WROCLAW UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF ELECTRONICS

FIELD OF STUDY: COMPUTER SCIENCE

SPECIALIZATION: ADVANCED INFORMATICS AND CONTROL

MASTER THESIS

Analysis of properties of algorithms for solving time and cost dependent TSP problem

Analiza własności algorytmów rozwiązujących zagadnienia TSP uwzględniające czas i koszt

AUTHOR:

B. Eng. Adrian Aleksander Strugała

SUPERVISOR:

Dr. Eng. Leszek Koszałka, W4/K2

GRADE:

WROCŁAW 2018

TABLE OF CONTENTS

[Abbreviations 3](#_Toc515456376)

[1. Introduction 4](#_Toc515456377)

[1.1 Motivation 4](#_Toc515456378)

[1.2 Objective 4](#_Toc515456379)

[1.3 Workflow management 5](#_Toc515456380)

[2. Problem statement 8](#_Toc515456381)

[2.1 Traveling Salesman Problem overview 8](#_Toc515456382)

[2.2 Formulation of TSP Time and Cost dependent Problem 9](#_Toc515456383)

[3. Different approaches to the cost management 10](#_Toc515456384)

[3.1 Only free roads 10](#_Toc515456385)

[3.2 Toll roads whenever possible 11](#_Toc515456386)

[3.3 Limit cost 12](#_Toc515456387)

[3.4 Evaluation of the cost of time 14](#_Toc515456388)

[4. Algorithms 15](#_Toc515456389)

[4.1 Ant Colony Optimization 15](#_Toc515456390)

[4.2 Nearest Neighbour 16](#_Toc515456391)

[4.3 God 17](#_Toc515456392)

[5. Research environment 18](#_Toc515456393)

[5.1 Traveling Eco-Salesman 18](#_Toc515456394)

[5.2 Clean Code approach 19](#_Toc515456395)

[5.3 Google Maps API and Michelin Maps API connections 23](#_Toc515456396)

[6. User’s manual 26](#_Toc515456397)

[7. Research 28](#_Toc515456398)

[7.1 research dataset 28](#_Toc515456399)

[7.2 Cost management analysis 29](#_Toc515456400)

[7.3 Experiment #1 – Italy 29](#_Toc515456401)

[7.4 Experiment #2 – Poland 35](#_Toc515456402)

[7.5 Experiment #3 – Europe 36](#_Toc515456403)

[8. Conclusion 37](#_Toc515456404)

[8.1 Summary 37](#_Toc515456405)

[8.2 Comments 37](#_Toc515456406)

[Bibliography 38](#_Toc515456407)

## Abbreviations

TSP – Traveling Salesman Problem

API – Application Programming Interface

TES – Traveling Eco-Salesman (software)

CC – Clean Code

NN – Nearest Neighbour

URL – Uniform Resource Locator

XML – Extensible Markup Language

# 1. Introduction

## 1.1 Motivation

Observing the past centuries, we can conclude, that the humans tend to search for optimal roads. Merchants, travelers, and explorers were creating more and more detailed maps [1]. In the digital era, everyone can use free and open real-time maps found on the Internet. Some of them provide a feature for finding the shortest road connecting a set of cities [2], but very few can answer common question “Is it really worth to pay for toll highway or should I just stay on the free road?” A solution of Traveling Eco-Salesman Problem provides an answer to this question.

The idea of Traveling Eco-Salesman Problem was born October 2017 during a round trip around northern Italy. The problem what was considered every evening was exactly the same as stated in the introduction: “Is it worth to get to another city by toll road?” Manually looking for a solution, calculations, searching on Google was taking two to three hours (for a single pair of cities). The natural way to improve the current situation was the will of automatization of manual work.

## 1.2 Objective

The purpose of the Master Thesis is to analyze properties of algorithms for solving time and cost dependent Traveling Salesman Problem, named Traveling Eco-Salesman Problem using the designed and implemented experimentation system.

In further sections of the Introduction used workflow management approach is described.

In Chapter 2. TSP (Traveling Salesman Problem) itself and example of the solution to the problem are described. Then, the introduction of time and cost dependency is stated and well described using a mathematical model.

The first analysis was made on different approaches to the problem, described in detail in Chapter 3. The general distinction of the approaches is based on deciding how to manage cost dependency in the presented problem.

Algorithms taken into consideration as a possible optimal solution to the problem were: Ant Colony Optimization, Nearest Neighbour, and God. They are detailed in Chapter 4.

To have a possibility of analysis and investigation, research environment was developed. Implementation of this program, named Traveling Eco-Salesman fulfills engineering requirements of the Master Thesis. Used technologies and implementation of TES (Traveling Eco-Salesman software) are described in Chapter 5.

To highlight a practical aspect of the Thesis, user’s manual for the developed software is given in Chapter 6.

The investigation presented in Chapter 7. was made in fields of the value of goal function of the problem, length of the found path and time required to receive a result of running the algorithms. The research was made on a diversified dataset, which ensures that the investigation was meaningful. What is more, Chapter 7. contains also an investigation of the most optimal cost management approach.

Conclusion contained in Chapter 8. summarize obtained results. What is more, the most optimal solution to the problem is proposed and final thoughts of the author are incuded.

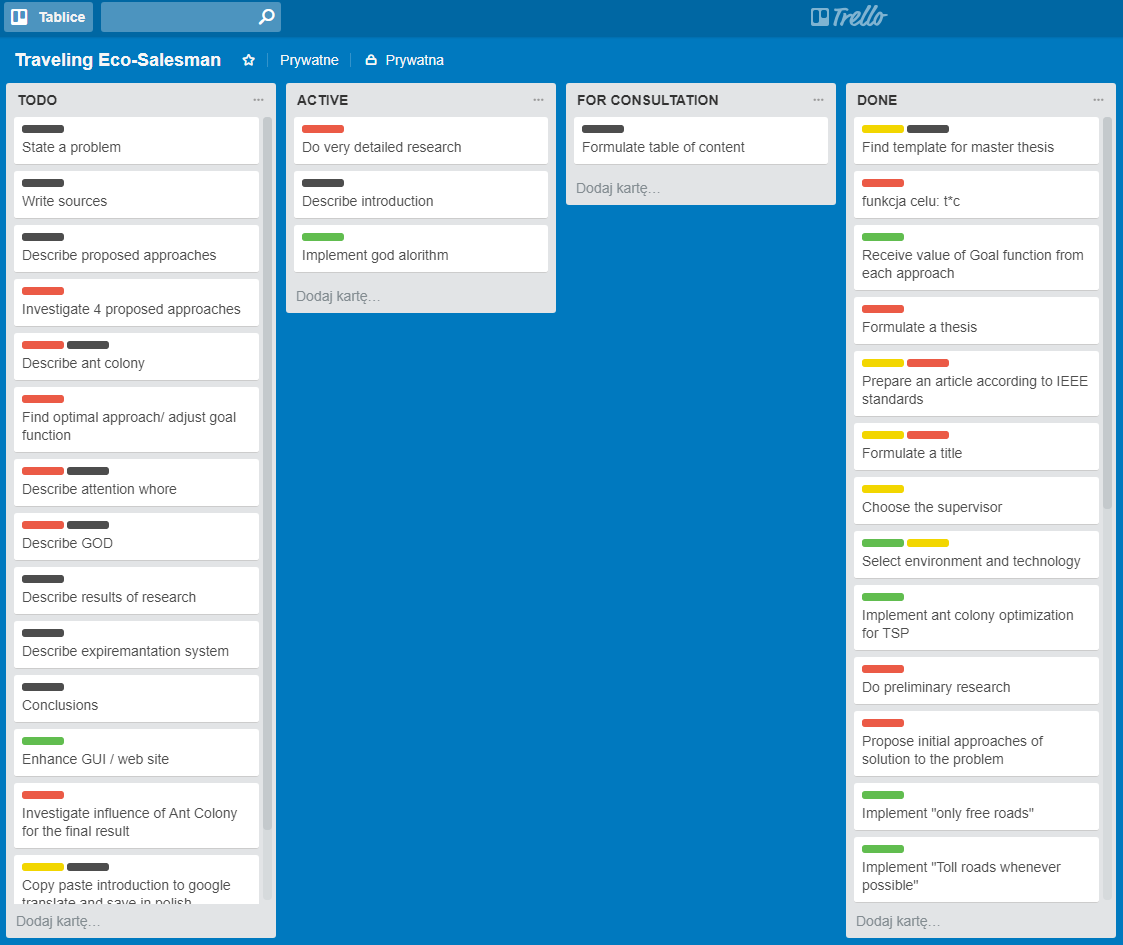
## 1.3 Workflow management

First work used later in TES program was done in spring 2017 for the purpose of Research Skill and Methodologies I course from Advanced Informatics and Control field of study. It was the implementation of Ant Colony Optimization algorithm for solving TSP problem.

The approach to software development used in the TES program is Agile, Scrum. The general idea of Scrum approach is to divide work into small, independent tasks. Then define the single unit of time, named sprint. Each sprint is built in three phases: planning, execution, and summary.

In case of TES program, length of the sprint was one week. Initial three sprints were mostly used for estimation velocity of the development, measured in units of work. From this point, developer knew exactly how much work he can perform and could estimate the size of the program and formulate a research plan.

Framework helping in the organization of work is Trello. Trello provides interactive dashboards used for workflow management. Dashboard used for TES is visible in Figure 1.



**Figure 1.** Trello dashboard

Figure 1. shows Trello Dashboard used in the work planning and management. It is logically divided.

There are specified in four columns:

* TODO: tasks which need to be done
* ACTIVE: currently active tasks
* FOR CONSULTATION: tasks which require consultation with the supervisor
* DONE: finished tasks

What is more, colorful labels mark tasks and split them into few categories:

* Green – engineering work (code)
* Red – research work
* Yellow – formal requirements
* Black – parts of Master Thesis

Such division and distinction highly improve visibility and readability and in result increase productivity.

Source control system used in TES is GitHub. GitHub provides tools for management and synchronization of the code.

Address of repository of the TES is given below:

<https://github.com/AdrianStrugala/Traveling-Eco-Salesman>

# 2. Problem statement

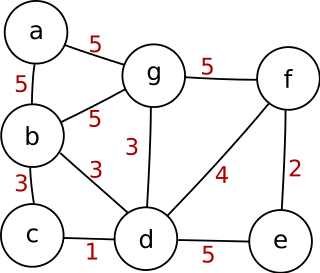
## 2.1 Traveling Salesman Problem overview

Traveling Salesman Problem (TSP) is an NP-hard optimization task. The goal is to find the shortest route connecting a set of *n* cities. There is given starting and ending city. Each city has to be visited exactly once [3][4].

The problem was formulated for the first time by Karl Menger in the 1930s [5] and considered for the first time as a solution of the real-life problem in the purpose of finding the shortest route for the school bus by Merrill Flood [5].

However, origins of the problem may be found in the early 1800s when Irish and British mathematicians were formulating graph theory [6]. A solution of TSP fulfills the requirements of finding a minimal Hamiltonian cycle in the weighted graph.

Set of cities is represented as a weighted graph *G(V,E)*, analogical to the example in Figure 2*.*



**Figure 2.** Example of weighted graph

Vertices *(V)* are the representation of cities:

|  |  |
| --- | --- |
|  | (1) |

A number of cities are equal to n.

Edges *(E)* are a representation of cost for traveling connected cities. Cost of the travel may be measured in time of travel, length of road, cost of the trip or combination of those factors. A number of edges is less or equal to *n2*.

Considering an example from Figure 2, where weights of the edges are times of travel. A hypothetical traveler wants to get from city A to city E. He has various of possibilities. The optimal solution to the problem would be a sequence of journeys A-G-F-E because the overall time of the travel is 12 and this is the lowest possible value.

## 2.2 Formulation of TSP Time and Cost dependent Problem

To answer a question when toll road is worth to choose, a correlation between the cost of the road and time of traveling this road had to be formulated.

This correlation is a statement of Goal Factor *(GP)* for a single path:

|  |  |
| --- | --- |
|  | (2) |

Where *GP* is the value of Goal Factor, *C* is the cost of the travel, *T* is a time of the travel for a single path and *I* is practical importance factor based on the value of saved time.

Empirical meaning of the Goal Factor is to find a road connecting two cities with the possible minimal time and cost of the travel.

Following assumptions are made:

1. Time of traveling free road *(TF)*is always longer than traveling the toll road *(TT):*

|  |  |
| --- | --- |
|  | (3) |

1. Cost of traveling free road *(CF)* is always lower than traveling the toll road *(CT)*:

|  |  |
| --- | --- |
|  | (4) |

1. Cost of traveling free road is a cost of gasoline only *(CG).* Cost of gasoline is calculated as the multiplication of distance, average combustion and price of fuel:

|  |  |
| --- | --- |
|  | (5) |

1. Cost of traveling toll road is a cost of highway payment *(CP)* and gasoline *(CG)*. Combustion on the highway is 1.25 times higher than on the national road, because of higher average velocity:

|  |  |
| --- | --- |
|  | (6) |

The final Goal Function is the minimal sum of Goal Factors for single paths. This is the key function of the future research:

|  |  |
| --- | --- |
|  | (7) |

Empirical meaning of Goal Function is to find the route connecting all the cities, according to TSP conventions, for which the overall time and cost of travel is the lowest.

This extension was inspired by [7].

# 3. Different approaches to the cost management

In the following chapter, four different approaches to managing cost in Traveling Eco-Salesman Problem are described. Each of them was examined in the same dataset. Results of this examination and conclusion are stated in the Chapter 7.

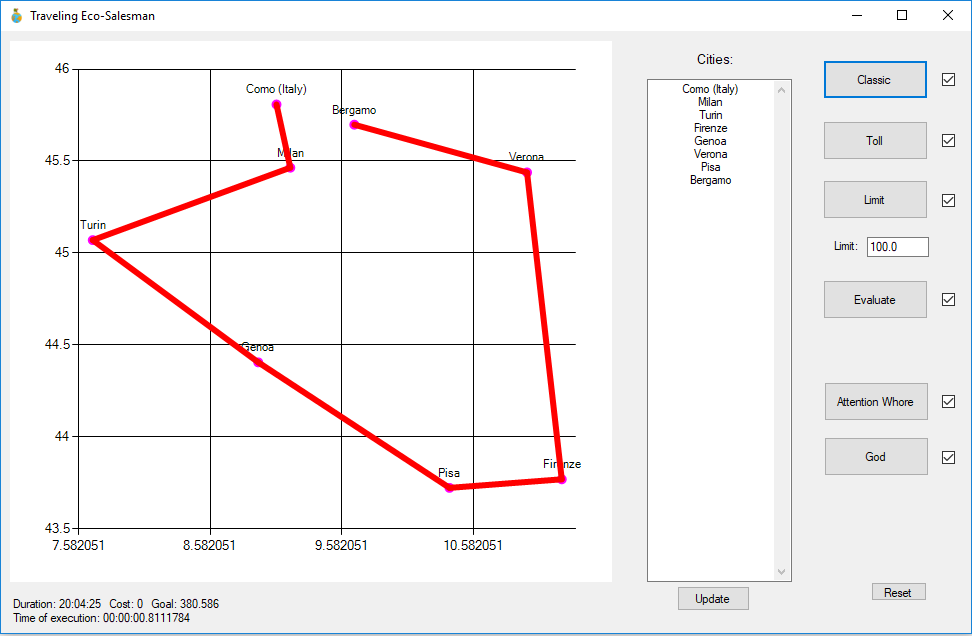
## 3.1 Only free roads

The first scenario taken into consideration as an optimal solution is to avoid toll roads completely. This is the simplest corner case and from the intuition, it cannot be optimal. If it was, nobody would use toll roads. Nevertheless, to be sure that research was meaningful and complete, this case was also taken into consideration. In fact, this is classical Traveling Salesman Problem.

Importance parameter from equation (2) is equal to 1, for every part of the road, when free option is chosen:

|  |  |
| --- | --- |
|  | (8) |

The result of choosing classic TSP approach is shown in Figure 3.



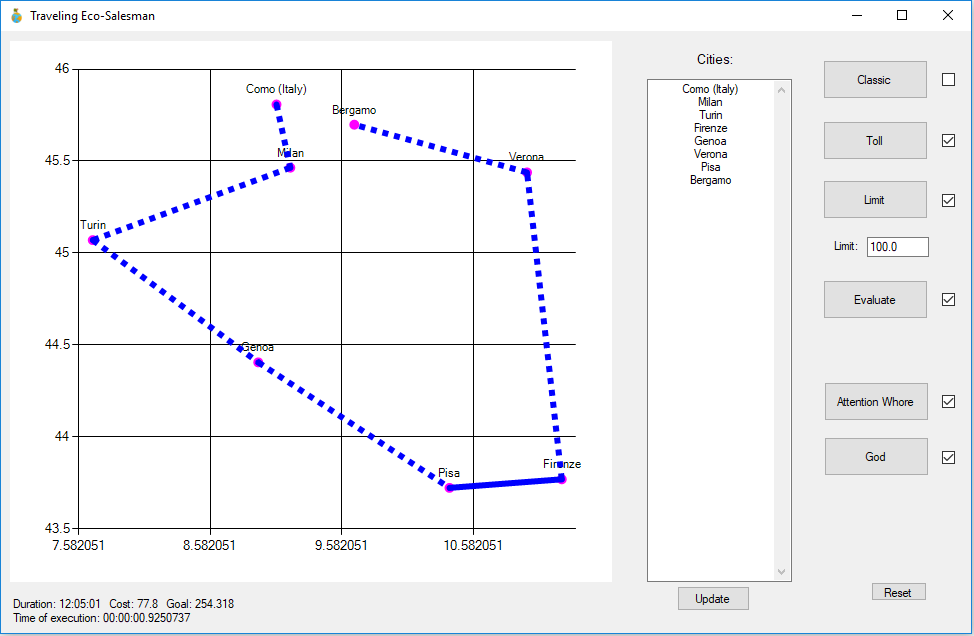
**Figure 3.** The result of choosing classic TSP approach

## 3.2 Toll roads whenever possible

The second idea is to investigate opposite corner case scenario if choosing toll road is always profitable. This case sounds like a promising solution to the problem.

*I* parameter from equation (2) is calculated as follows, whenever toll part of the road is chosen:

|  |  |
| --- | --- |
|  | (9) |



**Figure 4.** The result of choosing the only toll roads approach

As presented in Figure 4, time of travel and value of Goal Function have much lower value than in approach described as the first case.

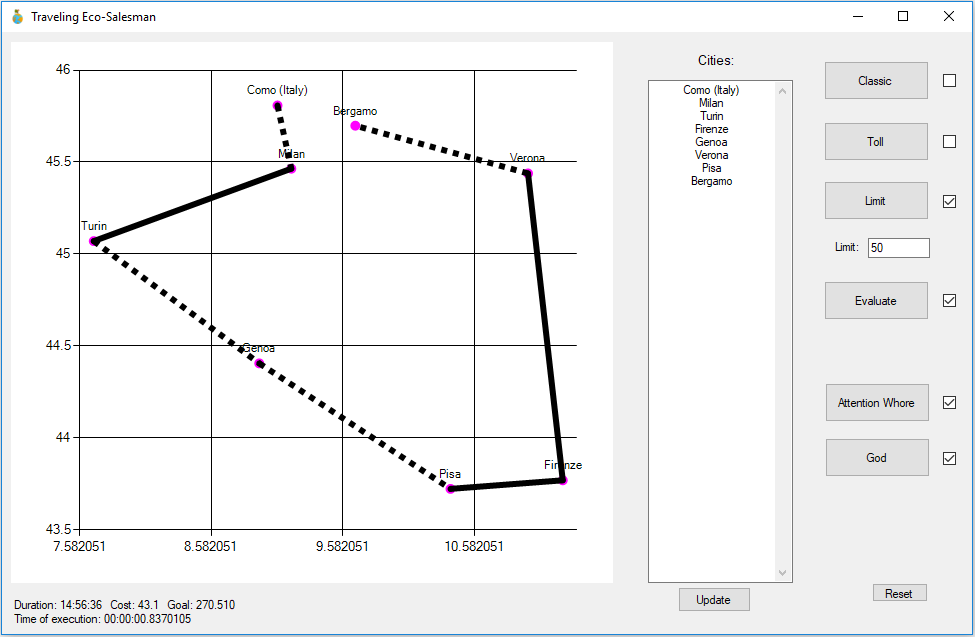
What information is shown more: for some of the parts of the roads it is not possible to use the toll road, simply because there is no highway connecting some of the cities.

## 3.3 Limit cost

The third idea is to introduce additional constraint of limiting overall cost of travel. This is the most flexible approach and the most practical for future users.

|  |  |
| --- | --- |
|  | (10) |

This scenario goal is to prioritize section of roads with the highest gain over those the least profitable. The Goal Function is strongly dependent on financial capabilities of the user. This is the most promising approach for future commercial development.



**Figure 5.** The result of choosing the approach with a cost limit

As shown in Figure 5. cost of the travel was reduced to a value lower than set upper boundary. On the other hand, duration of the travel and value of Goal Function increased in comparison to the previous approach.

In Table 1. results of the exaction of the TSP with the limit engine are collected. Based on those results influence of limit parameter *(L)* on the value of Goal Function is clearly visible in Figure 6.

**Table 1**. Influence of parameter L on the value of Goal Function

|  |  |  |
| --- | --- | --- |
| **Limit [€]** | **Cost [€]** | **Goal [€ × s]** |
| 0 | 0 | 380.586 |
| 7 | 3.5 | 388.3 |
| 14 | 12.2 | 385.553 |
| 21 | 12.2 | 385.553 |
| 28 | 27.9 | 313.413 |
| 35 | 27.9 | 313.413 |
| 42 | 27.9 | 313.413 |
| 49 | 43.1 | 270.51 |
| 56 | 43.1 | 270.51 |
| 63 | 61.2 | 292.605 |
| 70 | 61.2 | 292.605 |
| 77 | 61.2 | 292.605 |
| 84 | 77.8 | 254.318 |

**Figure 6.** Chart showing the influence of parameter L to value of Goal Function

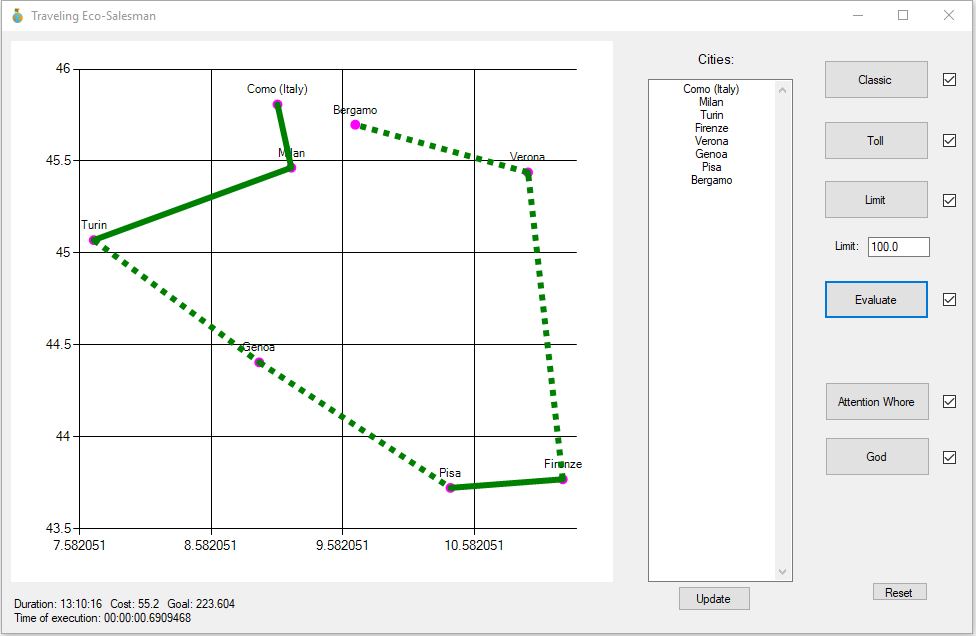
The general trend is, that with increasing limit of cost, the value of Goal Function decreases. For some intervals of the cost, the value of Goal Function stays on the same level.

The upper limit of research range was the value of cost received in the approach of using toll roads whenever possible.

## 3.4 Evaluation of the cost of time

The fourth and final idea is to take into consideration the relationship between cost and time parameters calculated for the Goal Factors. Then, for every part of the road chose separately if it is worth to use toll or free road.

During execution of the TSP engine, the algorithm builds path based on this evaluation of the cost of time.



**Figure 7.** The result of choosing the approach for automated evaluation of the cost of time

As shown in Figure 7, this approach minimizes the value of Goal Function the most. Detailed analysis is carried out in Chapter 7.

# 4. Algorithms

The conclusion from the previous chapter states that evaluation of the cost of time performed by TES gives the most promising results, minimizes the Goal Function the most.

Another field of research that has to be taken into consideration is if ACO provides an optimal solution to the problem. In purpose of comparison, two other TSP solving algorithms were implemented.

The research was performed on the various dataset. Because of properties of metaheuristics, execution of each algorithm was performed 30 times, data in the table contains mean values of the executions.

## 4.1 Ant Colony Optimization

The idea of Ant Colony Optimization algorithm for solving Traveling Salesman Problem comes from real life observations [8].

Ants living in ant colony are constantly looking for food. There are leaving hive and travel to the closest food source. At first, they search randomly. Each ant leaves pheromones trail on the road he passed. Later, ants traveling for food are heading the way indicated by pheromones.

As time passes pheromones are evaporating. The strongest trail is left on the path connecting directly hive and food source because it is used the most often and by the biggest number of ants.

Pseudocode of Ant Colony Optimization is shown in Figure 8.

**Input:** distance matrix, no. of ants, no. of cities

**Output:** shortest route

1: current shortest route = initial solution;

2: while (ant can chose different path than current shortest)

3: while (ant < no. of ants)

4: while (no. of cities in current path != no. of cities)

5: select city i with probability based on distance and pheromone matrices;

6: add city i to shortest path for this ant;

7: leave pheromone trail;

8: shortest path in this iteration = compare paths for each ant and take the shortest;

9: if (shortest path in this iteration < current shortest path)

10: current shortest path = minimum path in this iteration;

11: evaporate pheromone trials;

12: shortest route = current shortest path;

**Figure 8.** Ant Colony Optimization pseudocode

Ants are released from the hive. Their goal is to solve Traveling Salesman Problem. According to a real-life example, at first, they are choosing the route randomly and leaving pheromones. The strongest pheromone trail is left on the most commonly used route, which is considered as the optimal path.

## 4.2 Nearest Neighbour

Nearest Neighbour was one of the first solutions to TSP problem [13]. The idea is very straightforward. Starting from the initial city, the program chooses the nearest city of already considered. Then adds it to the path and marks as visited. City, which is set as an ending point, is simply added as the last city in the path.

Pseudocode of Nearest Neighbour is shown in Figure 9.

**Input:** distance matrix, no. of cities

**Output:** shortest route

1: current shortest route = initial solution;

2: add starting city as the first;

2: for (no. of cities -2)

3: find nearest neighbour (exclude already visited);

4: add nearest neighbour to path;

5: mark nearest neighbour as visited;

6: add ending city as the last;

7: shortest route = found path;

**Figure 9.** Nearest Neighbour pseudocode

## 4.3 God

God algorithm comes from philosophy and religion, to be more specific, from human considerations of the beginning of the word and life [14]. One of the theories says that at beginning of everything God created our universe. In this theory, God created an infinite number of universes, with random laws of physics, rules of chemistry and other rules governing the world. Universes were collapsing one after another, and only one of them survived – with the optimal rules. This is our universe.

Pseudocode of the God algorithm is shown in Figure 10.

**Input:** distance matrix, no. of cities, no. of universes

**Output:** shortest route

1: current shortest route = initial solution;

2: parallel for (no. of universes)

3: draw random route

4: shortest route = find shortest route in random routes;

**Figure 10**. God pseudocode

Adjusting God to the TSP is simple. The program chooses random route connecting all the cities numerous times. It uses all the strengths of parallel programming – drawing random routes may occur in the same time. The only limit is a computational power of the computer.

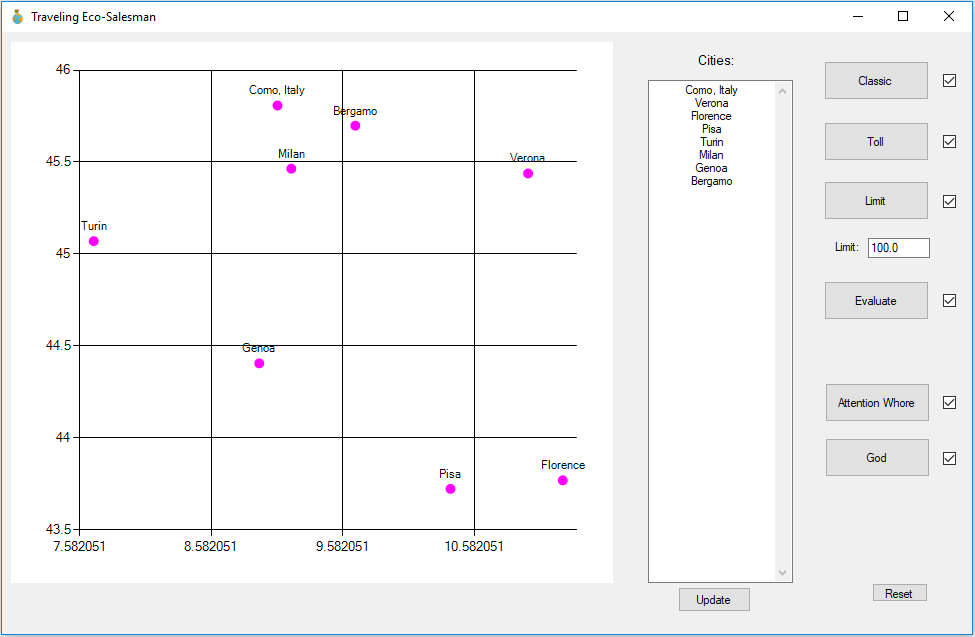
In the end, the shortest path is chosen between the random.

# 5. Research environment

## 5.1 Traveling Eco-Salesman

In purpose of the research, dedicated experimentation system was developed. The program is written in C# 7.0, Microsoft object-oriented programming language and is an example of the innovative approach to software development.

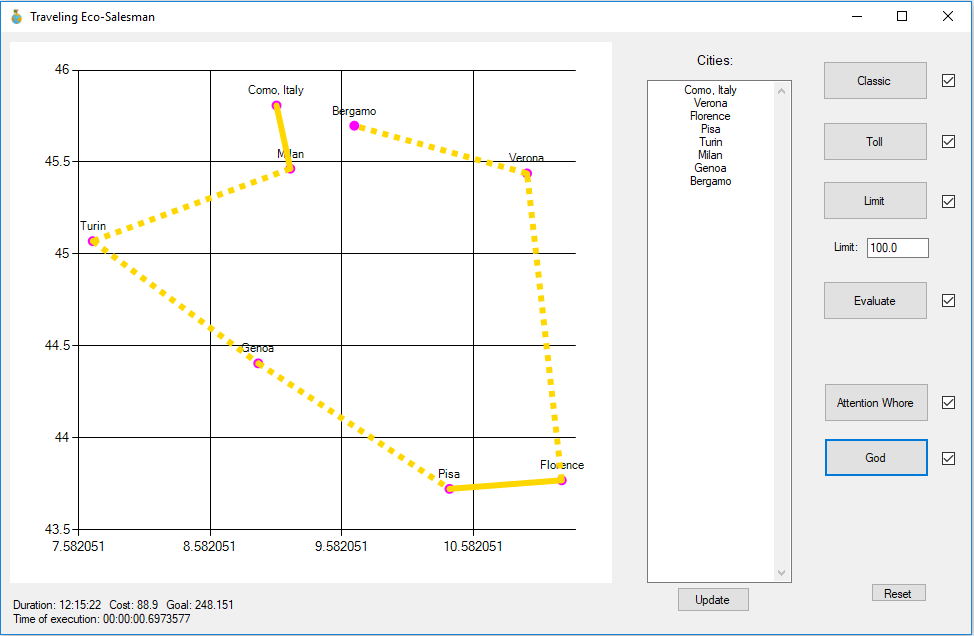
Development of the TES was done in the Visual Studio 2017 Community Edition working under operating system Windows 10. TES to work requires a stable internet connection.



**Figure 11.** Initial user interface

The flow of the application is as follows.

1. Application welcomes user with interface visible in Figure 11. The user has a possibility of specifying a list of the cities, which he or she wants to visit. The only reminder is to set starting city as the first on the list and the last city is finishing point.
2. The application connects to the external sources and downloads data of the distances and costs of travels between listed cities from external third-party sources.
3. The user chooses the option for solving Traveling Eco-Salesman problem, which fits his or her needs the most.
4. After execution of work, the application shows a screen similar to presented in Figure 12. Application displays route between cities, which specification if a toll or free way is preferable on every part of the route. The application gives information on duration of the travel and cost of toll fees. For test purposes additionally, the value of Goal Function and duration of execution of the TES algorithm are visible.



**Figure 12.** Example of use of TES

## 5.2 Clean Code approach

In purpose of keeping TES software easy to maintain and extend, the newest approach to software development was used. Rules of keeping the code simple, following SOLID design and implementing the newest technologies and ideas are generally known as Clean Code approach [10].

Few of the CC (Clean Code) features present in TES are described below:

5.2.1 Polymorphism

The most obvious field of future improvement and research is to implement another algorithm solving TSP problem. To ensure cohesion and keep code extensible, the idea of polymorphism was used.

General class for TSP Engine (part of the code solving TSP) is an abstract class named TSP. This class stores instances of data used in the program, to ensure that every TSP algorithm uses exactly the same values of data. Additionally, contains a definition of SolveTSP() method, to enforce inheriting classes implementation of TSP solver. General idea of implementation of polymorphism is shown in Listing 1.

abstract class TSP

{

protected static readonly IDistanceMatrix TollMatrix = DistanceMatrixForTollRoads.Instance;

protected static readonly IDistanceMatrix FreeMatrix = DistanceMatrixForFreeRoads.Instance;

protected static readonly IDistanceMatrix EvaluatedMatrix = DistanceMatrixEvaluated.Instance;

protected static readonly BestPath BestPath = BestPath.Instance;

protected static int NoOfCities = Cities.Instance.ListOfCities.Count;

public abstract void SolveTSP();

}

**Listing 1.** Implementation of polymorphism

A similar way of thinking accompanies part of the program solving Ant Colony Optimization based functionalities. One more time, one general abstract class AntColonyAbstract is inherited by four different approaches to implementation of Ant Colony Optimization. Abstraction tree of Ant Colony implementation is visible in Listing 2.

abstract class AntColonyAbstract : TSP

class AntColonyClassic : AntColonyAbstract

class AntColonyEvaluation : AntColonyAbstract

class AntColonyToll : AntColonyAbstract

class AntColonyWithLimit : AntColonyAbstract

class NearestNeighbour : TSP

class God : TSP

**Listing 2**. Abstraction tree of AntColony

5.2.2 Design patterns – Singleton

Design patterns are repeatable and common solutions to problems met by developers. One of the most popular design patterns is Singleton. As the name suggests, the unique feature of this code ensures developers, that only one instance of the class is used in code.

The purpose of Singleton is to be sure, that exactly the same data is used in every part of the code. An example in Listing 3. is Singleton of CostMatrix class. This class stores costs of travel coming from the external source. Costs have to be the same in every execution of the TSP engine algorithms (up to the moment of inserting a new list of cities).

What is more, below example is thread-safe Singleton. That means an instance of CostMatrix class can be used in parallel programming and security mechanism for breaking the execution of the program is implemented.

namespace TSPTimeCost.Singletons {

public sealed class CostMatrix {

public double[] Value { get; set; }

private static volatile CostMatrix \_instance;

private static readonly object SyncRoot = new object();

private CostMatrix() {}

public static CostMatrix Instance {

get {

if (\_instance == null) {

lock (SyncRoot) {

if (\_instance == null)

\_instance = new CostMatrix();

}

}

return \_instance;

}

}

}

}

**Listing 3.** Example of singleton implementation

5.2.3 Parallel programming

Because of a great number of operations, implementation of parallel programming significantly accelerates execution of the program [9].

In the Listing 4. InitiallizeAsync() function runs calculation of various matrices at the same time, without the need for waiting for ending of calculation of any of them.

public async Task InitializeAsync()

{

Task initializeSingletons = Task.Factory.StartNew(ProcessInputData.InitializeSingletons) ;

initializeSingletons.Wait();

NNait Task.Factory.StartNew(ProcessInputData.CalculateDistanceMatrixForFreeRoads);

NNait Task.Factory.StartNew(ProcessInputData.CalculateDistanceMatrixForTollRoads);

NNait Task.Factory.StartNew(ProcessInputData.CalculateDistanceMatrixEvaluated);

NNait Task.Factory.StartNew(ProcessInputData.CalculateCostMatrix);

}

**Listing 4.** Parallel call of five tasks

## 5.3 Google Maps API and Michelin Maps API connections

First connection required by the TES is to receive information from Google Maps API (Application Programming Interface) about geolocation of the inserted cities. Code of this call is visible in Listing 5. The request is sent as a standard HTTP Get Method to URL specified in the documentation [11]:

[https://maps.googleapis.com/maps/api/geocode/json?address={cityName}&key=KEY](https://maps.googleapis.com/maps/api/geocode/json?address=%7bcityName%7d&key=KEY)

The response comes back in JSON format, is deserialized and saved in the memory.

public static List<City> GetCitiesFromGoogleApi()

{

List<City> cities = new List<City>();

List<string> cityNames = ReadCities();

foreach (var cityName in cityNames)

{

City toAdd = new City { Name = cityName };

JObject locationJson = GetLocationJson(cityName);

toAdd.Latitude = locationJson["results"][0]["geometry"]["location"]["lat"].Value<double>();

toAdd.Longitude = locationJson["results"][0]["geometry"]["location"]["lng"].Value<double>();

cities.Add(toAdd);

}

return cities;

}

**Listing 5.** Get cities from Google Maps API call

The second call to Google Maps API is done in the purpose of populating matrices storing information about the time of the travels between each pair of the cities. In the Listing 6. is an example of the call for a single pair of cities, and the result is a time of travel between them, by free road.

private static int GetDurationBetweenTwoCitiesByFreeRoad(double originLan, double originLon, double destinationLan, double destinationLon)

{

string url =

$"https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial&origins={originLan},{originLon}&destinations={destinationLan},{destinationLon}&avoid=tolls&key=KEY";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

request.Method = "GET";

request.UserAgent = "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36";

request.AutomaticDecompression = DecompressionMethods.Deflate | DecompressionMethods.GZip;

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

string content;

using (Stream stream = response.GetResponseStream())

{

using (StreamReader sr = new StreamReader(stream))

{

content = sr.ReadToEnd();

}

}

var json = JObject.Parse(content);

return json["rows"][0]["elements"][0]["duration"]["value"].Value<int>();

}

**Listing 6.** Get duration of travel from Google Maps API call

Cost matrix, keeping information about cost of toll fee between each pair of the cities is filled by actual data downloaded from Michelin Maps API, ViaMichelin [12] as shown in the Listing 7. Response comes in XML format, is deserialized and saved into memory.

private static double GetCostBetweenTwoCities(City origin, City destination)

{

string url =

$"http://apir.viamichelin.com/apir/1/route.xml/fra?steps=1:e:{origin.Longitude}:{origin.Latitude};1:e:{destination.Longitude}:{destination.Latitude}&authkey=KEY";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);

request.Method = "GET";

request.UserAgent = "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36";

request.AutomaticDecompression = DecompressionMethods.Deflate | DecompressionMethods.GZip;

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

string content;

using (Stream stream = response.GetResponseStream())

{

using (StreamReader sr = new StreamReader(stream))

{

content = sr.ReadToEnd();

}

}

XmlDocument doc = new XmlDocument();

doc.LoadXml(content);

XmlNode node = doc.DocumentElement.SelectSingleNode("/response/iti/header/summaries/summary/tollCost/car");

double result = Convert.ToDouble(node.InnerText);

return result / 100;

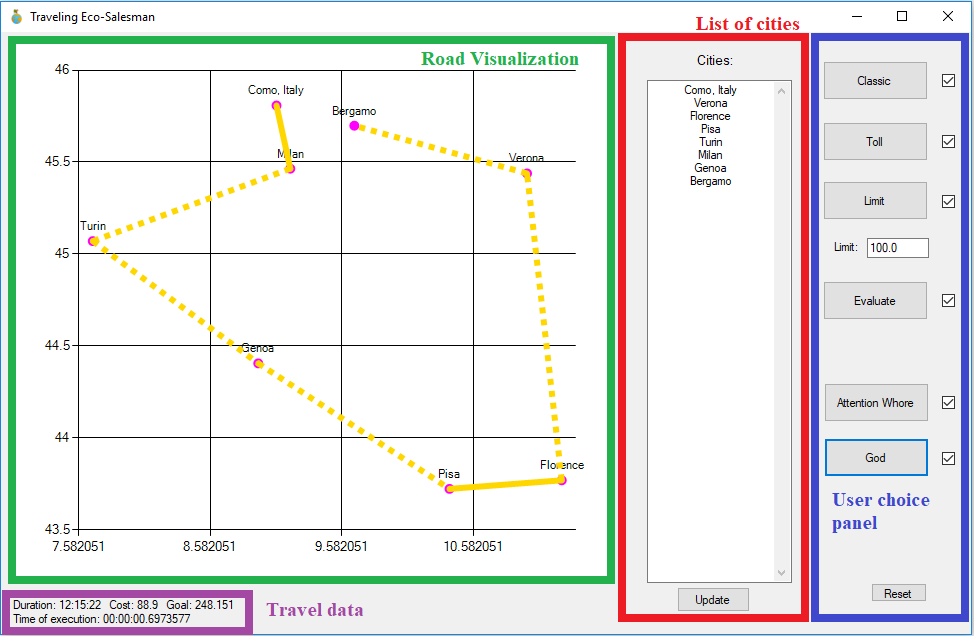
}

**Listing 7.** Get cost of travel from ViaMichelin API call

# 6. User’s manual

To have a possibility of utilize source code attached to the Master Thesis, the document contains short user’s manual of Traveling Eco-Salesman problem software.

The purpose of the program is to provide the most optimal route connecting given set of cities, according to the chosen TSP solution engine. List of cities and chosen TSP engine are information requested from user.



**Figure 13**. Visualization of TES interface

Interfaces of application shown in Figure 13. will be described in the sequence of interaction with the user:

1. List of cities – user applies a list of cities that are taken into consideration during the trip. To apply changes “Update” button must be clicked.
2. User choice panel – user specifies which TSP engine should be run.
   1. Classic – classic TSP including only free roads. Runs based on ACO algorithm
   2. Toll – program chooses toll road whenever possible. Runs based on ACO algorithm
   3. Limit – user specifies upper boundary of road fee (in euro currency) that he takes into consideration during travel planning. The limit is set in the text box below in form of a number. Runs based on ACO algorithm.
   4. Evaluation – program performs an evaluation of cost and time according to values of Goal Factors, then chooses the most optimal (which minimizes Goal Function the most). Runs based on ACO algorithm.
   5. Nearest Neighbour – program performs an evaluation of the cost and time. Runs based on Nearest Neighbour (NN) algorithm.
   6. God – program performs an evaluation of the cost and time. Runs based on God algorithm.
3. Road Visualization – cities specified in point 1) are drawn on the cartographic grid. After execution of TES engine, the colorful line connects the cities according to obtained result. The line may be drawn in two ways:
   1. Continuous – informs the user that free road should be chosen for this part of the road.
   2. Dotted – information to a user, that toll road should be chosen.
4. Travel data – after execution of TES engine information about the road is displayed. It contains four fields
   1. Duration – information about the length of the whole journey expressed in HH:MM:SS format.
   2. Cost – the cost of travel in euro.
   3. Goal – the value of Goal Function.
   4. Time of execution – time needed for the program to calculate and display information about the planned trip.

# 7. Research

## 7.1 research dataset

The research was done on three various datasets with unique features. The first was set of cities:

Como

Verona

Florence

Pisa

Turin

Milan

Bergamo

All the cities are located in northern Italy. This region is very well communicated and because of mountainous terrain, the time difference between traveling toll and the free road is significant. This set is named Italy for a shortcut.

The second is set of cities in Poland:

Wroclaw

Gdansk

Krakow

Poznan

Warsaw

Lublin

This region is not that well communicated, and toll fee is relatively high. This set is named Poland for a shortcut.

The last set of cities are locations all over Europe:

Moscow

Warsaw

Barcelona

London

Prague

Budapest

Paris

Amsterdam

Madrid

Oslo

Berlin

Vienna

Athens

Rome

Stockholm

Istanbul

Lisbon

Those cities are located in different countries and tests also connection between different regions in Europe. This set is named Europe for a shortcut.

## 7.2 Cost management analysis

Analysis of the results is done on the most optimal solution to the problem for each approach. Dataset used in this comparison is set of cities from northern Italy. Results are collected in Table 2.

**Table 2.** Comparison of results for different approaches to cost management

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Time of travel [hh:mm:ss]** | **Cost [€]** | **Goal [€ × s]** | **Time of execution [ms]** |
| **Free** | 20:04:25 | 0 | 380.586 | 816 |
| **Toll** | 12:05:01 | 77.8 | 254.318 | 933 |
| **Limit** | 12:05:01 | 77.8 | 254.318 | 610 |
| **Evaluation** | 13:10:16 | 55.2 | **223.604** | 692 |

Considering the value of Goal Function as a criterium of choosing the optimal solution to handling the cost of travel, evaluation of the cost of time is the most promising approach. Further research, deciding the most promising TSP engine were performed based on the very same mechanism of automatically choosing if travel between two cities should be done on toll or free road.

## 7.3 Experiment #1 – Italy

After deciding the most promising approach to the cost management, tests of different TSP engines have started. First one is performed on Italy dataset.

Remark: algorithm experiments were done on a different day, than the cost management experiment. Because of that, values of the received results are different. The program uses real-time data.

Values of 30 executions of the program for different TSP engines are shown in Table 3, Table 4 and Table 5.

**Table 3. God** implementation executed 30 times - results

|  |  |  |
| --- | --- | --- |
| **Time of travel [hh:mm:ss]** | **Cost [€]** | **Goal [€ × s]** |
| 13:04:58 | 74.00 | 295.24 |

**Table 4. Nearest Neighbour** implementation executed 30 times - results

|  |  |  |
| --- | --- | --- |
| **Time of travel [hh:mm:ss]** | **Cost [€]** | **Goal [€ × s]** |
| 16:10:43 | 70.60 | 436.76 |

Every execution of God and Nearest Neighbour algorithms provided the same result. For shortening, only one row on Tables 3. and 4. are included into the Thesis.

**Table 5. Ant Colony Optimization** implementation executed 30 times - results

|  |  |  |
| --- | --- | --- |
| **Time of travel [hh:mm:ss]** | **Cost [€]** | **Goal [€ × s]** |
| 14:06:17 | 69.90 | 349.06 |
| 13:04:58 | 74.00 | 295.24 |
| 13:04:58 | 74.00 | 295.24 |
| 14:06:17 | 69.90 | 349.06 |
| 13:04:58 | 74.00 | 295.24 |
| 14:06:17 | 69.90 | 349.06 |
| 14:06:17 | 69.90 | 349.06 |
| 14:06:17 | 69.90 | 349.06 |
| 13:04:58 | 74.00 | 295.24 |
| 13:04:58 | 74.00 | 295.24 |
| 14:06:17 | 69.90 | 349.06 |
| 14:44:25 | 61.13 | 411.93 |
| 13:04:58 | 74.00 | 295.24 |
| 13:04:58 | 74.00 | 295.24 |
| 14:06:17 | 69.90 | 349.06 |
| 13:04:58 | 74.00 | 295.24 |
| 13:04:58 | 74.00 | 295.24 |
| 14:44:25 | 61.13 | 411.93 |
| 14:06:17 | 69.90 | 349.06 |
| 14:06:17 | 69.90 | 349.06 |
| 13:04:58 | 74.00 | 295.24 |
| 13:04:58 | 74.00 | 295.24 |
| 13:04:58 | 74.00 | 295.24 |
| 14:06:17 | 69.90 | 349.06 |
| 14:06:17 | 69.90 | 349.06 |
| 14:44:25 | 61.13 | 411.93 |
| 13:04:58 | 74.00 | 295.24 |
| 14:06:17 | 69.90 | 349.06 |
| 13:04:58 | 74.00 | 295.24 |
| 14:44:25 | 61.13 | 411.93 |

The key parameter, informing about the performance of each engine is the value of Goal Function.

The first investigation was: how many occurrences of specific values of Goal Function are present in the results.

**Figure 14**. **God** – occurrences of different values of Goal Function

**Figure 15. ACO** – occurrences of different values of Goal Function

**Figure 16. NN** – occurrences of different values of Goal Function

Results of this investigation are a little bit surprised. Moving to the explanation.

Nearest Neighbour produces only one value of Goal Function. This is according to the property of this algorithm. It produces always the same result for the same input data.

Ant Colony Optimization is a metaheuristic algorithm, and this is a reason, why it produces various values of Goal Function. Nevertheless, this value can take only fixed quantities. The empirical explanation is as follows: the program doesn’t allow to travel on pathless tracks but uses only existing roads.

Even though God algorithm is built on the idea of randomness, it produces the same value of Goal Function for every execution of the program. This is because number of universes is high enough to draw the shortest route in every execution of the algorithm.

In theory, sufficient number of universes to ensure the occurrence of the optimal solution is given by equation (11): [14]

|  |  |
| --- | --- |
|  | (11) |

where:

U – the number of universes

n – the number of cities

From the practice – this number must be almost 10 times bigger.

**Table 6.** Comparison of mean results for different TSP engines for Italy dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Italy** | **Time of travel [hh:mm:ss]** | **Cost [€]** | **Goal [€ × s]** | **Time of execution [ms]** |
| **ACO** | 13:47:21 | 70.64 | 332.33 | 640 |
| **NN** | 16:10:43 | 70.60 | 436.76 | 0.01 |
| **God** | 13:04:58 | 74.00 | **295.24** | 693 |

Table 6. contains mean values of 30 executions of the program for various algorithms.

Another investigation was made of the field of increase of the time of travel. God algorithm, which produced the best result was set as a base. Values for ACO and NN algorithms are showing in Figure 17. increase in the value of God time of travel, in percentages.

**Figure 17.** The increase of values of time of travel for different algorithms

Mean time of travel according to ACO algorithm is higher by 5.4%, which is not a great value. Difference between results for God algorithm and the Nearest Neighbour is 23.66%, which is not acceptable for practical usage.

Because of simplicity of implementation, time of execution of NN engine is significantly shorter. This is a true statement for every performed experiment. Comparison of the times of execution is visualized in Figure 18.

**Figure 18.** Time of execution of the program for different algorithms

Time of execution of the different engines depends on various factors. For Ant Colony Optimization, the most impact has a number of ants. Similarly, for God algorithm change of the number of universes has the biggest impact.

Nevertheless, optimization of time of the execution of the program is a separate problem. It may be considered as an important issue in a future extension of the TES.

It doesn’t affect the result of the Goal Function and will be not discussed in more details.

Moving to the clue, the best TSP engine used in this experiment was God algorithm. Value of Goal Function is 12.5% lower than for the Ant Colony Optimization and 48% lower than for the Nearest Neighbour. Hence, time and cost of travel have also lower values.

## 7.4 Experiment #2 – Poland

Second and third experiments are not discussed that deeply as the first one. Their main goal is to confirm, that God algorithm produces the most optimal results using considered criterion.

The second experiment was done on the Poland dataset. Table 7. contains mean values of 30 executions of the program for various algorithms.

**Table 7**. Comparison of mean results for different TSP engines for Poland dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Poland** | **Time of travel [hh:mm:ss]** | **Cost [€]** | **Goal [€ × s]** | **Time of execution [ms]** |
| **ACO** | 22:38:57 | 10.71 | 680.57 | 403 |
| **NN** | 23:15:58 | 10.26 | 712.96 | 0.002 |
| **God** | 18:58:04 | 0 | **396.12** | 747 |

For Poland dataset, once more, the optimal algorithm was God. In this experiment value of Goal Function for God was 42% lower than for Ant Colony Optimization and 44% lower than for Nearest Neighbour.

## 7.5 Experiment #3 – Europe

The third experiment was done on the Europe dataset. Table 8. contains mean values of 30 executions of the program for various algorithms.

**Table 8.** Comparison of mean results for different TSP engines for Europe dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Europe** | **Duration [hh:mm:ss]** | **Cost [€]** | **Goal [€ × s]** | **Time of execution [ms]** |
| **ACO** | 227:19:36 | 166.92 | 34132.99 | 2523 |
| **NN** | 233::21:45 | 176.69 | 37021.90 | 0.01 |
| **God** | 218:20:44 | 115.58 | **22331.69** | 548 |

The third experiment performed on the list of cities placed in all over Europe confirmed, that God algorithm provides the optimal results between implemented TSP engines.

In this case, the result was also drastically better. Mean value of Goal Function for God was 35% lower than for ACO and 40% lower than for Nearest Neighbour.

# 8. Conclusion

## 8.1 Summary

The master thesis contains complete results of research performed for the purpose of finding the optimal solution to the Traveling Eco-Salesman Problem.

Traveling Salesman Problem and an extension including time and cost dependency was described.

The investigation was made in two phases. First exposed the most promising approach to cost management in the considered problem. Second, found the optimal TSP engine among considered. The research environment and chosen dataset were described in detail.

The optimal approach for cost management is an evaluation of the cost of the time performed by the program, based on values of Goal Factors defined by formula (2).

The optimal TSP engine implemented in the Traveling Eco-Salesman program, according to research described in the previous chapter, is God algorithm.

## 8.2 Comments

This extension to TSP problem provides a number of possible future investigation fields.

Firstly, the proposition of new formula for Goal Function expressed by equation (7). Goal presented in this article allows for finding profitable routes. Reformulation of Goal though may allow finding even more optimal paths.

Secondly, the proposition of another scenario for deciding how to manage the cost of the travel. Analogical to presented in Chapter 3.

Thirdly, implementing another algorithm for solving TSP problem. Another possible improvement in TSP engines is calibration of used parameters.

And finally, commercialize Traveling Eco-Salesman program. The most promising way is to propose it as an extension of existing Internet maps. It may also succeed as a standalone application or web site.

Another way of improvement and expanding could be turning TES program into real time driver assistant. Assistant could show nearest hotels, gas stations or suggest making a break during long travels.

Nevertheless, first job which is needed to be done is optimization of execution of the program. Connections to the external APIs are taking too much time to be comfortable for the real users. One of the ideas is to store some of the data in the database.

# Bibliography

1. Unknown author, *Clay tablet containing city map of Nippur,* Babylon, prob. 1400 BC
2. E. Balais, *The traveling salesman problem and its variations,* Springer, 2007
3. G. B. Dantzig, R. Fulkerson, S. Johnson, *Solution of a Large-Scale Travelling-Salesman,* Journal of the Operations Research Society of America, Vol. 2, No. 4, 393-410, 1954
4. M. Held, R. M. Karp, *The Travelling-Salesman Problem and Minimum Spanning Trees,* Operations Research, Vol. 18, No. 6, 1138-1162, 1970
5. E. L. Lawler, *The Traveling salesman problem: a guided tour of combinatorial optimization,* Wiley, 1985
6. N. L. Biggs, E. K. Lloyd, R. J. Wilson, *Graph Theory 1736-1936,* Oxford University Press, 1999
7. S. Qin, S. Liu, Q. Zhang, *The Traveling Salesman Problem with Profits Considering Time-Dependent Cost and Multi-Trip Mode,* Proc. 36th. Chinese Control Conference, pp. 2899-2903, 2017.
8. S. Oliveira, M. S. Hussin, A. Roli, M. Dorigo, T. Stützle, *Analysis of the population-based ant colony optimization algorithm for the TSP and the QAP,* 2017 IEEE Congress of Evolutionary Computation, pp. 1734-1741, 2017.
9. B. Albahari, J. Albahari, *C# 7.0 in a Nutshell,* O'Reilly Media, 2017
10. R. C. Martin, *Clean Code,* Prentice Hall, 2009
11. https://developers.google.com/maps/documentation/, 2017
12. http://dev.viamichelin.com/getting-started-rest.html, 2017
13. G. Gutin, A. Yeo and A. Zverovich, *Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP,* Discrete Applied Mathematics 117, pp. 81-86, 2002
14. A. Brüngger, A. Marzetta, K. Fukuda and J. Nievergelt, *The parallel search bench ZRAM and its applications*, Annals of Operations Research 90, pp. 45–63., 1999