only the best are selected to continue, while the rest are deleted.

- Selection and Crossover

Two chromosomes are selected from k chromosomes to produce two new offspring through the crossover operation. The crossover is the creation of one or more offspring from the parents. The most common case of crossover is when two parents produce two offspring. A crossover point should be randomly (in some problems can be not randomly) selecteded between the first and last bits of the parents' set of chromosomes.

- Mutations

Random mutations change a certain amount of the bits in the list of chromosomes. *"A single point mutation changes a 1 to a 0, and visa versa."* [9]

- The next generation

After the mutations, we calculate the costs related to the offspring and mutated chromosomes. This process is iterated. The results after crossover and mutation will be ranking and will be formed new generation. For example, the bottom m chromosomes are discarded and replaced by offspring from the top m parents.

- Convergence

After several iterations all the chromosomes and related costs will become the same if it were not for mutations. At this stage the algorithm should be stopped.

Also we want to say that genetic algorithms are different from the majority of ordinary optimization and search methods in 4 points [14]:

- GA operates with coded set of parameters, and not by parameters themselves;
- GA finds a population of points, not a single point;
- GA uses the value of the object (target) function and not a derivative or other support values;
- In the GA it is used probabilistic transition rule, not deterministic.

In our problem we need to apply GA for the Vehicle Routing Problem.

If we will refer the present problem to some type of optimization, we can see that our problem belongs to following categories [9]:

- multidimensional - the problem requires several variables
- dynamic - the solution depends on the time
- discrete - the values can have only finite number of possible values
- constrained - we have some constrains to variables, for example positions of charging stations on the route

2.3.2 Applying GA for Vehicle Routing Problem

The main problem for applying GA for Vehicle Routing Problem was how to implement the crossover operator. The mutation operator was more clear for implementation.

We can look at the example of the crossover proposed in [13] and [21]: the **Best cost route crossover**. Best cost route crossover creates two offsprings from two parents. To explain the method let's consider that we have to routes A and B. First we choose two subroutes from each route: a and b. Then we remove the subroute a from route B and subroute b from route A. The next step will be to locate deleted subroutes to the best possible locations. So we will locate the subroute a in route B and subroute b in route A. The best insertion location is the location which give us total minimum cost routes.

2.4 Hamiltonian paths

The next theory we can consider is the theory about Hamiltonian paths [1], because we need to implement the algorithm for building a route through all nodes and visit each node once. Other words we need to solve "Travelling salesman problem" (Fig. 2.2).

When we will find all routes through all nodes we can predict optimal conditions for routes, for example predict positions of extra charging stations using genetic algorithm.

For temperature, road conditions and topography we can add some parameters of the arc and then use these coefficients in the algorithm.

But now we will just implement the algorithm for finding the route through all nodes.

In a theoretical way we can formulate the problem like this: we need to find the Hamiltonian paths (or cycles, depends on route) in the graph (route). Hamiltonian path is a path in an undirected or directed graph which visits each vertex exactly once, so that is what we need in our problem [1].

Hamiltonian cycle (path) is called a simple cycle (path) that contains all the vertices of the graph. In the graph shown in Fig. 2.3 the left Hamiltonian cycle is, for example, the sequence $1, 2, 5, 3, 4, 1$. The graph shown in the center, no Hamiltonian cycles, but there is Hamiltonian path, for example $2, 1, 5, 3, 4$. In the right column there are no Hamiltonian paths.

For the Hamiltonian cycle (and for paths) there is no simply verifiable, necessary and sufficient conditions for their existence. All known algorithms require to look over a large number of variants when you apply algorithm for a big graphs.

A Hamiltonian cycle is, with a combinatorial point of view, a permutation of

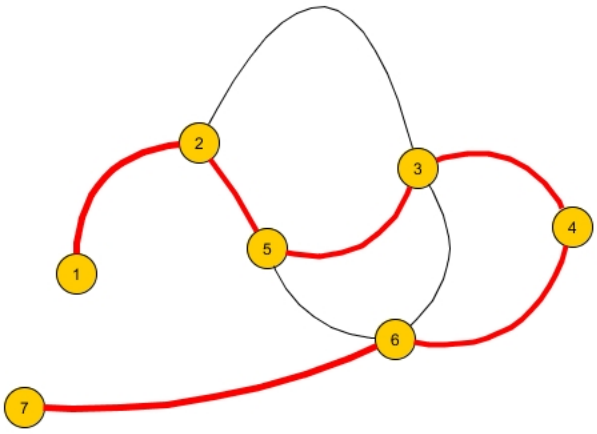

Figure 2.2: Travelling salesman problem

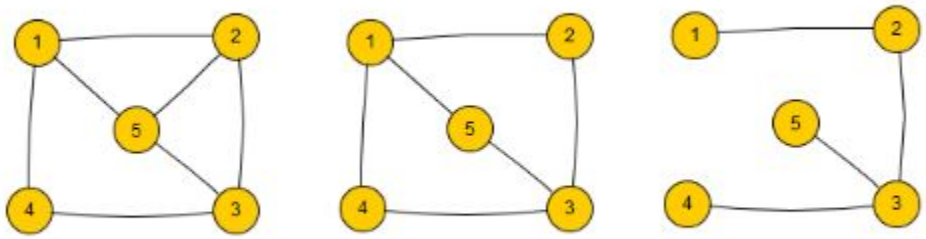

Figure 2.3: Hamiltonian paths

vertices. At the same time you can select any node as the starting top of the cycle. The most easy Hamiltonian cycle research plan is a consistent consideration of all these permutations and verification for each of them, if it is a cycle in the graph. This method is already impracticable at not a very large number of vertices because of the fast growth of the number of permutations - there is $(n-1)!$ permutations of n elements if the first member is fixed.

A more rational approach is to consider all possible ways starting at an arbitrary starting vertex a, as long as a Hamiltonian cycle is detected or all possible paths have been explored. In fact, we are also talking about iterating through the permutations, but reduced - if, for example, the vertex b is not adjacent to a vertex a, then all the $(n-2)!$ permutations in which the first place is a, and b in the second, are not considered.

Consider this algorithm in detail. We assume that the graph is given like neighborhoods of vertices: for each vertex x it is given a set of vertices adjacent to x. At each step of the algorithm there is already constructed part of the route, it is stored in the $PATH$ stack. For each vertex x, included in the $PATH$, it is stored a set $N(x)$ of all vertices adjacent to x, which was not yet seen as a possible way out from the vertex x. When the vertex x is added to the path, a set $N(x)$ is assumed equal to $V(x)$. In further consideration all the visited vertices are removed from the set. The next step is to explore the adjacent vertices of the last vertex x of the path $PATH$. If $N(x) \neq \oslash$ and $N(x)$ have vertices that do not belong to our route, one such vertex is added to the route. Otherwise vertex x is excluded from the stack. When, after adding to the path the next vertex, the path contains all the vertices of the graph, it remains to verify whether the first and the last vertex of the path are neighbours, and if this is true, than we got the Hamiltonian cycle [1].

Now we can describe the algorithm using pseudo programming language [1]:

```
Select randomly vertex a
a ⟹ PATH
N(a): = V(a)
while PATH ≠ ⊘ do
    x : = top(PATH)
if N (x) ≠ ⊘
    then take y ∈ N(x)
N(x): = N(x) - y
if the vertex y is not in the PATH
    then y ⟹ PATH
```

```
N(y): = V(y)
if PATH contains all the vertices
    then if y is adjacent to a
       then show the cycle
else remove the vertex x of PATH
```

Now we can try to implement Hamiltonian path (and cycle) using Python. There are two cases of routes: in first case the starting station and the final station are the same, in the second case they are different. So in the first case we need to find just the Hamiltonian path, in the second case we need to implement the Hamiltonian cycle.

The first case is more simple. We can just find all paths in the graph and check if the length of the path is equal to the amount of nodes. But of course this method is not optimal and will work slowly when we will have a graph with high amount of nodes. But if we have not so complex graph we can apply this algorithm.

```
def hamilton_path(start, path = []):
    path = path + [start]
    if start == goal:
       return [path]
    if not graph.has_key(start):
       return []
    paths = []
    adjacent = graph[start].keys() # all adjacent nodes to our node
    for x in adjacent:
       if x not in path:
          newpaths = hamilton_path(x, path)
          for newpath in newpaths:
            if len(newpath) == len(graph):
              paths.append(newpath)
return paths
```

For example we have such graph like on Fig. 2.4.

We will get the following result of the function.

$$[1, 2, 3, 4, 5, 6, 7]$$

$$[1, 2, 4, 5, 6, 3, 7]$$

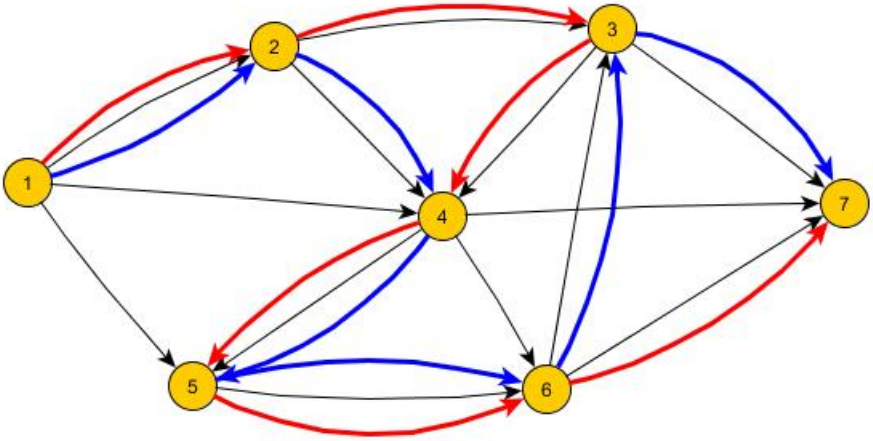

Figure 2.4: Hamiltonian path.

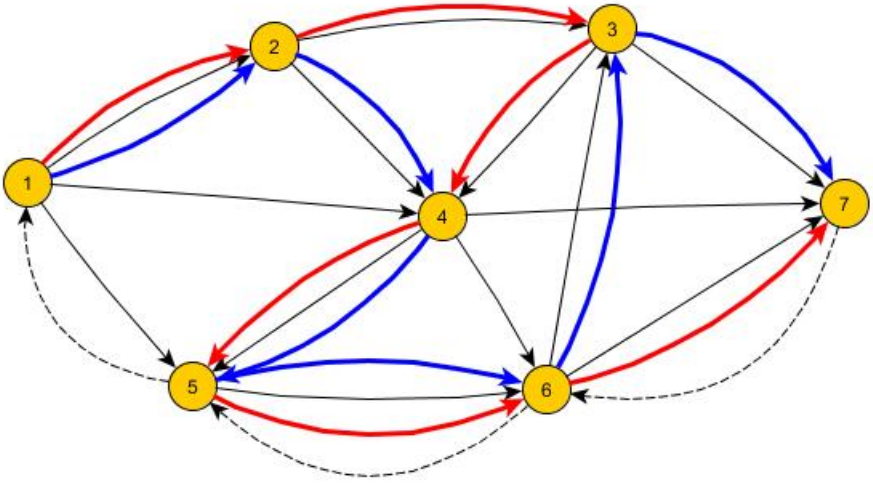

Figure 2.5: Hamiltonian cycle.

Next step is to implement the Hamilton cycle. For example we add three extra edges to our graph, we will get the following graph (Fig. 2.5).

We will have here only one cycle starting on node 1.

$$[1, 2, 3, 4, 7, 6, 5, 1]$$

The presented theory about Hamiltonian paths is useful if we want to change the bus route (sequence of bus stops), if we want to advice better and more optimal solution for placing bus stops. For example, we have many routes, which have general bus stops. So we can change the sequence of stops (by that we change sequence of charging stations) for several routes, and by that we can optimize the cost.

We will not use presented algorithm for Hamiltonian paths in our program, because we will start to work and analyse only one route. For one route, changing placement of bus stops can influence cost, but we will use a route in the city where bus stops can be placed on perpendicular streets, so we are not so free to change sequence of bus stations. It will be useful for future work.

Chapter 3

Specification for application

The first step is to write a kind of specification for future program. The main goal here is to familiarize with presented previous work of Kristoffer Tangrand [18], read more relevant theory and create a plan for work. Presented specification summarize results from previous work, which are useful for solving our problem. The goal of the specification is not to describe the precise solution, the main goal is to start to analyse which data we need to use, if we can use the previous results of Kristoffer's work and how we can use it, if we can use these results, how to apply and improve presented application for our case. The specification consists of a general overview of possible ways of solution.

The work was started with familiarization with Kristoffer's application for a single vehicle case with homogeneous nodes. Several networks were created and we can see one of the output results on Fig.3.1. The ant tries to find the shortest path through the graph, he has some energy amount, he try to recharge on the charging stations (nodes) just enough to reach the next node. Because this is a homogeneous case, every node will have the same class of charging stations, it means the time of recharging on stations will be the same (the output will be the same). Also every arc has cost of travel (this is the time of travelling) and weight (this is the pheromone trail from previous ants) [18][19].

In the case with inhomogeneous nodes, the class of charging stations will be different, so we will have different time of charging and the ant will also need to choose between the charging stations. The solution in present application is to charge on maximum level if we don't know the next class of charging station. After some iterations the agent which is travelling on the graph can already have access to information regarding the charge output for the current node and the next node on the path.

We can present a table of parameters for the present program (single vehicle case) (Table 3.1).

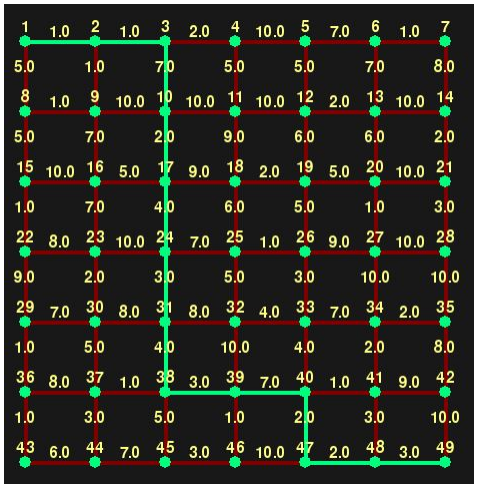

Figure 3.1: Single vehicle test with homogeneous nodes.

Table 3.1: Table with parameters for presented application.

Part of the graph	Parameters
Nodes	cost of charging (time of charging)
	output (case with inhomogeneous nodes)
Arcs	treveling cost (time and distance)
	weight (pheromone trail)
Ants	energy capacity
	energy amount

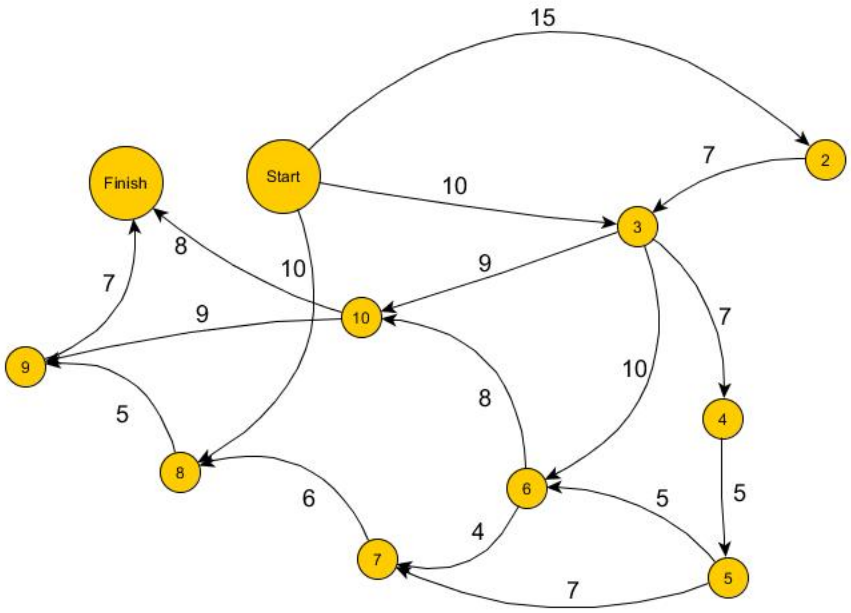

Figure 3.2: Example of possible network

Account of present program for single vehicle case contains such information: cost of travel and pheromone trail on the arc, cost of charging (different for inhomogeneous case) for nodes, energy of ants (cars).

What we considered next was a multi - vehicle case (deterministic case). The main difference from single vehicle case is that here we also consider traffic congestions. We can create the test graph of network to try to use the present application with it. On Fig. 3.2 we can see the directed graph with eleven nodes.

On Fig. 3.3 we can see the matrix of weights of this graph.

On Fig. 3.4 we can see the result of applying multi-vehicle case (deterministic case) for the graph of network on Fig. 3.2.

If we will analyse the data we can see that there are two shortest paths:

$$1 \to 3 \to 10 \to 11$$

$$1 \to 8 \to 9 \to 11$$

Now we can try to describe the case for routes of electric buses. For the case of electric busses we will need some extra information to take in account.

We need to describe some extra conditions:

0	15	10	0	0	0	0	10	0	0	0
15	0	7	0	0	0	0	0	0	0	15
10	7	0	7	0	10	0	0	0	9	0
0	0	7	0	5	0	0	0	0	0	0
0	0	0	5	0	5	7	0	0	0	0
0	0	10	0	5	0	4	0	0	8	0
0	0	0	0	7	4	0	0	6	0	0
10	0	0	0	0	0	6	0	5	0	0
0	0	0	0	0	0	0	5	0	9	7
0	0	9	0	0	8	0	0	9	0	0
0	15	0	0	0	0	0	0	7	0	0

Figure 3.3: Matrix for network

```
After iteration 10000:
1:  0.02->8  0.00->2  0.02->3
2:  0.00->3
3:  0.01->10  0.00->4  0.01->6
4:  0.00->5
5:  0.00->6  0.00->7
6:  0.01->10  0.00->7
7:  0.01->8
8:  0.03->9
9:  0.03->11
10:  0.00->9  0.02->11
11:
1 -> 8 -> 9 -> 11 = cost: 198.5 flow: 0.45242
1 -> 2 -> 3 -> 10 -> 9 -> 11 = cost: 293.8 flow: 0.00000
1 -> 2 -> 3 -> 10 -> 11 = cost: 237.3 flow: 0.00000
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 10 -> 9 -> 11 = cost: 312.5 flow: 0.00000
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 10 -> 11 = cost: 256.0 flow: 0.00000
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 11 = cost: 316.6 flow: 0.00000
1 -> 2 -> 3 -> 4 -> 5 -> 7 -> 8 -> 9 -> 11 = cost: 307.4 flow: 0.00000
1 -> 2 -> 3 -> 6 -> 10 -> 9 -> 11 = cost: 322.2 flow: 0.00000
1 -> 2 -> 3 -> 6 -> 10 -> 11 = cost: 265.7 flow: 0.00000
1 -> 2 -> 3 -> 6 -> 7 -> 8 -> 9 -> 11 = cost: 326.4 flow: 0.00000
1 -> 3 -> 10 -> 9 -> 11 = cost: 255.0 flow: 0.00000
1 -> 3 -> 10 -> 11 = cost: 198.5 flow: 0.23229
1 -> 3 -> 4 -> 5 -> 6 -> 10 -> 9 -> 11 = cost: 275.2 flow: 0.00000
1 -> 3 -> 4 -> 5 -> 6 -> 10 -> 11 = cost: 219.2 flow: 0.02152
1 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 11 = cost: 279.5 flow: 0.01779
1 -> 3 -> 4 -> 5 -> 7 -> 8 -> 9 -> 11 = cost: 270.6 flow: 0.06751
1 -> 3 -> 6 -> 10 -> 9 -> 11 = cost: 286.2 flow: 0.00000
1 -> 3 -> 6 -> 10 -> 11 = cost: 229.6 flow: 0.16203
1 -> 3 -> 6 -> 7 -> 8 -> 9 -> 11 = cost: 292.2 flow: 0.13776
```

Figure 3.4: Output of application with network from Figure 3.2.

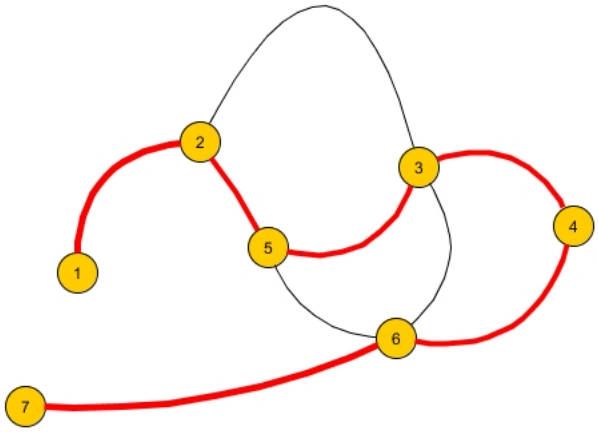

Figure 3.5: Travelling salesman problem.

- visit all bus stations

First of all we can say that in case of buses we will need to solve "Travelling salesman problem". The difference between present program and the case with buses is that now we need to visit all nodes (bus stations), one time each, and use the shortest path. So the ants must go through every node and reach the goal. After several iterations the shortest path should be selected.

- two types of buses (two types of charge)

In Table 3.2 we can see difference between two types of buses, which have different types of recharging method. The first type will recharge on every bus stops during the route. It means the bus need to have enough energy to reach the next bus stop and charge just enough on the recharge station. Another type of bus will charge just one time during the day, it can stay and recharge all night before the work day. This type of bus need to have very big battery capacity to work the whole day without recharging.

- influence of the outdoor temperature on energy consumption

It is known that the energy consumption is different at different outdoor temperature. So in the winter the car will consume more energy than in the summer. In Table 3.3 we can see the approximate difference between the energy consumption in different seasons [22]:

- traffic congestions and traffic incidents influence on the time in the route and energy consumption

Table 3.2: Two types of electric buses.

Type of bus	Parameters
"en route" charging	The bus applies a kind of pantograph at bus stops to recharge the bus batteries. This implies more frequent, but short recharging intervals.
"depot recharging"	This means that the buses will be recharged when they park at depots for the night. The latter will usually require larger batteries that again will require more space and impose greater weight loads for the bus.

Table 3.3: Energy consumption of electric buses in different seasons.

Season	Energy consumtion
Summer	approximately 300 - 315 kWh/h
Winter	approximately 350 - 365 kWh/h

In the present program of Kristoffer Tangrand these conditions are presented by using the following formula [10]:

$$T_{ij}^{k}(x_{ij}) = \frac{d_{ij} * x_{ij}^{k}}{v_f(1 - (\frac{k_i}{k_{jam}})^l)^m}$$

This is the travel time on a single arc for an ant k. The pheromone deposition is dependent on the travel time which is a combination of distance and congestion [10].

- road conditions: ice, bumpy road

We can present an example of how road conditions can influence on energy consumption.

For example, if we will drive a car on a flat road 50 km from point A to point B we will need x kWh [5].

Of course, we will need more energy to drive the same car from point A to point B if we travel on a bumpy road. The amount of energy is calculated by the distance traveled over the friction, which is the friction factor (F), to compensate for the bumpiness of the road, times the distance to travel divided by the kilometers per kWh the car gets on flat surfaces ($D = \frac{km}{KMperkW}$,where km - kilometers traveled and $KMperkW$ - kilometers per kWh), resulting in the following formula, where E - is amount of used energy [5]:

$$E = F * D$$

- topography

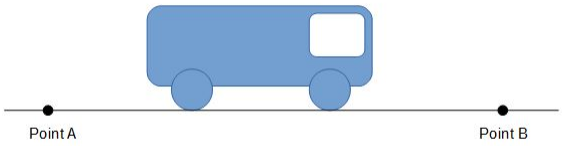

Figure 3.6: Energy expended traveling on a flat road is a function of distance.

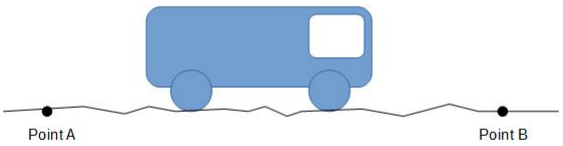

Figure 3.7: Extra energy is expended traveling on a bumpy road.

Let's calculate the amount of energy if the route from point A to point B is uphill. For now we can ignore the fact that additional energy would be necessary to move the car uphill. If we will contunue the previous example, we can see that the car now must travel on the surface for a longer distance. When considering surface distance (SD replaces D), the following formula is for used energy [5]:

$$E = F * SD$$

Another factor that can influence to the energy consumption of the car is the uphill or downhill slope, we can call this factor the vertical factor. We can see if the car is going downhill, the total cost of travel will decrease, because the car can have possibility to charge; if it is going uphill, the total cost will increase. If we will add the vertical factor (VF) into the previous formula we will get the following formula [5]:

$$E = F * SD * VF$$

Also down-hill driving opens opportunity for quick charging by means of brakes and braking. Some buses have this opportunity.

Also we can describe the dependence of energy consumption on the topography

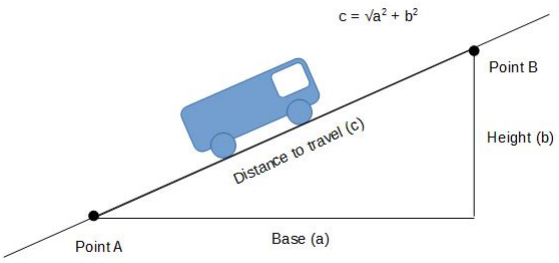

Figure 3.8: Extra energy is expended when going uphill.

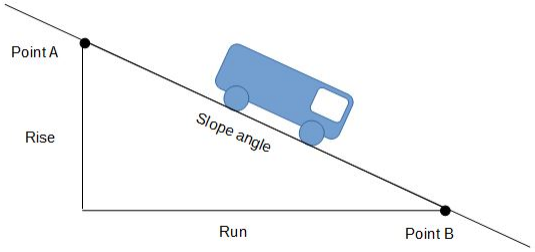

Figure 3.9: Less energy is expended going downhill, in this case the slope is negative.

Table 3.4: Table with parameters in case of electric buses.

Part of the graph	Parameters
Nodes	cost of charging
	output - inhomogeneous nodes
	energy capacity of charging station
Arcs	treveling cost (time)
	treveling cost (distance)
	weight (pheromone trail)
	capacity of arc
Ants (bus)	energy capacity
	energy amount
	type of bus
Another	temperature
	congestions
	road conditions
	traffic incidents
	topography

in another way. If we need amount of energy E to travel a distance D , and the specific energy consumption required to travel from A to B is R, so we have:

$$E = R * D + X$$

X here includes the increased energy consumption from rolling resistance and topography, we can describe X in following way:

$$X = X_{roll} + X_{up} + X_{sd} + X_{down}$$

where X_{roll} - the increased energy consumption from rolling resistance and $X_{up} + X_{sd} + X_{down}$ - the increased energy consumption from topography.

So now we can summarize all these factors and decide what we can take in account in the case with electric busses.

In Table 3.4 we can see parameters which we can take in account for the case of electric buses:

After writing the specification we can start to implement necessary algorithms.

Chapter 4

Genetic algorithm implementation

One of the main research goals of the project is to develop the Genetic algorithm approach. The important questions here are: choose the data structure for chromosomes and choose fitness function for ranking individuals in the population. The structure selection is very important, because we need to choose such chromosomes with which we can produce more various populations, so the algorithm can converge. We will describe the data structure for chromosome in a separate section. We need to remember that *"the genetic operators must always generate legal solutions"* [15], so it means we need to check everytime if we have some undesirable cases of positions of charging stations in our populations, for example in cases when the bus will have not enough energy to reach a depot.

Like we said earlier, the role of so-called fitness function in our application will play ACO approach. The cost function in our ACO algorithm will consist of two parameters, which we want to minimize: this is cost of charging stations and this is battery consumption for electric bus. This cost function will be our fitness function in GA algorithm. We will describe the fitness function later in a separate section.

First we can try to model some example network and try to apply for this experimental network GA approach. In the next section we will describe the modelled network.

4.1 The experimental network

On Fig. 4.1 we can see example of the route with charging stations. The red dots represent proposed charging stations. The GA suggests charging stations. The ACO analyse how beneficial the various configurations of charging stations are for the traffic system and rank solutions produced by the GA.

First we need to develop the genetic algorithm which can offer to us different positions of charging stations.

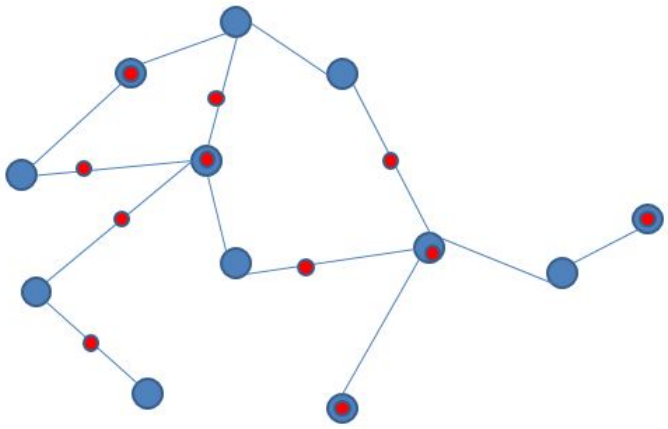

Figure 4.1: Test route with charging stations.

4.2 First try

As we said earlier the important question in the beginning is which data structure to use for implementing the algorithm. The first thinking was to create the coordinate system like on Fig. 4.2. All nodes and charging stations will have its own coordinate. All routes will have their own linear equation.

So we can start to describe the data in the following way:

```
#describe the first selection of positions of charging stations

    st_1 : [5.5,4.5];
    st_2 : [9.5,5.5];
    st_3 : [9.0,10.0];
    st_4 : [5.5,11.0];
    st_5 : [8.0,16.0];
    st_6 : [13.0,10.5];
    st_7 : [18.0,11.0];
    st_8 : [19.0,9.0];
    st_9 : [15.0,8.0];
    st_10 : [27.0,9.0];
    st_11 : [17.0,13.0];

#describe all lines (all roads)
#each part of route has own line equation
#we need a point on line and direction vector, coefficients: [M(x, y), V(u, w)]
#if we want calculate distance from point A(a, b) to the line we calculate the following:
# AM = (x - a, y - b) # AM x V = (x - a)*w - (y - b)*u
# distance = ((x - a)*w - (y - b)*u)/sqrt(u^2 + w^2)
```

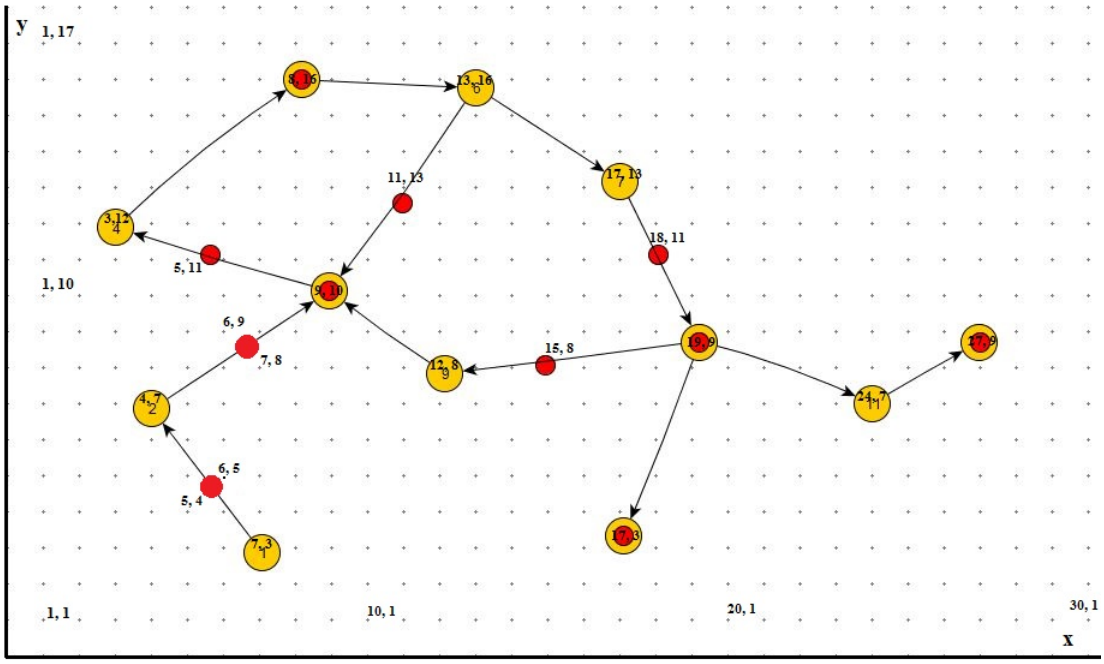

Figure 4.2: Coordinate system with charging stations.

```
        one_two_line_coeff = [7.0, 3.0, -3.0, 4.0]
        two_three_line_coeff = [4.0, 9.0, -5.0, 3.0]
        three_four_line_coeff = [9.0, 3.0, -6.0, 2.0]
        four_five_line_coeff = [3.0, 8.0, 5.0, 4.0]
        five_six_line_coeff = [0.0, 16.0, 1.0, 0.0]
        six_seven_line_coeff = [13.0, 17.0, 4.0, -3.0]
        six_three_line_coeff = [13.0, 9.0, -4.0, -6.0]
        seven_eight_line_coeff = [17.0, 19.0, 2.0, -4.0]
        eight_nine_line_coeff = [19.0, 12.0, -7.0, -1.0]
        three_nine_line_coeff = [9.0, 12.0, 3.0, -2.0]
        eight_eleven_line_coeff = [19.0, 24.0, 5.0, -2.0]
        eleven_twelve_line_coeff = [24.0, 27.0, 2.0, 2.0]
        eight_ten_line_coeff = [19.0, 17.0, -2.0, -6.0]

    #describe fitness function
    #graphic implementation
```

In researched problem we don't need so precise variables in data structure for chromosomes and there is another more simply and effective way for implementing GA. It is better to place charging stations in the same place as bus stops, by that reason we will not use the coordinate system for prediction. We will use positions of bus stops in out chromosomes, in another case we will use position in between the

two bus stops for checking the new opportunity for bus stop. The data structure for chromosomes and all necessary operators of Genetic algorithm are described in the next sections.

4.3 Implementation

4.3.1 Data structure

As we said earlier choice of data structure is very important step. For the case where we will not add new bus stops we can just use one-dimensional arrays:

```
[0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]
```

Where each value of the array presents bus stop, if the value is 0 it means that the bus stop doesn't have a charging station, if the value is 1 it means the bus stop has charging station.

For the case where we will add new bus stops we can use the following structures (dictionary type in Python):

```
cs = {
1: {2: 1.0},
2: {3: 1.0},
3: {3: 0.0, 4: 1.0},
4: {4: 0.0},
5: {5: 0.0},
6: {6: 0.0},
7: {7: 1.0},
8: {8: 0.0, 9: 1.0},
9: {9: 0.0},
10: {10: 0.0},
11: {11: 0.0},
12: {12: 1.0}
}
```

The following cases describe the positions of stations:

- 1: {1: 0.0} - no charging station in node 1;
- 1: {1: 1.0} - there is charging station in node 1;
- 1: {2: 1.0} - charging station between node 1 and node 2;
- 1: {2: 0.0} - no charging stations between node 1 and node 2;
- 1: {9: 1.0} - charging station between not connected nodes.

We will describe structure more clearly in the Chapter 7.

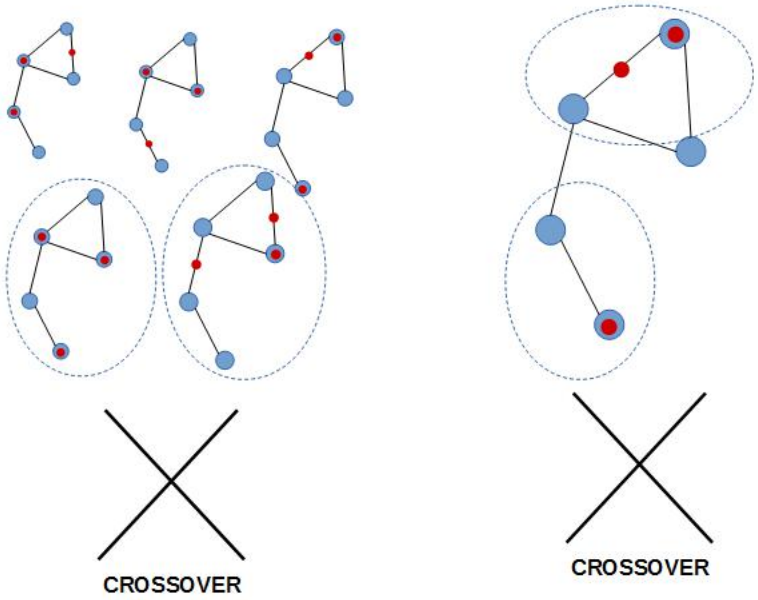

Figure 4.3: Variants to form the initial population.

4.3.2 Initial population

First step we need to implement is to choose the initial population of our charging stations. For the first population we need to generate randomly several variants of positions for charging stations. But of course it will be better to not choose all randomly. There is several cases to do this:

- create amount of routes with different charging stations positions, implement crossover between the different routes (Fig. 4.3, left);
- use only one first route with initial positions of charging stations and implement the crossover operator between the sub routes (Fig. 4.3, right).

In our algorithm we use the first case.

For first implementation I chose the first case with only four initial charging stations positions, because the main target for me now was to try to implement crossover and mutation operators. We will describe crossover and mutation operators more detailed in the next sections. We will describe GA operators using the case with new bus stops, like more complicated (the new bus stop appears between the

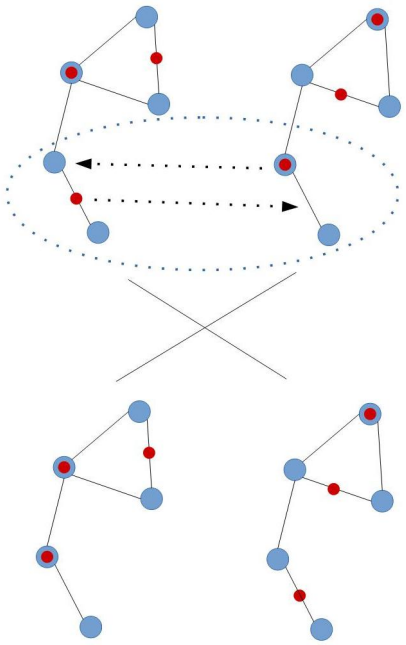

Figure 4.4: Crossover operator in the case with new bus stops.

existing bus stops), but in the Results section we will describe the results for cases without adding new bus stops, by reason that we analyse only one route and we don't need extra bus stops (in case with large amount of routes it can be applicable).

4.3.3 Crossover operator

So next we could look at other publications according to this problem, and found some interesting works. There is many interesting science works according the VRP problem solving the GA approach, for example [2], [13], [16] and [21].

The main problem in developing of genetic algorithm is to define the fitness function, crossover and mutation. There is many variants of crossover operator of genetic algorithm in Vehicle Routing Problem. First we can try to find some simple case for describing these operators. The first case can be to define the crossover and mutation like this: the mutation is changing of position of charging station to nearest node. The crossover can be different combinations of positions of charging stations of two routes. We can also use some supporting operators.

On the Fig. 4.4 we can see the possible crossover operator. We choose on the first route a subroute, the same we do with the second route. Then we just swap the subroutes.

The following code was implemented for crossover operator.
First we randomly choose two parts of the routes (subroutes):

```
def choose_parents():
    print 'Parents:'
    first_pos = math.ceil(random()*5)
    second_pos = math.ceil(random()*7)
    print(first_pos)
    print(second_pos)
    for i in range(1, 10):
        first_parent[i] = cs[first_pos+i]
        second_parent[i] = cs2[second_pos+i]
    positions = [int(first_pos), int(second_pos)]
return positions
```

Then implement crossover operator:

```
def crossover(first, second):
    temp0 = first[1]
    temp1 = first[2]
    first[1] = second[1]
    first[2] = second[2]
    second[1] = temp0
    second[2] = temp1
    offspring_1 = first
    offspring_2 = second
    arr = [offspring_1, offspring_2]
return arr
```

4.3.4 Mutation operator

The mutation operation can be described in the following way (Fig. 4.5). We can just move the charging station to the nearest bus stop or if we want to create new bus stop we can move it in the middle between existing bus stops (like on Fig. 4.5).

We implemented the following code for mutation operator, here we move the charging station to the nearest node:

```
def mutation(offspring):
    print('Mutation')
    p = math.ceil(random()*10)
# move cs to the nearest node
    offspring[p] = {p: 0.0}
    offspring[p+1] = {p+1: 1.0}
return 0
```

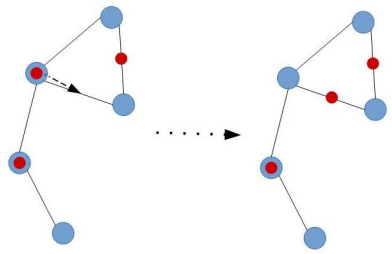

Figure 4.5: Mutation operator in the case with new bus stops.

4.3.5 Graphic representation

For graphic representation we use classes *Node*, *Arc* and *ACO* from present program. We show all charging stations for the case with new bus stops using the following code:

```
if cs_array[int(variable.get())][i] == {i: 1.0}:
    cs_positions[i-1] = node[i-1]
        if cs_array[int(variable.get())][i] == {j: 1.0}:

            cs_positions[i-1] = (node[i-1] + node[j-1])/2
        if cs_array[int(variable.get())][i] == {i: 1.0, j: 1.0}:
            cs_positions[i-1] = node[i-1]
            cs_positions[i] = (node[i-1] + node[j-1])/2
```

4.4 Fitness function - ACO algorithm

As fitness function we can use ACO approach, which will rank the results of the genetic algorithm. We will describe the ACO algorithm in next chapter.

4.5 Results

In this report we presented one available simple example of genetic algorithm operators. On Fig. 4.6, 4.7, 4.8, 4.9 we can see the output of the application - the initial positions of charging stations and two offsprings, which we get after crossover between two routes.

The next step will be to implement the ranking function for genetic algorithm. We will use the ACO approach as ranking function . The ACO approach was already

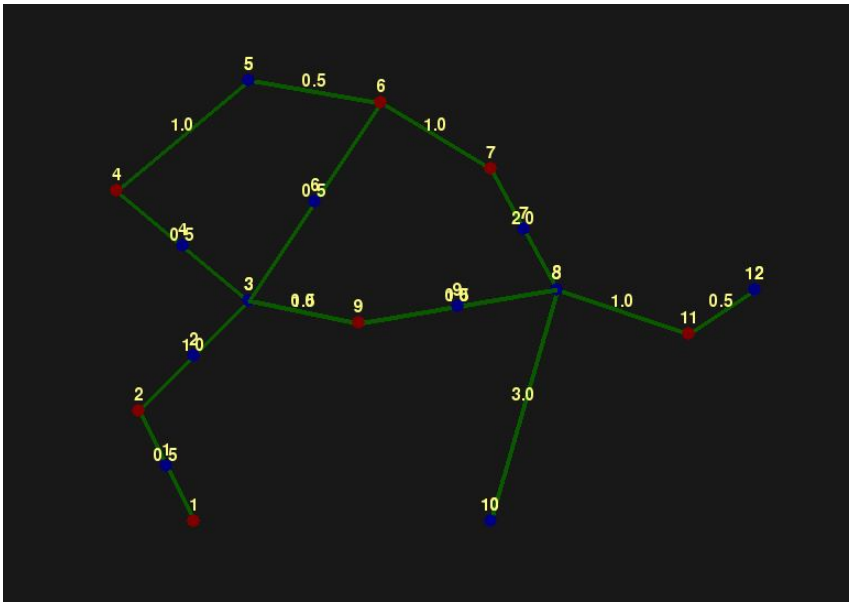

Figure 4.6: Initial positions of charging stations, first parent.

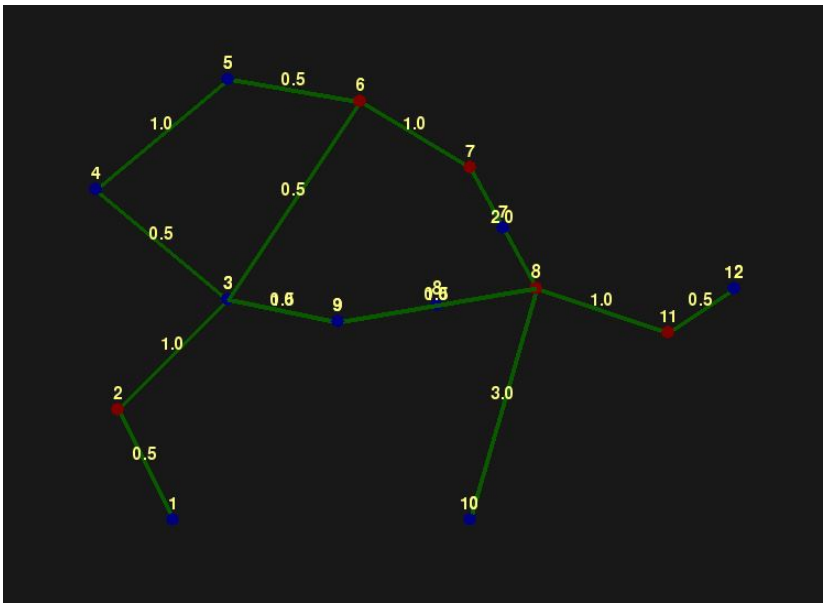

Figure 4.7: Initial positions of charging stations, second parent.

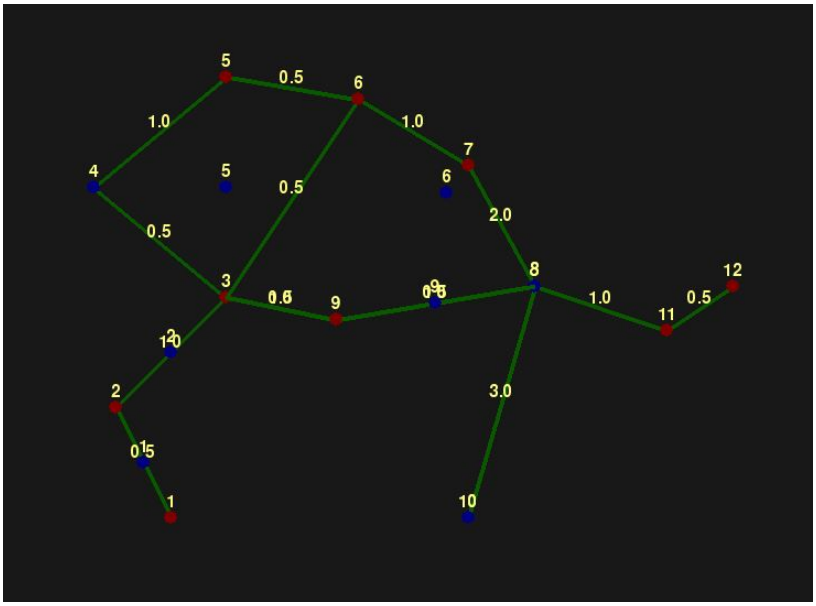

Figure 4.8: First offspring.

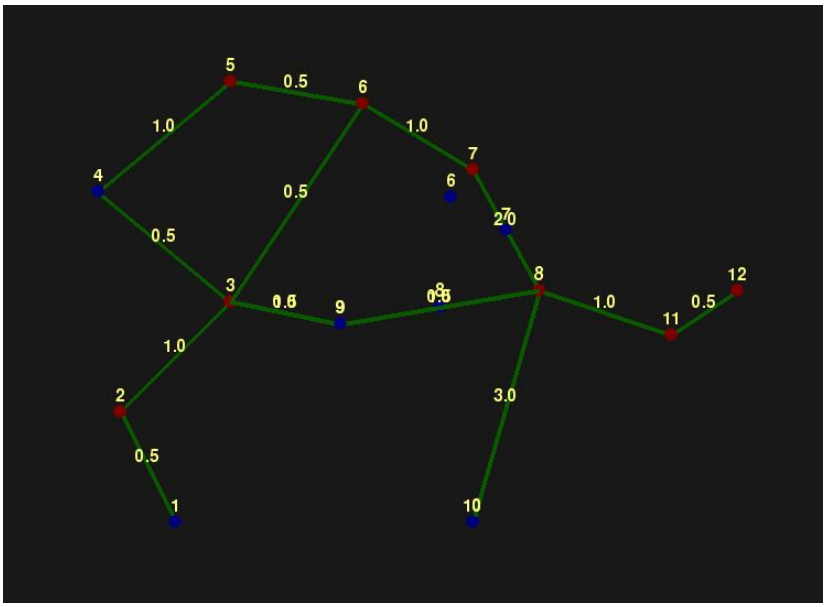

Figure 4.9: Second offspring.

implemented by Kristoffer Tangrand, but we need to improve and add other cost parameters to the arc.

Chapter 5

ACO improvement

In this Chapter we will analyse how to improve the presented ACO algorithm, developed by Kristoffer Tangrand for our problem. For our ACO implementation we use the present application of Kristoffer [18]. But we need to improve the base class according to our requirements.

The following code describes in which way we add cost parameters.

```
def step(self):

    alt = self.graph[self.pos]
    if not alt:
        return None
    dest = wrand(alt)
    arc = self.graph[self.pos][dest]
    time = 0
    while arc.cost > self.energy:
        time += self.graph[self.pos].costPerTime
        self.energy += 1.0
    self.cost += arc.cost
    self.cost += time
    self.energy -= arc.cost
    self.pos = dest
    self.path.append(dest)
    return self.pos
```

We will use the following part of the code to add other cost parameters:

```
self.cost += arc.cost
self.cost += time
```

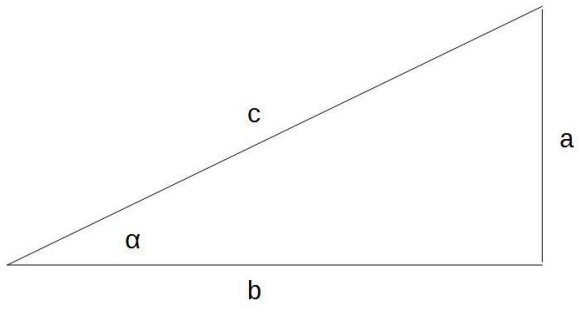

Figure 5.1: The energy consumtion will depend on angle of inclination.

5.1 Topography analysis

In this section we will analyse the possible influence of topography to energy consumption. The energy consumption can increse in several times depending on the angle of inclination. In the presented project we will analyse only topography factor. In the future we can also analyse other factors like temperature, congestions, road conditions, traffic incidents (see specification).

5.1.1 Uphill driving

If the route from point A to point B is uphill, the energy consumtion will depend on angle of inclination.

We can simply describe dependence of energy consumption with $\sin\alpha$. The function $\sin\alpha$ increases on the segment $\left[0, \frac{\pi}{2}\right]$. The bigger $\sin\alpha$ is the bigger will the energy consumption be (Fig. 5.1):

$$\sin\alpha = \frac{a}{c}$$

where a is the difference between meters above sea level of two different bus stops and c is the distance between bus stops.

The following code can be used:

```
self.energy -= arc.cost                 #distance
self.energy -= self.energy*arc.hight    #topography: hight = sin(angle)
```

But we can also use extra forces that affects the bus (Fig. 5.2) [23].

$$A = F_1 c + F_{fr} c$$

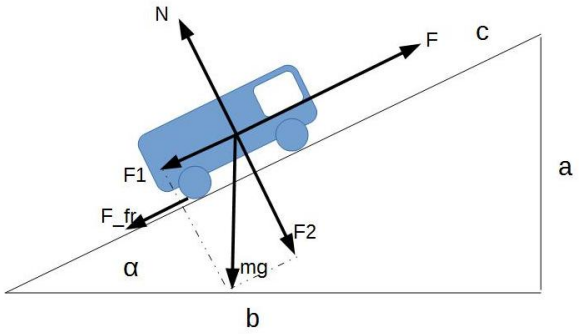

Figure 5.2: Extra energy is expended when going uphill.

where

$$F_1 = mg \sin \alpha$$

$$F_{fr} = \mu mg \cos \alpha$$

where μ- friction coefficient.

$$cos\alpha = \sqrt{1 - \sin^2 \alpha}$$

$$A = mgc(\sin \alpha + \mu \cos \alpha)$$

$$P = \frac{A}{t}$$

The code:

```
mass = 15000 + 30*70 # kg
a = 5 #meters
c = 500 #meters
f = 0.72 #traction coefficient for dry asphalt
A = mass*9.8*c(math.sin(a/c) + f*math.cos(a/c)) #Joule
t = 5*60 #seconds
P = A/t #Watt
```

Data that we need for analysis of topography when the bus goes uphill is presented in the Table 5.1.

Table 5.1: Data that we need for analysis of topography when the bus goes uphill.

Parameters	Measure
Mass of the bus	kg
Distance between the bus stops	km
Difference between the meters above sea level	m
Speed of the bus	km/h
Timetable	h
Total amount of seats	
Average amount of passengers	

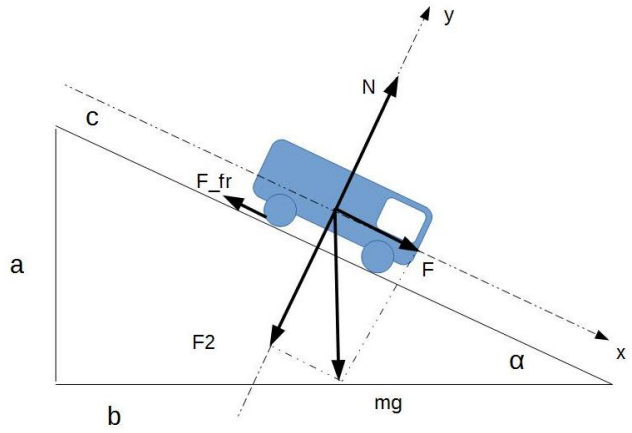

Figure 5.3: Less energy is expended going downhill.

5.1.2 Downhill driving

In this example, if the bus is going downhill, the total cost of travel will decrease (Fig. 5.3) [23].

The second law of dynamics [23]:

$$m\overrightarrow{a} = m\overrightarrow{g} + \overrightarrow{N}$$

$$Ox : mg\sin\alpha = ma_x$$

Table 5.2: Data that we need for analysis of topography when the bus goes downhill.

Parameters	Measure
Mass of the bus	kg
Speed of the bus	km/h
Distance between the bus stops	km
Difference between the meters above sea level	m

Data that we need for analysis of topography when the bus goes downhill is presented in the Table 5.2.

Also down-hill driving opens opportunity for quick charging by means of brakes and braking. Some buses have this opportunity. In presented project we don't use charging by means of brakes anf braking, we just analyse the energy which we will lose. In the future we can improve the application by adding the opportunity to charge going downhill and maybe it can help to decrease the amount of charging stations.

5.2 Simplify the calculations

To simplify the calculations we can use the following data:

- E_{diesel} - amount of energy that diesel bus uses on flat road for 1 km
- $E\alpha_{diesel}$ - amount of energy that diesel bus uses going uphill with angle α for 1 km
- E_{el} - amount of energy that electric bus uses on flat road for 1 km
- $E\alpha_{el}$ - amount of energy that electric bus uses going uphill with angle α for 1 km

We can get the first three parameters and calculate the fourth one using the following proportion:

$$\frac{E\alpha_{diesel}}{E_{diesel}} = \frac{E\alpha_{el}}{E_{el}}$$

$$E\alpha_{el} = \frac{E\alpha_{diesel} * E_{el}}{E_{diesel}}$$

We can get data for diesel for one angle and calculate values for diesel for other angles using the following proportion:

Table 5.3: Increasing of energy consumption depending on incline.

Consumption increases going uphill 5% incline	approximately by 4.5 times
Consumption increases going uphill more than 5% incline	approximately by 7.3 times

$$\frac{E\alpha_{diesel}}{E\beta_{diesel}} = \frac{\alpha}{\beta}$$

$$E\beta_{diesel} = \frac{E\alpha_{diesel} * \beta}{\alpha}$$

The following code illustrates this:

```
angle = 30 # angle about which we got the necessary data
energy_d = x (unknown) #amount of energy that diesel bus uses on flat road for 1 km
energy_d_angle = y (unknown) #amount of energy that diesel bus uses going
#uphill with angle alpha for 1 km
energy_el = 1.2 #amount of energy that electric bus uses on flat road for 1 km
energy_d_arc_angle = (energy_d_angle*arc.angle)/angle
energy_el_arc_angle = (energy_d_arc_angle*energy_el)/energy_d
self.energy -= arc.dist*energy_el_arc_angle
```

Now we don't use this code because we don't have the value of consumption for certain angle for diesel buses. But we can use the data from Table 5.3.

We use the following code to describe the increasing of energy consumption:

```
if (arc.angle == 0.0):
    self.energy -= arc.dist*energy_km
if ((arc.angle > 0.0)&(arc.angle <= 2.0)):
    self.energy -= (arc.dist*energy_km)*2.3
if ((arc.angle > 2.0)&(arc.angle < 5.5)):
    self.energy -= (arc.dist*energy_km)*4.5
if (arc.angle > 5.5):
    self.energy -= (arc.dist*energy_km)*7.3
```

Chapter 6

Connection of GA and ACO

6.1 Connecting two algorithms

The next step is to connect both algorithms in one application.

The ranking is based on cost of charging stations and battery capacity of the bus. The following code shows this:

```
for j in range(0, len(all_cost)):
    for i in range(len(all_cost) - 1, j, -1):
        if (all_cost[i] < all_cost[i-1]):
            temp = cs_array[i]
            cs_array[i] = cs_array[i-1]
            cs_array[i-1] = temp
            temp_d = all_cost[i]
            all_cost[i] = all_cost[i-1]
            all_cost[i-1] = temp_d
            temp_d = all_energy[i]
            all_energy[i] = all_energy[i-1]
            all_energy[i-1] = temp_d
```

Exclude not suitable results:

```
for i in range(0, len(all_energy)):
    if (all_energy[i] < 0):
        del all_cost[0]
        del cs_array[0]
```

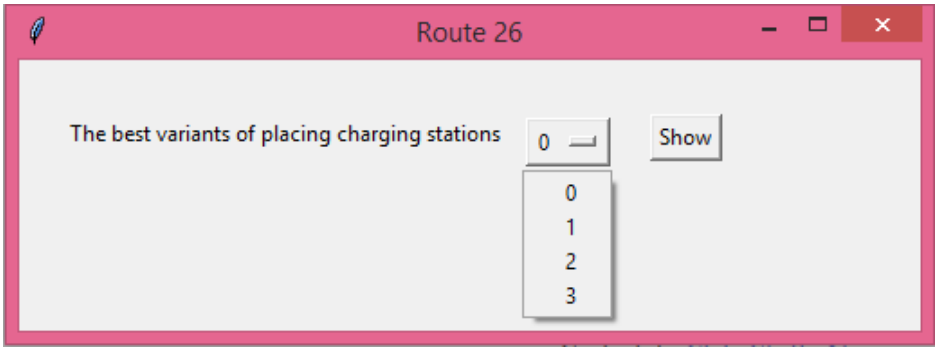

Figure 6.1: Interface of the program.

6.2 Interface

The interface of the program is presented on Fig. 6.1. It has one dropdown list and one button. From the dropdownlist we can select the four best variants of placing charging stations. When we click on the button "Show" the window with graphic will be opened. On the picture we can see the route with bus stops.

After pushing on the button "Show" the window with graphic result will be appear (Fig. 6.2).

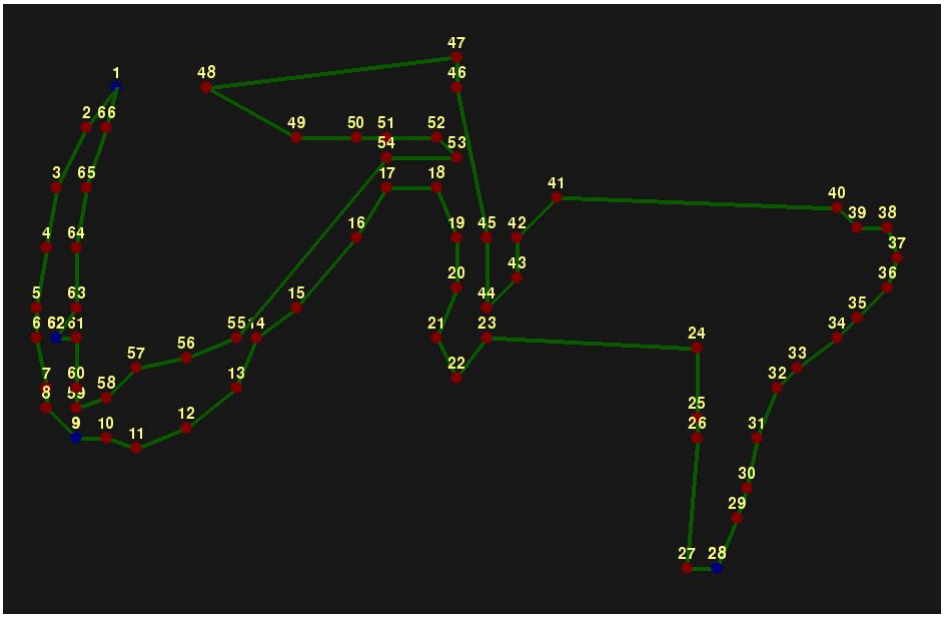

Figure 6.2: Output of the program.

Chapter 7

Results

7.1 Analysed cases

We will analyse the five following cases:

- no new bus stops, homogeneous charging stations, fixed battery capacity
- no new bus stops, inhomogeneous charging stations, fixed battery capacity
- new bus stops, homogeneous charging stations, fixed battery capacity
- new bus stops, inhomogeneous charging stations, fixed battery capacity
- no new bus stops, homogeneous charging stations, variable battery capacity

7.1.1 No new bus stops, homogeneous charging stations, fixed battery capacity

In this case we use the following type of chromosome:

```
[0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]
```

where "0" means that there is no charging station in the node and "1" means thet there exists charging station in the node. On Fig. 7.1 we can see the example of placing charging stations only on bus stops.

The following code presents how we choose the positions of charging stations

```
for j in range(0, len(cs_array)):
    for i in range(0, len(cs)):
        if cs_array[j][i] == 1:
            all_cs.append(i+1)
```

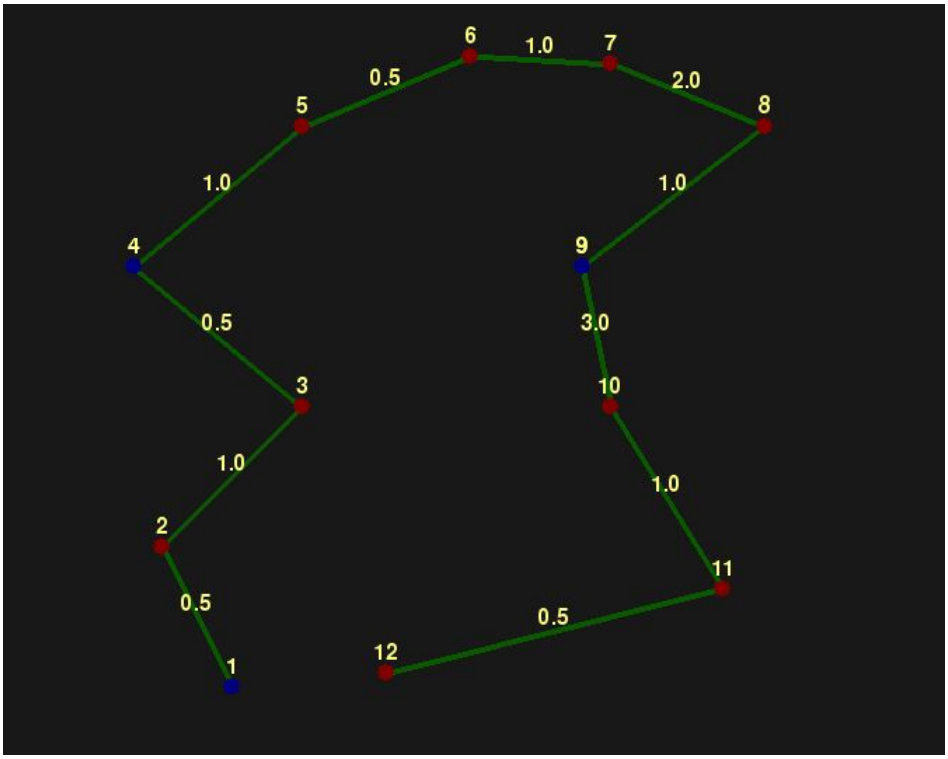

Figure 7.1: Example of placing charging stations only on bus stops.

```
aco.get_all_cs(all_cs)
cs_amount = len(all_cs)
print('All cs:')
print(all_cs)
cs_cost = cs_amount*100
all_cost.append(cs_cost)
```

7.1.2 No new bus stops, inhomogeneous charging stations, fixed battery capacity

In this case we use the following type of chromosome:

```
[0.9, 0, 0, 0.7, 0.7, 0, 0, 0.9, 0, 1, 0, 0.7]
```

Here we also have positions of charging stations with different parameters.

7.1.3 New bus stops, homogeneous charging stations, fixed battery capacity

In this case we use the following type of chromosome:

```
cs = {
1: {1: 0.0},
2: {3: 0.0},
3: {3: 0.0},
4: {4: 0.0, 5: 1.0},
5: {5: 1.0},
6: {6: 0.0},
7: {7: 0.0},
8: {8: 1.0},
9: {9: 0.0},
10: {10: 1.0},
11: {11: 0.0},
12: {12: 0.0}
}
```

Here we can add charging stations between nodes. On the Fig. 7.2 the charging stations (the new bus stops) appear between nodes 6 and 7, and also between nodes 7 and 8.

7.1.4 New bus stops, inhomogeneous charging stations, fixed battery capacity

In this case we use the following type of chromosome:

```
cs = {
1: {1: 0.0},
2: {3: 0.0},
3: {3: 0.0},
4: {4: 0.0, 5: 0.7},
5: {5: 0.3},
6: {6: 0.0},
7: {7: 0.0},
8: {8: 0.5},
9: {9: 0.0},
10: {10: 0.9},
11: {11: 0.0},
12: {12: 0.0}
}
```

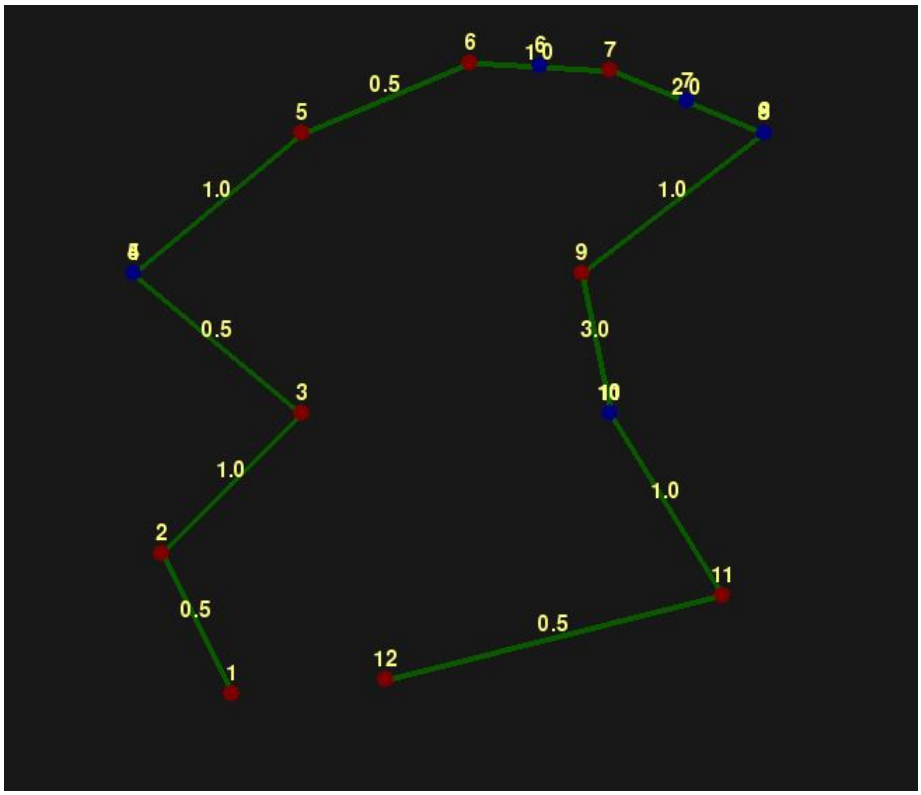

Figure 7.2: Example of placing charging stations not only on bus stops.

Table 7.1: The data about chosen bus route.

Parameter	Value
Chosen bus route	26
Total time of the route	1h 10 min (both directions)
Average speed on the route	19 km/h
Consumption of diesel for bus (100 km)	55 l
Consumption increase going uphill 5% incline	by 4.5 times
Consumption increase going uphill more than 5% incline	by 7.3 times
Consumption of electric bus (1 km)	1.2 kWh
Amount of trips for one bus a day	14

Table 7.2: The bus parameters.

Parameter	Value
Weight	12 400 kg (empty) /18 000 kg
Amount of passengers	32 (seated), 49 (standing)
Length of the bus	12 m
Battery capacities for bus	120 kWh

7.1.5 No new bus stops, homogeneous charging stations, variable battery capacity

In this case we use the following type of chromosome:

```
[1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 100]
[1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 120]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 80]
[1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 110]
```

The last value in the chromosome can be the battery capacity.

7.2 Real data

As experemental route we chose the route 26 in Tromsø.

We got the following data from Troms fylkeskommune (Tables 7.1, 7.2, 7.3, 7.4, 7.5).

Table 7.3: The charging station parameters.

Parameter	Value
Type of connection	Pantograph system
Type of charging station	300 kW
Charging for 2 minutes on bus stop	+10 kWh
Charging for 3 minutes on depot	+15 kWh
Charging for 8 minutes on depot	+40 kWh
Charging for 18 minutes on depot	+90 kWh

Table 7.4: The general overwiew of topography on the rooute 26 direction 1.

Parameter	Value
Meters above sea level (start of the route)	7 m
Distance to highest point	5542 m
Highest point (meters above sea level)	104 m
Distance from highest point	4714 m
Meters above sea level (end of the route)	5 m

Table 7.5: The general overwiew of topography on the rooute 26 direction 2.

Parameter	Value
Meters above sea level (start of the route)	5 m
Distance to highest point	7266 m
Highest point (meters above sea level)	104 m
Distance from highest point	5156 m
Meters above sea level (end of the route)	7 m

Table 7.6: Route 26, direction 1.

Bus stop	Time	Distance (m)	Topography
Giæverbukta	05:57	0	Flat
Postterminalen	05:59	928	Flat
Sjølundvegen	06:00	336	Flat
Fiolvegen	06:01	374	Flat
UNN, Åsgård	06:02	645	Flat
Lars Eriksens veg	06:02	372	Uphill 5.2%
Holtvegen	06:03	480	Uphill 5.2%
Barduvegen	06:04	304	Uphill 5.2%
Elverhøy	06:05	158	Uphill 5.2%
Sommerlyst skole	06:05	301	Uphill 5.0%
Kirkegården	06:06	145	Uphill 5.0%
Vervarslinga	06:06	280	Uphill 5.0%
Trykkbassenget	06:07	211	Uphill 5.0%
Myrengvegen sør	06:07	244	Uphill 5.0%
Myreng	06:08	264	Uphill 5.0%
Grimsbyvegen	06:09	217	Uphill 5.0%
Skoglyst	06:10	283	Uphill 5.0%
Maristuen	06:10	190	Downhill 4.6%
Snarvegen	06:11	342	Downhill 4.6%
Petersborggata	06:12	569	Downhill 4.6%
Kongsbakken	06:13	360	Downhill 4.6%
Wito	06:15	166	Downhill 4.6%
Sjøgata S1	06:20	318	Downhill 4.6%
Skippergata	06:21	353	Uphill 8.0%
Tromsdalen Bruvegen	06:23	1301	Bridge
Novasenteret	06:24	732	Downhill 8%
Pyramiden	06:25	383	Flat

Data from chosen route 26

Route 26, direction 1: travel time 28 min, 27 bus stops (Table 7.6). The percent of the incline in the tables is calculated according to the following formula:

$$\frac{diff}{distance} * 100$$

, where *diff* - height difference between two points in meters, *distance* - distance between two points in meters.

Route 26, direction 2: travel time 42 min, 40 bus stops (Table 7.7).

Table 7.7: Route 26, direction 2.

Bus stop	Time	Distance (m)	Topography
Pyramiden	06:25	0	Uphill 2%
Hjorten	06:25	231	Uphill 2%
Isbjørnvegen	06:26	219	Uphill 2%
Nordselvegen	06:26	238	Uphill 2%
Møllervegen	06:27	259	Uphill 2%
Fjellheisen	06:27	252	Uphill 2%
Risøyvegen	06:28	239	Uphill 2%
Ecornsenteret	06:29	234	Uphill 2%
Gausdalsvegen	06:30	284	Uphill 2%
Fløyvegen	06:31	165	Uphill 2%
Åsvegen	06:32	369	Downhill 2%
Tromsdalen skole	06:33	263	Downhill 2%
Skogvegen	06:34	287	Downhill 2%
Tromsdalen Kirke	06:35	225	Uphill 8%
Skippergata	06:37	1265	Bridge
Torgcenteret	06:40	314	Downhill 8%
Fr. Langes gate F2	06:45	334	Uphill 7%
Petersborggata	06:46	464	Uphill 7%
Bispegården	06:47	600	Uphill 7%
Myrengvegen sør	06:47	260	Uphill 7%
Myreng	06:48	264	Uphill 5% (up and down hills)
Grimsbyvegen	06:49	217	Uphill 5% (up and down hills)
Skoglyst	06:50	283	Uphill 5% (up and down hills)
Maristuen	06:50	190	Uphill 5% (up and down hills)
Snarvegen	06:51	342	Uphill 5% (up and down hills)
Bispegården	06:52	193	Uphill 5% (up and down hills)
Trykkbassenget	06:53	173	Uphill 5% (up and down hills)
Vervarslinga	06:53	213	Uphill 5% (up and down hills)
Kirkegården	06:54	280	Uphill 5% (up and down hills)
Sommerlyst skole	06:55	196	Uphill 5% (up and down hills)
Elverhøy	06:56	243	Downhill 5%
Barduvegen	06:56	160	Downhill 5%
Holtvegen	06:57	305	Downhill 5%
Boligstiftelsen	06:57	292	Downhill 5%
Lars Eriksens veg	06:58	245	Downhill 5%
UNN, Åsgård	06:59	256	Downhill 5%
Fiolvegen	07:00	652	Downhill 5%
Sjølundvegen	07:01	344	Downhill 5%
Postterminalen	07:03	440	Flat
Giæverbukta	07:07	632	Flat

Table 7.8: Schedule of charging on depots during the working day.

№ trip	Depot	Pyramiden	Pyramiden	Depot	Stop (min)	Charging time
1	5:57	6:25	6:25	7:07	5	3
2	7:12	7:40	7:40	8:22	10	8
3	8:32	9:00	9:00	9:42	10	8
4	9:52	10:20	10:20	11:02	10	8
5	11:12	11:40	11:40	12:22	10	8
6	12:32	13:00	13:00	13:42	10	8
7	13:52	14:20	14:20	15:02	10	8
8	15:12	15:45	15:45	16:32	0	0
9	16:32	17:00	17:00	17:42	5	3
10	17:47	18:15	18:15	18:57	20	18
11	19:17	19:45	19:45	20:27	20	18
12	20:47	21:15	21:15	21:57	20	18
13	22:17	22:45	22:45	23:27	20	18
14	23:47	00:15	00:15	00:29 - Torgcenteret		

Presented analysis of topography was done with a bit overlapping, it was done to consider the worst result. In practice many factors can influence on energy consumption besides the topography. These factors we can consider in future work.

In Table 7.8 we can see all trips and possible charging time for the bus on depot (the time for connection to pantograph and unconnection we consider 1 min).

We will also consider that the charging time on charging stations which are placed on bus stops is 2 minutes (plus the time for connection to pantograph and unconnection - 1 min).

7.3 Execution and results

First we construct our route (Fig. 7.3). The graphic representation of the route on the output was made schematically, the drawing scale is not accurate.

The distance between the bus stops and approximate percent of incline:

```
...
aco[7][8] = Arc(0.304, 5.2)
aco[8][9] = Arc(0.158, 5.2)
aco[9][10] = Arc(0.301, 5.2)
aco[10][11] = Arc(0.145, 5.0)
...
```

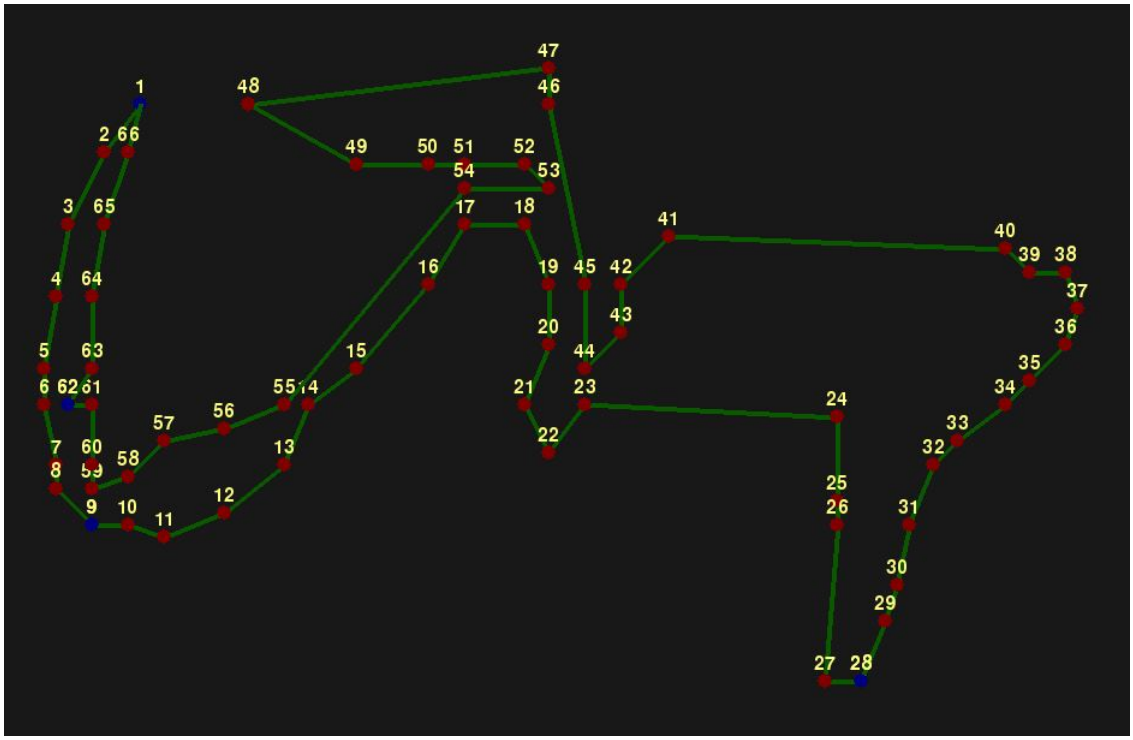

Figure 7.3: Route 26.

Add the parameters to the code:

```
energy_km = 1.2 #kWh energy consumption for 1 km
self.energy = 120.0 #kWh battery capacity - start amount of energy
```

Amount of trips we take are 14 for both directions of the route.

Result output

We will consider that the cost of one charging station is 100. We will consider the first, second and the fifth cases for both directions like more interesting cases. The rest of the cases (third and fourth) we will consider as examples only for one direction.

For better understanding of the output of the program we can describe the steps which the application does. The application does three cycles with considered amount of trips (n). Every cycle algorithm offer several positions for charging stations and the bus goes through the bus stops n times, - simulating one work day, and these results are ranked. The best results goes to the next cycle. The one best result after three cycles is presented in the tables below. Also it is important to note that when we choose the best result we consider only variants when the amount of energy is more than 30 percent of battery capacity.

The best result after n trips means that we did three cycles only when bus goes n trips.

We are interested in 14 trips for route 26 (one work day), but we also consider the result after 5 trips, just to see the difference. For the cases three and four we considered 28 trips as an example. Also it is important to say that we can get different results for the same amount of charging stations at the same route. We can easily explain it. The battery capacity of the bus is fixed, so if on one charging station the bus will get more energy than the battery capacity, the amount of energy will be equal the battery capacity.

7.3.1 Case 1: no new bus stops, homogeneous charging stations, fixed battery capacity

In case 1 we present the results, when we consider both directions of the route. Here we consider that on bus stop number 1 is a depot, where the bus can charge from 3 to 18 minutes depending on the number of trip. On other bus stops the bus can charge 2 min. The results are presented in Tables 7.9, 7.10.

Here we can see that we need 4 charging stations. We can analyse all routes on the map of Tromsø and place charging stations on bus stops which are general for several bus routes, so we can use these charging stations not only for one route, by that we can decrease the cost. For now GA algorithm offers positions for charging

Table 7.9: The best result after 5 trips.

Parameter	Value
Amount of charging stations	4
The rest of the energy in the end of the day (start with 120 kWh)	40.773 kWh
Cost of charging stations	400
Battery capacity	120 kWh

Table 7.10: The best result after 14 trips.

Parameter	Value
Amount of charging stations	5
The rest of the energy in the end of the day (start with 120 kWh)	44.977 kWh
Cost of charging stations	500
Battery capacity	120 kWh

stations without considering the other routes, later we can add such check. The output for case 1 is presented on Fig. 7.4.

7.3.2 Case 2: no new bus stops, inhomogeneous charging stations, fixed battery capacity

The results, if we consider both directions of the route (Tables 7.11, 7.12). The output for case 2 is presented on Fig. 7.5. Here we need to say that best result from first population in GA was 6 charging stations with the rest of the energy 60.977 kWh in the end of the working day. So in the future we need to work for better convergence of the algorithm.

Table 7.11: The best result after 5 trips.

Parameter	Value
Amount of charging stations	6
The rest of the energy in the end of the day (start with 120 kWh)	91.045 kWh
Cost of charging stations	600
Battery capacity	120 kWh

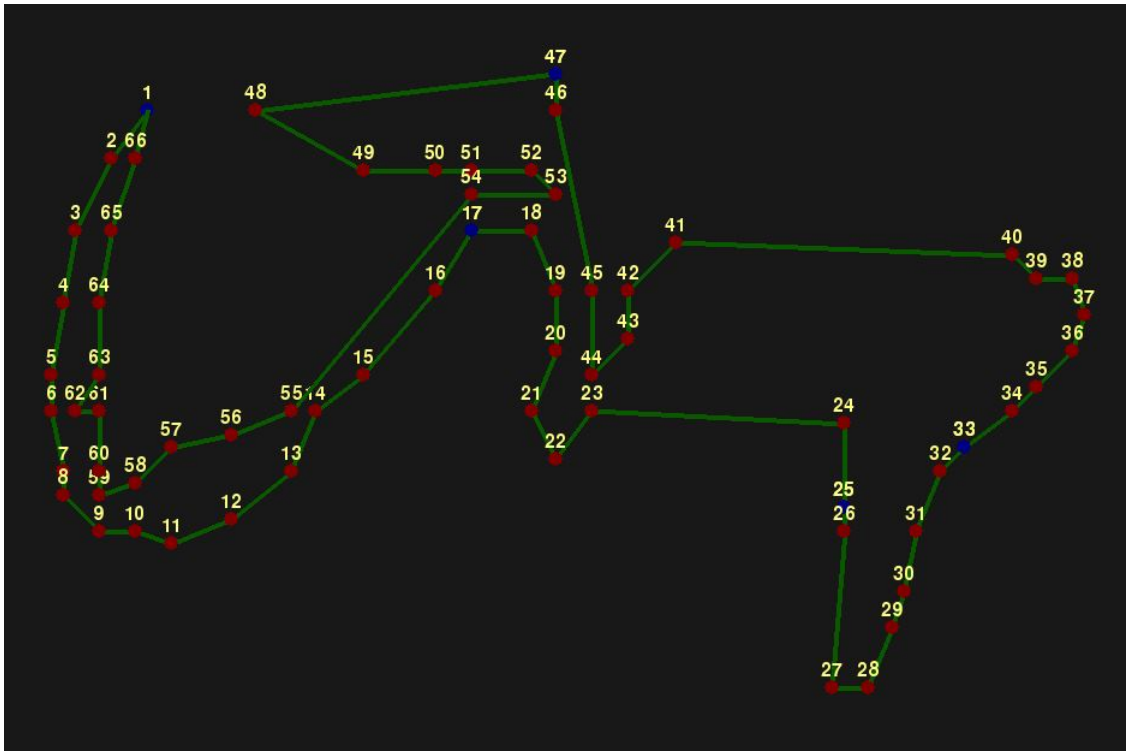

Figure 7.4: Output for case 1.

Table 7.12: The best result after 14 trips.

Parameter	Value
Amount of charging stations	7
The rest of the energy in the end of the day (start with 120 kWh)	58.977 kWh
Cost of charging stations	700
Battery capacity	120 kWh

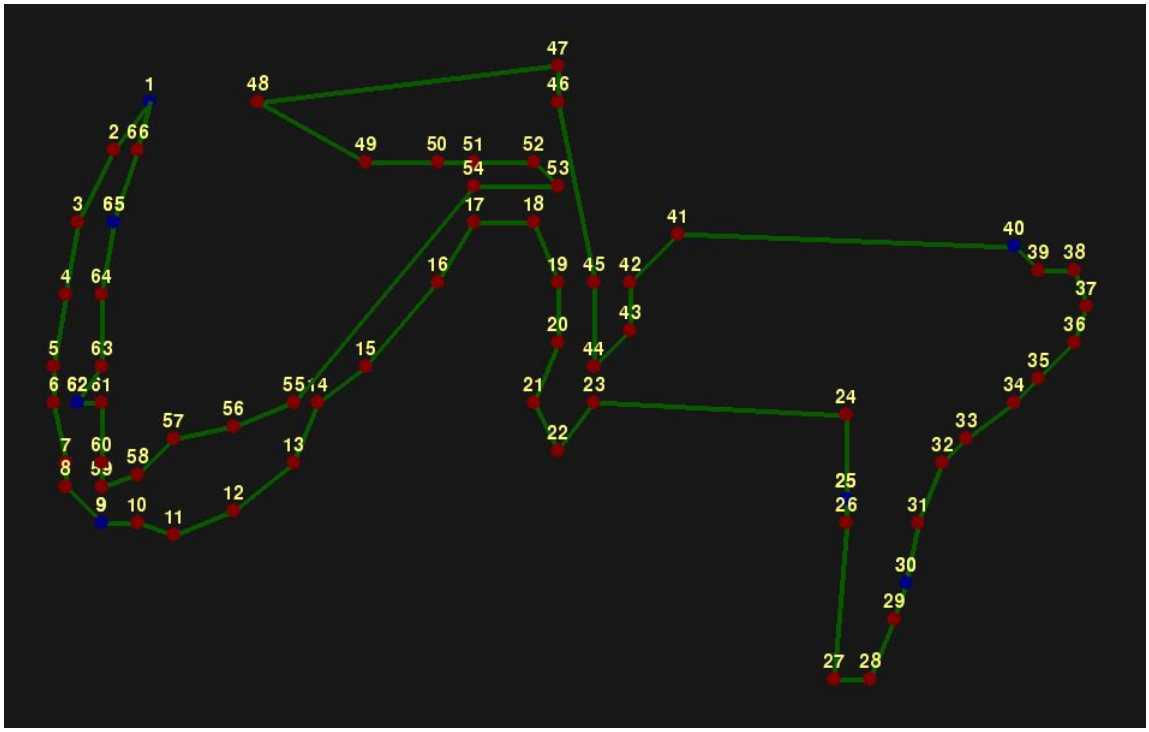

Figure 7.5: Output for case 2.

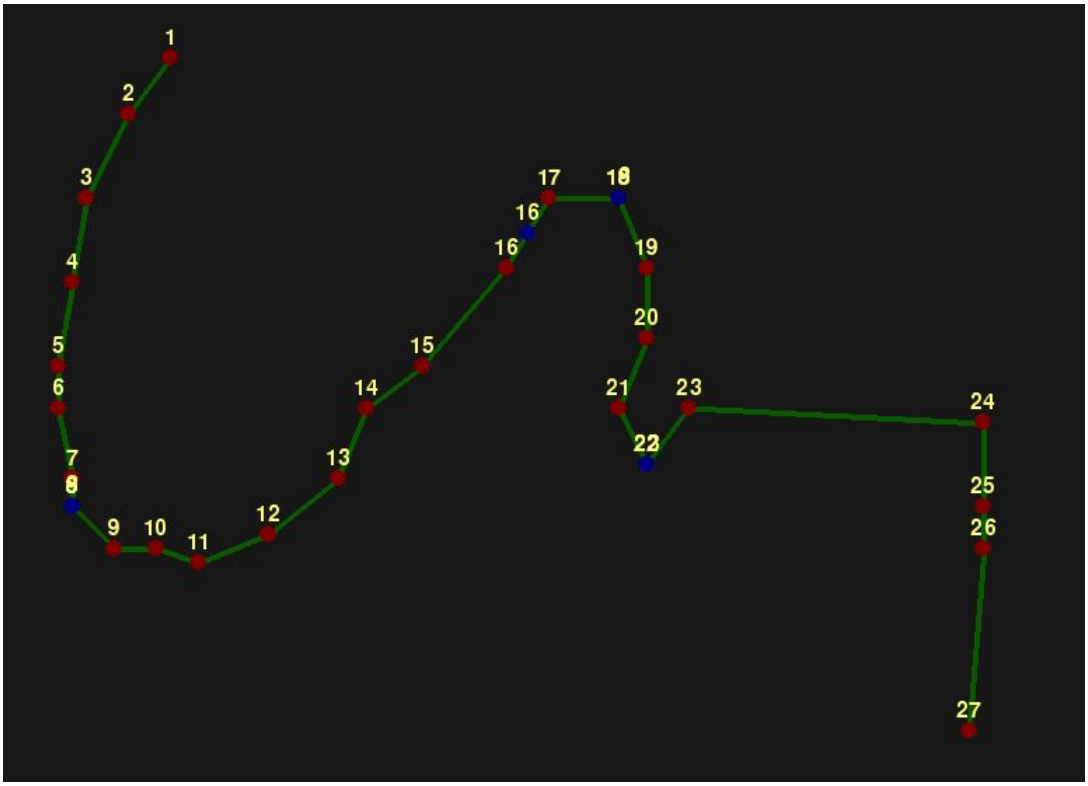

Figure 7.6: Output for case 3.

7.3.3 Case 3: new bus stops, homogeneous charging stations, fixed battery capacity

This case we consider as an example case and it was implemented only for one direction of the route. We can see on Fig. 7.6 that new bus stop appears between the bus stops number 16 and 17.

7.3.4 Case 4: new bus stops, inhomogeneous charging stations, fixed battery capacity

This case we consider as an example case and it was implemented only for one direction of the route. As we can see on Fig. 7.7 we need more charging stations than in case 3, by reason that we use inhomogeneous charging stations. We can also see that several charging stations can appear between the bus stops, but we can move them to the nearest bus stop.

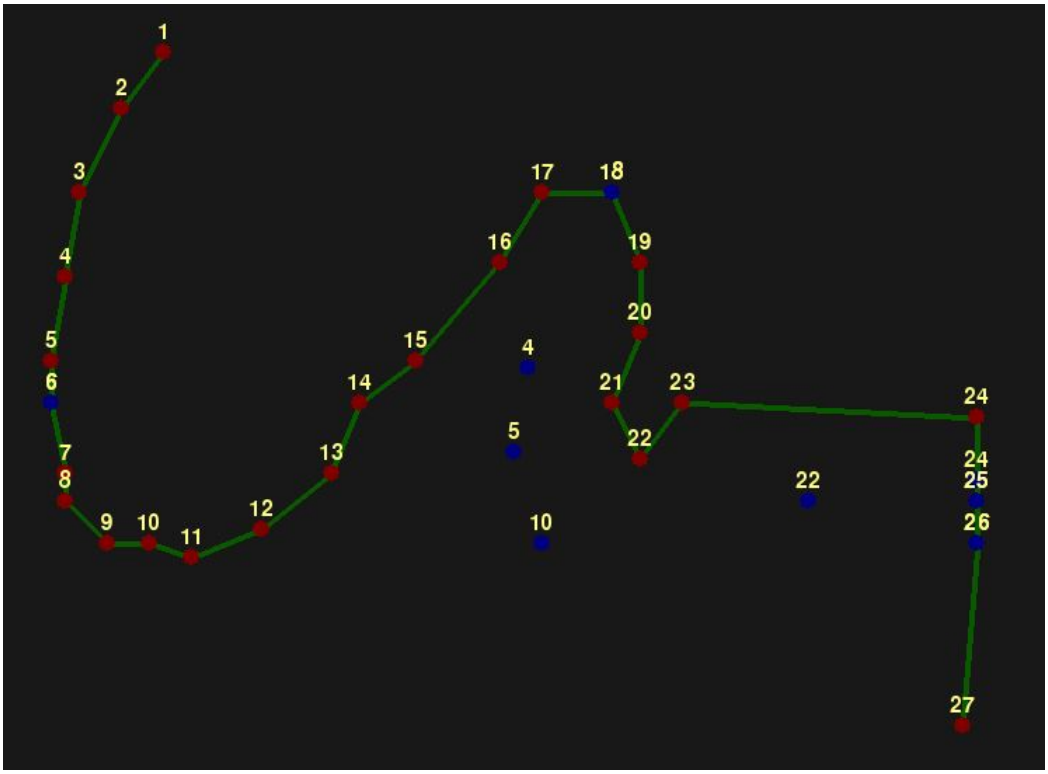

Figure 7.7: Output for case 4.

Table 7.13: The best result after 5 trips.

Parameter	Value
Amount of charging stations	4
The rest of the energy in the end of the day	40.773 kWh
Cost of charging stations	500
Battery capacity	120 kWh

Table 7.14: The best result after 14 trips.

Parameter	Value
Amount of charging stations	5
The rest of the energy in the end of the day	34.977 kWh
Cost of charging stations	500
Battery capacity	110 kWh

7.3.5 Case 5: no new bus stops, homogeneous charging stations, variable battery capacity

The results, if we consider both directions of the route (Tables 7.13, 7.14). We can see from results in Table 7.14 that we it is possible to decrease the battery capacity for the bus. The output for case 5 is presented on Fig. 7.8.

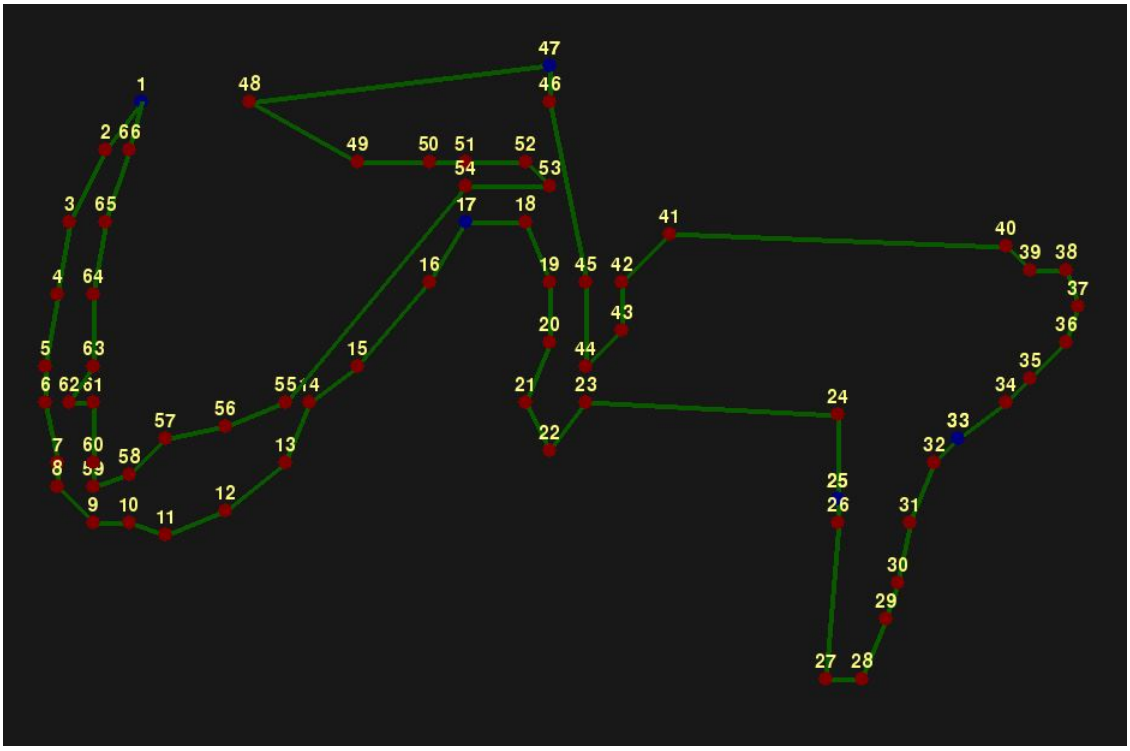

Figure 7.8: Output for case 5.

Chapter 8

Concluding Remarks

8.1 Discussion

In this work we used the machine learning methods, but we didn't discussed why we apply such methods for present problem. In this part we want to compare the classical optimization methods with GA and ACO approach.

Advantages of GA [9], [14]:

- Suitable for large-scale optimization problems;
- Can be used in problems with the changing environment;
- Effective parallelization;
- Does not need any information about the behavior of the function (continuous, differentiability);
- Easy to implement.

GA disadvantages [14]:

- We can not optimally encode parameters for all problems;
- Difficult to apply for multi extremal problems (Rastrigin function), GA is difficult to apply for isolated functions.

The optimization method for vehicle routing problem can be for example: The Least Squares Method, Lagrangian Method [12] and others. Let's consider the simple problem. We have a coordinate system and we have three points, which are not on a line. We need to find the line which will be near to the all points, it means the distance between the point and the value on the line will be possibly minimum. Let's consider the same problem, but the amount of points will be much more. To solve

this problem with the least squares method will not be so much complicated as when we had just three points.

Instead of simple example presented above the Vehicle Routing Problem is the NP-hard problem. The main reason why we use GA for such large problems is because we don't need a precise solution, we can use close-to-optimal solution. Of course for one route we can use more simple algorithms, instead of using GA connecting with ACO approach for calculating the energy and ranking results. But let's consider the large network with a big amount of routes. Having our prepeard algorithm we can simply extend it to the large amount of routes. So here it already will be useful to build new bus stops and change sequence of some bus routes. And for fast working of application we can apply parallelization in genetic algorithm for calculation of different populations.

At present, GA is used for tasks such as [14]:

- Search for global extremum multiparameter function
- Approximation of functions
- The problem of the shortest path
- Location problem
- Set up an artificial neural network
- Game strategy
- Machine learning

The important question is the convergence of the genetic algorithm. In article "Convergence of genetic algorithm" [17] the authors focus on convergence of algorithm.

8.2 Conclusions

Based on the main results and discussion we can conclude that we reached the main goal of the master's thesis problem. We developed the application that can help to analyse necessary amount of charging stations for certain bus route. The following cases were analysed:

- no new bus stops, homogeneous charging stations, fixed battery capacity,
- no new bus stops, inhomogeneous charging stations, fixed battery capacity
- new bus stops, homogeneous charging stations, fixed battery capacity

- new bus stops, inhomogeneous charging stations, fixed battery capacity
- no new bus stops, homogeneous charging stations, variable battery capacity

More usefull case for our route becomes the case number five: no new bus stops, inhomogeneous charging stations, variable battery capacity. So far as we have only one testing route and this route has not so long distance between bus stops, we don't need to build new bus stops, but it is interesting to see if we can decrease the battery capacity for the bus. As we can see from results, we can do it.

The cases where we recommend new bus stop are not so applicable in one not so long route.

8.3 Recommendations for Future Work

We can add the following improvements in future application:

- Improve GA considering several bus routes.
- Add cycle for GA implementation which will check the convergence of algorithm.
- In present program we analyse just one type of system: "en route" charging. Later we can also analyse another system, using "depot recharging". This means that the buses will be recharged when they park at depots for the night.
- Also in the present program we don't use the possible charging going downhill, we can add such possibility in the future.

Bibliography

[1] Vladimir Alekseev and Vladimir Talanov. Graphs and Algorithms, lectures. The National Open University "INTUIT", http://www.intuit.ru/studies/courses/101/101/lecture/2957?page=2.

[2] Áslaug Sóley Bjarnadóttir. Solving the Vehicle Routing Problem with Genetic Algorithms. Master's thesis, Technical University of Denmark, DTU, 2004.

[3] John E. Bell and Patrick R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41 – 48, 2004.

[4] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. 2004.

[5] Esri - GIS Mapping Software. ArcGIS, article "Understanding path distance analysis", http://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/understanding-path-distance-analysis.htm.

[6] EV Norway, Information about Electric Vehicles. http://www.evnorway.no/.

[7] Liv Cecilie Evenstad. Electric buses in Tromsø? Microsoft PowerPoint presentation, 2016.

[8] FlexChEV. FlexChEV project: Flexible Electric Vehicle Charging Infrastructure. http://flexchev.com.

[9] Randy L. Haupt and Sue Ellen Haupt. *Practical genetic algorithms, second edition*. John Wiley & Sons, Inc., 2004.

[10] Fu-Sheng Ho and Petros Ioannou. Traffic flow modeling and control using artificial neural networks. *IEEE Control Systems*, 16(5):16–26, Oct 1996.

[11] Aleksander Ignatyev. Using ant colony algorithm to solve the vehicles routing problem, 2009. Moscow, Moscow State University named after M.V. Lomonosov, IV International Scientific and Practical Conference "Modern information

technology and IT-education", http://2009.it-edu.ru/docs/Sekziya_8/3_Ignat'ev_Ignatyev.doc.

[12] Gilbert Laporte. The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 1991.

[13] Beatrice Ombuki, Brian J. Ross, and Franklin Hanshar. Multi-objective Genetic Algorithms for Vehicle Routing Problem with Time Windows, Technical Report. 2004. Brock University.

[14] T. V. Panchenko. *Genetic algoritms.* The publishing house "Astrakhan University", 2007.

[15] Francisco B. Pereira, Jorge Tavares, Penousal Machado, and Ernesto Costa. GVR: a New Genetic Representation for the Vehicle Routing Problem. 2002.

[16] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 2004.

[17] R. R. Sharapov and A. V. Lapshin. Convergence of Genetic Algorithms. *Pattern Recognition and Image Analysis*, 2006.

[18] Kristoffer Tangrand. Optimal Routing of Electric Vehicles in Networks with Charging Nodes: Ant Colony Optimization with Genetic Algorithms. Master's thesis, Narvik University College, 2015.

[19] Kristoffer Tangrand and Bernt A. Bremdal. Using Ant Colony Optimization to determine influx of EVs and charging station capacities. 2015.

[20] The University of Tennessee at Chattanooga, College of Engineering & Computer Science, Center for Energy, Transportation and the Environment. article "Electric Vehicles": http://www.utc.edu/college-engineering-computer-science/research-centers/cete/electric.php.

[21] Gintaras Vaira. *Genetic algoritm for Vehicle Routing Problem.* Doctoral dissertation, Vilnus University, 2014.

[22] www.car-tesla.ru. The impact of winter on the Tesla battery and regenerative braking system, 02.12.2014. http://www.car-tesla.ru/s/vliyanie-zimy-na-akkumulyator-tesla-i-sistemu-regenerativnogo-tormozheniya.html.

[23] Hung D. Young and Roger A. Freedman. *University Physics, 12th Edition.* 2008.

Appendix A

CD index

In the following directories on the attached CD you may find:

REPORT *This report (.pdf file).*

CODE *The Python code of the application for 5 considered cases.*