Hochschule Rhein-Waal

Rhine-Waal University of Applied Sciences

Faculty of Communication and Environment

*Professor, Dr. Agatha Kalhoff*

*Professor, Dr. Mona Wappler*

# Application of the Ant Colony Algorithm for solving routing optimization problem in logistics

**Bachelor Thesis**

by

*Anna Pogrebnyak*

Hochschule Rhein-Waal

Rhine-Waal University of Applied Sciences

Faculty of Communication and Environment

*Professor, Dr. Agatha Kalhoff*

*Professor, Dr. Mona Wappler*

# Application of the Ant Colony Algorithm for solving routing optimization problem in logistics

A Thesis Submitted in

Partial Fulfillment of the

Requirements of the Degree of

Bachelor of Science

in

Mobility and Logistics

by

*Anna Pogrebnyak*

[redacted]

Matriculation Number:

30850

Submission Date:

12.04.2025

## Abstract

In logistics there are different options of how the Traveling Salesman Problem can be solved. For that purpose, the Ant Colony Optimization (ACO) may be used. In the thesis, the method is implemented in order to solve the routing optimization problem. In that case, the mathematical application is observed and analyzed. ACO parameters that affecting the working principles were researched in the thesis with the corresponding graphical representation. Based on the theoretical logic the Java language program is written for the purpose of realizing the ACO algorithm. In order to grade the effectiveness of the obtained results the additional model the Greedy method was applied. The developed program tool demonstrates the practical applications and a solution of the routing problem based on the real data in Wesel region. The thesis is focused on the route length optimization and cost reduction in terms of ACO implementation.

**Keywords**: Ant Colony Optimization, Traveling Salesman Problem, routing problem, cost reduction, route length optimization, Java software program

## Confidentiality Clause

The following Bachelor thesis contains no confidential data and information. All data used in this research are publicly available and have been sourced from publicly accessible repositories. The findings and conclusions presented in this thesis may be freely shared, published, and distributed.

The Bachelor thesis is made available to members of the Examination Board solely for the purpose of assessment. Beyond this purpose, the thesis may be made available to the public through the university's library or any other means deemed appropriate by the institution.

# Table of Contents

Abstract ..... i
Confidentiality Clause ..... ii
List of Abbreviation ..... v
List of Figures ..... vi
List of Tables ..... vii
1. Introduction ..... 1
1.1 The focus of the research ..... 1
1.2 Scientific gap of the thesis ..... 2
1.3 Framework of the provided further research ..... 3
2. Literature review ..... 4
2.1 Background information ..... 4
2.2 Modern concepts ..... 5
3. Theoretical part ..... 8
3.1 Fundamentals of the ACO algorithm ..... 8
3.2 Application of the mathematical model ..... 12
3.2.1 Transition probability ..... 12
3.2.2 Updating the level of pheromones ..... 13
3.2.3 Implementation of Elite Ant System ..... 14
3.2.4 Sorting algorithm implementation ..... 15
3.3 Greedy Algorithm ..... 16
3.4 Algorithms complexity ..... 17
4. Methodology ..... 19
4.1 Introduction ..... 19
4.2 Stages of the Applied Research ..... 19
4.3 Applied Research application example ..... 21
5. Practical part ..... 24
5.1 Stage 1: Definition ..... 24
5.2 Stage 2: Design and Planning ..... 26
5.3 Stage 3: Implementation ..... 30
5.3.1 Program classes ..... 31
5.3.2 Parameters configuration ..... 35
5.3.3 Code realization ..... 41

5.4 Stage 4: Results ..... 46
6. Conclusion ..... 49
6.1 Overall research findings ..... 49
6.2 Future plans and ideas ..... 50
7. References ..... 54
8. Appendices ..... 59
8.1 Parameters determination ..... 59
8.2 Java application code ..... 61
8.2.1 Main.java ..... 61
8.2.2 ShortRouteAlgorithm.java ..... 63
8.2.3 AlgorithmParameters.java ..... 63
8.2.4 Ant.java ..... 64
8.2.5 AntColonyOptimizationAlgorithm.java ..... 64
8.2.6 Input.java ..... 67
8.2.7 ElitistAntSystem.java ..... 68
9. Declaration of Authenticity ..... 69

## List of Abbreviation

| | |
|---|---|
| **ACO** | Ant Colony Optimization |
| **EAS** | Elite Ant System |
| **TSP** | Traveling Salesman Problem |
| **NNA** | Nearest Neighbor Algorithm |
| **GPS** | Global Positioning System |
| **AI** | Artificial Intelligence |

# List of Figures

Figure 1- The ant movement ..... 8
Figure 2 - Main steps of the ACO ..... 10
Figure 3 - Stages of the Greedy algorithm ..... 16
Figure 4 - The main stages of the Applied Research ..... 20
Figure 5 - The observed research area for a case study ..... 25
Figure 6 - Implemented code framework ..... 31
Figure 7 - Java classes flow ..... 34
Figure 8 - Alpha configuration ..... 37
Figure 9 - Beta configuration ..... 38
Figure 10 - Evaporation configuration ..... 38
Figure 11 - Q configuration ..... 39
Figure 12 - Ant number effect ..... 39
Figure 13 - Algorithm parameters class ..... 40

# List of Tables

Table 1 - Distance matrix, km ..... 27
Table 2 - Fuel tariffs, euro ..... 28
Table 3 - Fuel tariff matrix, euro ..... 29
Table 4 - Total fuel costs, euro ..... 29
Table 5 - Alpha results calculation ..... 37
Table 6 – Combined Java outputs ..... 47
Table 7 - Percentage difference in results ..... 48
Table 8 - Q results calculation ..... 59
Table 9 - Beta results calculation ..... 58
Table 10 - EAS results calculation ..... 58
Table 11 - Evaporation results calculation ..... 60

# 1. Introduction

## 1.1 The focus of the research

The modern world economy can be considered and personalized as a dynamic organism, which is rapidly evolving and continuously demanding improvements and innovations in it. One of the most significant systems of the world economy is the logistics sector. To confidently compete and maintain strong positions in the world markets, the construction of processes requires be adapted to the rapid solution of logistical issues (Christopher, 2016). In order to adjust to the new conditions in the modern world and to be able to adapt to a serious competitive environment, the logistics sector inevitably and urgently needs to modernize.

To ensure the efficiency of processes within the system, it is essential to solve and address issues that are related to source management within companies, develop predictive demand plans, optimize routes between cities and reduce operational costs. It is appropriate to apply unconventional methods to find effective approaches in order to achieve significant results.

The application of intelligent methods for solving logistics problems is one of the most promising approaches in the modern world. One of the perspective approaches is algorithms that are inspired by natural models, such as the Ant Colony Optimization (ACO) algorithm. This method is based on the biological process of ant's behavior and their aim to search for the shortest path by using the pheromone trails between the found food source and the anthill (Kumar et al., 2023). This feature contributes this method as an effective tool for solving problems across various areas of industries due to its accessible structure and functionalities, such as in the energy sector, robotics, telecommunications and logistics (Omid Bozorg-Haddad, 2017). Applying the Ant Colony method in logistics may cause a variety of objectives that can be achieved, including optimizing routes and determining the most cost-effective solutions.

The purpose of this thesis is to explore the application of the Ant Colony Optimization algorithm by solving the problem of route optimization between towns in the selected region and examining how the ACO algorithm can be adapted and applied in logistics. The study will include the analysis of the algorithm with the detailed fundamental principles and key elements of the Ant Colony method, the creation of the own example for the practical part. This example will involve the applied mathematical model of the

algorithm, developed program code and the algorithm will be tested based on the developed example.

The thesis will involve certain research questions such as: What are the particular main principles of the ACO algorithm? What kind of parameters and variables need to be considered and how they are affecting to the work of ant system? Will all these iterations lead to the optimization? How the ACO method be applied in a logistic related case study under incomplete conditions and what result will be?

To conclude a result, it is necessary to evaluate how the ACO algorithm can be used to solve the routing problem in logistics and how to interpret the obtained result. The main aim in this thesis is to analyze and evaluate the effectiveness by comparing the ACO algorithm and the Greedy method.

In order to objective obtain an assessment of the effectiveness of the use of ACO, it is necessary to select such an algorithm, which will be act as a universal framework. It means to which the same conditions may be implemented in the simulation part of a given environment in a case study. For that purposes the Greedy algorithm is appropriate for such implementation criteria as it is widely spread.

That is why by choosing the Greedy Algorithm as a core for comparison with the ACO model, it will be possible to discover which classes handle better with the problem heuristic or metaheuristic optimization algorithms. The comparison can also be classified as a complex work, because the Greedy algorithm serves as a general for other models and principles of optimization. The main elements of it lie in techniques such as: the Prim and Kruskal algorithms, Dijkstra's algorithm and Huffman coding (Wu and Wang, 2018). It will be possible to compare the ACO method in a collective view with many other methods that are relied on the Greedy principle.

### 1.2 Scientific gap of the thesis

Firstly, it should be noted that in the most researches the main object there was the measurement of distances between vertices, where the ACO method was used to find out the shortest distance using its natural model. In the thesis the application of the model is more extended. It is used to find an optimized route where the fuel cost will be also calculated. This relationship will reveal whether the method can discover the best route and reduce fuel costs at the same time. The research itself will be based on the real environmental existing data in the Wesel region.

The practical part will demonstrate how exactly the method helps to solve logistical problems. The model is validated by using own developed software. The research at this point considers data of distance and fuel cost under conditions of uncertainty. The usage of the addition consideration such as factor of fuel cost is an essential factor for considering an economic component that has not been used explicitly in other studies or experimental processes.

In the presented research the method is created and considered without taking into account the factors that are affecting the real traffic situation. At the initial stage it is necessary to understand how effective the ACO method manages with the construction of routes between towns. Further use of the program can take into account other conditions and constraints that may affect cost, distance, transport volumes, delivery time intervals and other related variables.

The created complex solution in the form of software can be integrated into the work of logistics companies by optimizing routes and reducing costs. If that point and topic be further developed with the implementation of the ACO method, this research paper could be served as a basis for the developing of more flexible models, which together conduct a universal tool.

### 1.3 Framework of the provided further research

One idea that could be further integrated into the future research is the use of GPS system in the calculations. A real-life picture of what is happening is a valuable tool for making quick decisions. Additional tools as open data from maps on traffic, terrain and weather conditions can also be applied in the research. This will help to form a model to review a more detailed view, which will be based on cross-checking parameters that can ultimately improve accuracy of the model.

Integration of the modern artificial intelligence (AI) technologies should also be noted for the further usage. It might be especially helpful for models to adapt conditions and therefore require progression in the whole system. The use of additional sources of information and dynamic data will only strengthen the results of the model and guarantee the quality and accuracy of calculations to a fundamentally new level. The model becomes more competitive and can assure a stable and sustainable solution for practical use.

The are some of directions on exactly how the Ant Colony algorithm can be further modified. However, a basic understanding of it is an integral part of the research. This thesis operates as a basis model for the further formation of optimization proposals.

# 2. Literature review

## 2.1 Background information

A major challenge in logistics is the process of finding and optimizing routes between or within the cities. In the context of the study, Christopher (2016) proposes that globally the problem is made more complex by the increased transportation time. The defined problem to work with is the optimization of the delivery travel distances and time in the cycle of logistic processes.

Finding an appropriate route for further transportation of goods between the corresponding delivery area depends on multiple factors. According to Goel and Gruhn (2008) optimization algorithms can consider in their work the delivery time, the length between the cities and the total cost for that transportation. It is possible in addition to take into account depreciation costs of transport, load capacity of vehicles and size of it. The paper emphasizes that the main task in route optimization is to find the shortest route between the corresponding points and optimize the delivery along the entire route list.

In the field of studying algorithms for optimizing transportation fees, one of the most popular and frequently touched upon topics is the solution of the Traveling Salesman Problem (TSP). The majority of solving the state of problem is to visit each point on the route at least one time. Then, going through the most optimal sequence of visiting these vertices and return to the initial position. If the problem is solved, an optimal route for the current study area has been found.

The choice of selecting an appropriate method for solving the Traveling Salesman Problem can be divided into several classes according to its specific characteristics. To solve the problem in the classical version of TSP consideration, algorithms are divided into groups such as exact and heuristic methods (Roberto et al., 2020).

Authors Zarei, Yousefikhoshbakht and Khorram (2016) narrate that exact algorithms indicate conclusive results in TSP. However, the optimality of the data depends on the number vertices and ensures the obtained result only for those where the variety of nodes are minor. When the number increases, the calculations become inefficient.

Toloo, Talatahari and Rahimi, (2022) outline that heuristic method in a short period of time finds the optimal solution. The method deteriorates the accuracy of the results in order to achieve fast calculations, making them approximate but still accurate. The representation of this approach is the Greedy algorithm. The principle is based on spreading the initial data during each step of movement, providing fast calculation speed and ensure the simplicity of its application. (Wang, 2023)

### 2.2 Modern concepts

In recent years research papers categorize another group of algorithms for solving TSP as metaheuristic methods. Pop (2012) notes that in finding the optimal solution, more complex strategies are applied to achieve the objectives in optimization in different fields and areas of activity.

Yang (2014) notes the most widely used methods in optimization are techniques that are based on the principles of natural calculation and biological behaviors, that are combined with the mathematical approaches. These methods can be adapted to a broad range of spheres or activities and can be used for modeling algorithms under various conditions and types of constraints with specified variables. Authors Zang, Zhang and Hapeshi (2010) explore a variety of algorithms that are related to nature-based phenomena. An essential feature of nature-based algorithms is that they are able to perform extremely proficiently results under undetermined and constantly changing environmental conditions. Therefore, these algorithms are most suitable for tasks in which synchronization with real time and adaptation to multiple emerging variables is required.

One example of metaheuristic methods is the Ant Colony method (ACO). ACO is one of the practical implementations in solving optimization problems arising by realization of the natural computing. The method was first published in a dissertation by Dorigo (1992). The work observes the biological behavior of ants in finding the shortest path between their habitat to the food they have found. They collectively explore paths using essence-pheromones to communicate and provide an information about the quickest routes for the least time consumption (Colorni et al., 1992). The given biological process forms the basis for the construction of the mechanism of the model algorithm, which allows solving complex optimization problems of logistic processes. Dorigo and Gambardella (1996) review the working principle of ACO, which consists of operation in generating virtual ant models that sequentially discover solutions based on pheromone trails and information about the distances between locations.

Birattari, Caro and Dorigo (2000) consider an approach framework for applying the natural model of ACO and propose a three-phase strategy for solving problems using an ant programming configuration. These stages are forming walks based on the role of pheromones, handling judgments and updating the data contained in the pheromones, connecting all pathways in a group of ants and updating the overall graph. The authors Dorigo and Stützle (2004) describe distinguishing features of ACO that are different from other methods. ACO algorithm is able to store memories of previous movements through pheromone trails and problem is solving due to constructed stepwise using virtual ants. These specifications characterize the method as unique and applicable in different areas of optimization process.

Selvi and Umarani (2010) highlight a number of advantages of ACO such as: the simultaneous operation of an ant's colony, the probability in finding the superlative and reverse route due to the accumulation of pheromone level, the ability to solve route optimization problems and flexibility to emerging changes in the path.

In their study Ma and Liu (2024) outline some of the challenges of ACO. Poor orientation to the customer experience, difficulties in selecting the optimal travelling speed and global search capabilities, the risk of falling into localized outliers. However, in order to work properly Han et al. (2023) describe the necessity of setting suitable parameters that are controlling the working actions in a whole ant colony system. Moreover, under what conditions the ACO method should be able to perform its operational efficiency in calculation and in searching the optimal options.

Solnon and Firm (2010) suggest using software simulation based on the mathematical model of the algorithm to solve difficult complex issue by using the ACO method and at the same time, to implement constraints in the form of time intervals. The introduction of such constraints was due to the fact that often in real conditions all processes occur according to a certain schedule and in order to dock movements between points. In that case, it is necessary to have an understanding of the mode of operation. Solving optimization problems allows companies to reduce costs and increase the quality of providing service. The authors Rizzoli et al (2007) also describe several case studies of real-world applications of ACO algorithm. The study centers on finding it's solution to the problem of delivering 52000 pallets to 6800 shops in Switzerland. In this case, the delivery time was limited to 20 days. As a result of solving the problem, an ACO model was created that allowed to reduce the number of routes between shops from 2056 to

1614, the total kilometrage from 147 km to 126 km, the goods loaded transportation increased from 77% to 98%. The results of applying the ACO model showed that it is efficient and able to find optimal routes to solve the problem in the shortest time. Ning Li and Yanliang Yu (2022) also suggest using an e-commerce-oriented version of the ACO model. The model was tested into the key related points in e-commerce logistics that are the optimization of delivery routes, size and weight of goods. ACO performed clear results with selected suitable for optimization parameters.

In a study Dang, Singh and Allen (2021) tested the integration of the ACO method for the company DHL. The company provided a simulation there the results were achieved for the delivery sections in North America. The ACO model showed an optimal result for mixed types of vehicles and their delivery with the coordination and helped to reduce the cost up to 5 million dollars per year.

To summarize in real-world examples Salam, Ahmed and Merugu (2013) note that application of ACO algorithm is possible under conditions of frequent dynamic change. In this case, the areas of application of ACO can be different and can be applied in different fields. Florez, Gomez and Lola Bautista (2013) used the method for a job scheduling planning and Bhavya and Elango (2023) optimized the performance of reservoirs and watershed management.

In recent years, academic researchers have focused on combining algorithms to find new efficient solutions. For example, authors Liu, Hu and Jiang (2023) proposed their research in improving the ACO algorithm by combining it with another method belonging to the metaheuristic class, with the Genetic algorithm. This proposed model realizes in the principle only the strengths of both algorithms. Jirawat Thongpiem et al. (2024) suggest variants of mixed combined methods, where modified algorithms are more efficient in dealing with providing the accuracy results of the issue solutions and are better in searching capabilities.

Despite the suggested improvement steps, there are still weaknesses in the implementation of combined hybrid methods. Tomar, Bansal and Singh (2024) indicate that a large number of research papers in the field of ACO where the mixed hybrid algorithm operation is considered, are still experimental. The assumed hypotheses in them should still be verified and tested. In the research, the authors emphasize that in order to obtain valid results with qualitative measures of problem-solving performance many experiment trials involving mixed methods and combinations of finding the shortest path

must be conducted. Moreover, additional checking and testing is still required to ensure that the outputs are valid and can be applied in practice.

# 3. Theoretical part

## 3.1 Fundamentals of the ACO algorithm

In the Literature review Chapter 2, attention was paid to the basic information regarding the Ant Colony Optimization algorithm. In this chapter, it is necessary to describe about the fundamental concepts and parameters that are involved in the process of realizing the operation of the method.

As stated earlier, the ant's behavioral model is based on a natural biological phenomenon. The ant's movement can be represented as a step sequential process attached in research of Dorigo and Gambardella (1996). The sequence of movement actions is shown in Figure 1, considering in more detail the natural prototype of an ant colony.

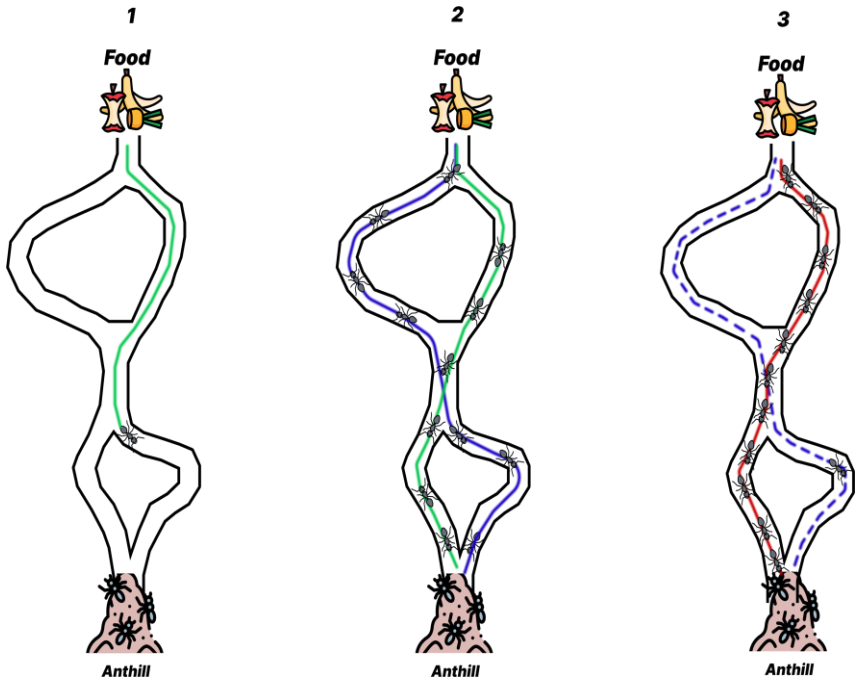


Figure 1- The ant movement

This figure shows how ants use pheromones to determine the fastest route from their location to a food source. Initially, the ant moves chaotically, following the smell of food flavor. At the begging of the work the trajectory of the ant's route is not defined. They start moving in the corresponding area independent to each over and have their own directions to the food source. This movement behavior can be indicated as illogical at first. However, ants are trying to search the food and are following to that aim.

Next step happens then the ants from the reached food source are trying to search backward path to their anthill. By returning back ant are leaving the pheromones on their paths. It serves as an indicator to other ants to start using the paths according to the pheromone trails. Over the time the pheromone level increases. Ants after the period of time and construction of several routes, give their privilege and prefer that path on which the concentration of pheromones is higher.

Furthermore, a larger proportion of the ants use the route that has accumulated the higher level of pheromones. Thus, demonstrated path has become the most effective among the other pathways. Over the time, the initial created routes are not attractive to the ant's choice and lose their importance due to the evaporation process on them. Then, the pheromone level accumulates on the more preferable paths and ants are following to the food based on the pheromone trails. As a result, the logic of ant behavior demonstrates how an entirely optimized route from home to food source can be formed.

Above steps imitates the main biological principles of the ACO working method and the logic of the ant`s movement is described. The following step by step rules are used in solving the tasks in searching of the optimal solutions in the observed area. However, only the principles were introduced above, at the same time the mail components and systematic methodology should be also described. In order to implement the ACO in finding the solution to a developed problem, there exists a gradual framework to which the realization should be attached. The flowchart is represented in the Figure 2.

For this purpose, during the processing of Figure 2, the flowchart was created based on the research study of Yue and Wang (2015). Based on the given illustration, it is necessary to explore the phases of the algorithm in detail and what parameters are required for the functioning.

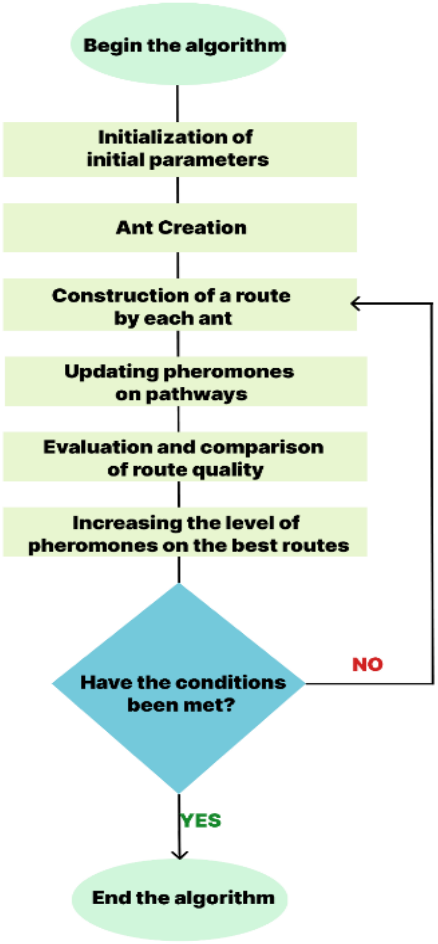


Figure 2 - Main steps of the ACO

The beginning phase of the ACO algorithm has been started to work.

Next, the initial parameters of the work were chosen. For the algorithm implementation it is necessary to determine the exact number of ants that are required in the system. However, the relevant data is also defined and in addition to the basic initial parameters the maximum number of iterations by the ant is selected.

Then, it is essential for the algorithm to create a digital ant colony that carries the determined chosen parameters. At the same time, virtual ants correspond to the algorithm with the natural behavior of real colony.

In the following stage, the created virtual ants start moving towards the goal based on the transition probability along all the possible paths. At this point, the first chaotic routes appear with the aim to reach the goal and return back to the initial points. Each designed route improves the previous one by reducing the distances between the average strong pheromone zones. That means after the first passes, ants create a map of intensity of pheromone trails and an overall idea of the total distance appears. Based on this logic

after several iterations, an approximate direction where the optimal route emerges, but this created option is still not the best solution.

Pheromone updating the process. The construction of each subsequent route by the ants implies the mechanism of leaving new pheromone traces, so that the concentration of which becomes more attractive to those ants who will make subsequent movements (Dorigo and Stützle, 2004). If the pheromone trail map is decoded, the more time movements have occurred over an area or point, the stronger the pheromone trail appears. At the same time, early points of pheromone concentration evaporate due to a decrease in the number of movements through them.

When the routes are created and the improvements are introduced to the system, the method of the ACO starts to evaluate the efficiency of the proposals. The overall grade is proceeded according to the minimum length of the distance or cost of travelling. A certain routes directions are determined by which the ant colony has to reach in the further next steps.

After a high number of passes through the vertices, the ant colony determines the most common route and disseminates this information to other ant colony members. The chosen optimal trajectory is expressed as an intense pheromone content over the route and the probability of crossings trough this path increases over the time.

An additional important step is the verification of fulfilled criteria. The process of constructing the optimal result continues until the moment when the required criteria is achieved. The multiple factors are checking whether the maximum number of iterations has been reached and whether the destination length or costs parameter is minimized. The solution is found to the problem and can be indicates as optimal to that problem and the best route is determined.

If the criteria are satisfied, the task can be considered as being completed. In case if any of the criteria is not achieved, the algorithm returns to the initial stage of the algorithm where each ant creates its own route. When all operations are completed, the algorithm demonstrates the created solution between the selected vertices and shows the best optional result.

## 3.2 Application of the mathematical model

The main steps and behavior of the ACO algorithm were introduced in Section 3.1.

In this chapter, the main mathematical formulas that are forming the ACO model need to be analyzed. Based on Colorni et al. (1992) and Wei (2014) the formulas and key parameters that are required for providing the ant method are taken below. Based on the mathematical approach the solution for own created case-study of the TSP problem will be tested and realized. Moreover, the program code will be relied on the provided formulas and will be written in Java language.

### 3.2.1 Transition probability

Formula 1 represents the probability of how ants are choosing a particular way from a given set of possible paths. In the most basic concepts and how it was described in the chapters above, the choice of selection depends on the accumulated trail of pheromones, informing the whole colony that a given path is preferred and it is essential to rearrange to that way.

The important point is to outline what are the parameters which are influencing to the ant behavior during the choice of appropriate route selection according to the research by Gaertner and Clark (2005). The parameter $\alpha$ – is a measure of pheromone relevance, which indicates how intensely ants will be orientated exclusively to the pheromone mark. The second parameter $\beta$ – is a representation of heuristic information that reflects to a preference in selection based on the path length.

If the value of $\alpha$ exceeds a particular value, then the ants completely trust the routes that other ants have previously been travelling. This feature causes pheromone concentrations to increase rapidly along all the length of the particular pathway, even if that path is far from optimal.

If the value of $\beta$ is larger, the higher the ants' preference for shorter routes appears. At the same time, if shorter routes have less pheromones than more popular routes it means that the shortest routes are still prioritized for ants. This adds variability and critical thinking in such cases where ants trust popular ones, and alternatively in other situations ants try to use less popular routes but shorter ones.

The relationship between the parameters $\alpha$ and $\beta$ is one of the key aspects during the process of the behavior of a group of ants in a visualized system. On the one hand, it takes into account the experience of building previous routes and as a result ants are trying to

find the most relevant solution. This symbiosis is aimed at finding the overall efficiency to the proposed problem.

$$P_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{a} \times [\delta_{ij}(t)]^{\beta}}{[\tau_{is}(t)]^{a} \times [\delta_{is}(t)]^{\beta}}, j \in allowed_k, \; s \in allowed_k \qquad 1)$$

The components in the formula are:

$P_{ij}^{k}(t)$ – ant transition probability $k$ into the chosen path at the moment *t* between the vertices $i \; and \; j$;

$\tau_{ij}(t)$ – value of pheromone intensity at the moment *t* on the point between the vertices $i \; and \; j$, the higher the value is, the more preferable that route is;

$\delta_{ij}(t)$ – the planned degree of heuristic data for vertices $i \; and \; j$, counted as the inverse of the path length, equal to $\frac{1}{d_{ij}}$, where *d* – length between the vertices $i \; and \; j$;

$\alpha$ – configurable pheromone importance indication used to find the assumed optimal route;

$\beta$ – an adjustable indication of the heuristic data used in the search of the estimated optimal route;

$j \in allowed_k$ – denotes the set of possible vertices to which ant *k* can be moved;

*s* – the index of summation of all possible vertices, where in the denominator $s \in allowed_k$ serves to normalize the probability.

### 3.2.2 Updating the level of pheromones

Formula 2 indicates how the pheromone level is updating at each interval between vertices $i \; and \; j$. After all virtual ants have completed a full cycle along all available paths in the system, the pheromone level is updated by evaporation factor. This phenomenon prevents unnecessary overaccumulation of pheromones on unsuccessful and inefficient pathways.

Also, by evaporation process it is possible to eliminate the possibility of the ACO algorithm reaching the local minimum without achievement to its goal. The new amount of pheromone is added to the better created routes and the probability of selecting these trails increases. Thus, a balance is accomplished between evaporation of old and addition of new accumulated amount pheromone, allowing the method to adjust to the process of changes in the routes.

$$\begin{cases} \tau_{ij}(t+n) = (1-\rho) \times \tau_{ij}(t) + \Delta\tau_{ij} \\ \Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}{}^{k} \end{cases} \quad 2)$$

The components in the formula are:

$\tau_{ij}(t+n)$ – pheromone parameter that is updated after n - number of steps;

$\rho$ – coefficient serves to be as an indicator of the pheromone evaporation and is equal to $0< \rho <10$;

$\Delta\tau_{ij}$ – value of the total amount of pheromone is added at the node between the vertices $i\ and\ j$;

$m$ – total number of ants appearing in the search for the optimal solution;

$\Delta\tau_{ij}{}^{k}$ – an indicator that represents the volume of pheromones from a specific ant $k$.

### 3.2.3 Implementation of Elite Ant System

According to Guo (2023) an improved model of Elite Ant System (EAS) can be applied in the work of the existing Ant Colony algorithm and be a part of the work of the ACO system. The author identifies a separate group of elite ants that attempts to strengthen pheromones along the shortest pathways. The group of elite ants serves as an additional indicator that redirects and suggests better routes to other participating ants, thus helping the ACO method to find optimal solutions more rapidly.

It should be considered that when a group of elite ants is being planned, their number should be taken into account. On the other hand, if there are too many of those ants, the system will loop on one route without visible opportunities to search for an optimal variant.

$$t_{ij}(t+n) = pt_{ij}(t) + \Delta t_{ij} + e\Delta t_{ij}^{bs} \quad 3)$$

The components in the formula are:

$t_{ij}(t+n)$ — new pheromone parameter that is updated after n - number of steps;

$pt_{ij}(t)$ — the number of pheromones reflecting with regard to the evaporation coefficient between the vertices $i\ and\ j$;

$\Delta t_{ij}$— total amount of pheromones added between the vertices $i\ and\ j$;

$e\Delta t_{ij}^{bs}$ — improving and adding the pheromones to the best trails through the actions of the elite ants, where e - is the coefficient denoting the increase in pheromones through the operation of the elite ants.

### 3.2.4 Sorting algorithm implementation

To achieve the best result, a pheromone sorting algorithm is being proposed to the existing ant colony system. Guo (2023) notes the importance of this action for the ACO. These actions are necessary for the competent distribution of pheromones during the searching corresponding pathways. If the method of sorting is neglected, ants may apply the pheromone trail evenly along the route avoiding the parameter of distances between the routes. In this case, inefficient paths can start to be more preferable to the ants in order to gain some pheromone level. As a result, the search for a solution may take a long time.

In that case, is necessary to implement the sorting method according to formula 4 in order to escape the inefficiency factor. The main idea of the method is so, that after each iteration the routes need to be sorted by length by choosing the shortest route. At the same time, elite ants, which have higher pheromone levels are attaching to these shortest routes. As a result, the number of pheromones that each ant leaves depends on the ant's position in the sorted list indicating as ω in the formula 4. A high position of an ant in a sorted list signaling that a large number of pheromones on the path is left by the ant.

In this process only quality paths are empowered and avoid reacting to non-optimal paths. That actions contribute to speed up the time of finding the better solution in order to the stop calculating and searching inefficient routes.

$$\Delta t_{ij}(t+n) = \sum_{k=1}^{\omega} {\Delta t_{ij}}^{k} \quad 4)$$

$${\Delta t_{ij}}^{k} = \begin{cases} \frac{(v-k)\times Q}{L_k}, & \text{if ant } k \text{ going through the nodes } i \text{ and } j \\ 0, & \text{otherwise,} \end{cases} \quad 5)$$

The components in the formula are:

$\Delta t_{ij}(t+n)$ – sum of the total number of pheromones after the n-th step between the vertices $i \; and \; j$;

$\omega$ – the order of the ant after being sorted;

${\Delta t_{ij}}^{k}$ – the amount of pheromone stored by a particular ant $k$ between the vertices $i \; and \; j$;

$Q$ – parameter indicating the value of the total amount of pheromone being left behind;

$v$ – the number of elite ants;

$L_k$ – the path length completed by the ant $k$.

Based on formula 5, the condition of adding pheromone in the sorting operations the algorithm depends on whether ant k moved between vertices *i* and *j*. If the ant has traveled between the corresponding nodes, then the pheromone level will be $\frac{(v-k)\times Q}{L_k}$. If the condition of the visited nodes by ant *k* is not satisfied, the pheromone level will be equal to 0.

### 3.3 Greedy Algorithm

As it was mentioned earlier in 1.2, the Greedy algorithm was chosen to be a method for comparison to the ACO algorithm. The Greedy model in most cases in the TSP problem is based on the use of the method NNA (Nearest Neighbor Algorithm) (Kizilateş and Nuriyeva, 2013). This method allows to calculate the route gradually, by starting the algorithm from the nearest point and continuing going to the closest locations, progressively extending it and as a result showing the completed route with all the passed stops.

A few basic and major steps of the Greedy algorithm for TSP can be identified based on the work of Ejim S. (2016). These stages are summarized in the scheme and presented in Figure 3.

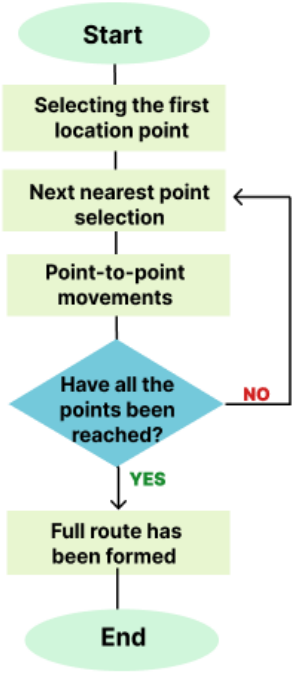


Figure 3 - Stages of the Greedy algorithm

Each element of the Figure 3 will be considered in more detail below, as it forms the algorithm process.

The algorithm starting to work and the first point has been selected. That means by searching for initial starting point any city or object of attraction can be chosen for those purposes. The calculation begins from this location and a zero index is assigned to it.

Next, in order to move further in a system, the algorithm should choose a suitable closest point. In a decision-making process algorithm outlines what city to select and follows the rule of movement. The principle is based on the length of the distance to which the following vertices is the closest. However, the chosen city must haven't been already traveled yet. By the transferring to the nearest city, it is essential to mark it as a visited point in order not to decide to go there twice. So, then add the corresponding index to the route sequence accordingly.

The above outlined actions are repeated in a cycle. The condition of selecting the unvisited closest location is obligatory. That means that by following steps 2 and 3, all points in a system must be reached by the algorithm by a certain moment.

The next stage is to confirm and verify if all points have been reached by the algorithm or not. If the answer is no and conditions have not achieved the goal, the algorithm returns to the stage of "Next nearest point selection". If the method has completed the work, the algorithm moves to its logical end.

After the time has expired, the algorithm returns to the starting point. This happened because of all the waypoints of the route have been identified during the specified period, the created path to the starting position is presented as a solution. Thus, at this stage the completed route is closing the algorithm, where all points have been visited and a closed cycle path has been created.

### 3.4 Algorithms complexity

Discovering the principles of algorithms, its features according to asymptotic complexity are important to consider. According to Mccool, Arch Robison and Reinders (2012) asymptotic complexity demonstrates the capabilities of an algorithm and shows the features based on only the mathematical models. It helps to show how algorithms may change over the time and how they act with increasing number of data set in their memory.

The time variation demonstrates how algorithms are responding and behaving to the resulting complexity created by initial data growth. The spatial complexity metric

responds to the increase in the memory size and reflects how the initial data growth affects to overall productivity performance. Mccool, Arch Robison and Reinders (2012) note that the complexity itself does not consider the constant coefficients, but focus exclusively on the asymptotic value of algorithm and how the algorithm operates due to its growth. To provide the measurement the Big-O notation is used, which describes the complexity level of algorithms by their performance through the time and space factors (Zhanar Bimurat et al., 2023).

According to Wei (2014) time complexity of ACO algorithm represents by $O(NC \times n^2 \times m)$, where:

$NC$ – the value indicating the number of iterations in the algorithm;

$n^2$– number of defined cities acting as vertices in the system. The measurement includes the start and end points and the corresponding distance between these points. Therefore, $n^2$ parameter denotes to the interaction between both vertices and also take into account the level of pheromone;

$m$ – the parameter indicates the number of ants.

As it follows from the formula of time complexity, the time depends on the number of iterations, city vertices and ants. A direct increase in one of the components leads to an increase in the total runtime of the algorithm ACO. This leads to the fact that for large amounts of data the ACO calculation time slows down.

Wei (2014) outlines that space complexity of ACO denotes by $O(n^2) + O(n \times m)$, where:

$O(n^2)$ – is responsible for storing the memory of distance data and level of pheromone trails;

$O(n \times m)$ – characterizes the memory in the algorithm of the travelled path which each ant performed.

Since the data are independent from each other and use the memory of the algorithm separately, so the indices need to be summarized. The space factor in the algorithm requires more memory as the data grows.

For the EAS subtype, the complexity factors will be similar to ACO, as EAS serves to be as a modernized version of the main method of ACO.

Regarding the Greedy algorithm Viswanathan, Sen & Chakraborty (2011) note time complexity in the form of $O(n^2)$ for a system processing data in a single node, where n - is the number of considered cities, denoting the node vertices. The algorithm time depends

on at which step the algorithm searches for the nearest city and for which number of cities. Comparing to ACO, it is quicker in data processing and faster in outputting solutions.

Space complexity according to Kossmann and Stocker (2000) in Greedy algorithm is defined as $O(n)$. The space is storing the sequence of visited cities in the algorithm memory. As the data grows, the memory enlarging according to the increase of the number of cities without sudden changes.

To summarize, based on asymptotic complexity, the following features of the algorithms can be distinguished. ACO model is more complex and requires more computation time and memory storage, but gives accurate calculation results. At that time, Greedy method is simpler and faster in performing computations, not requiring more capacity in calculations and in memory storage.

# 4. Methodology

## 4.1 Introduction

To perform the research, it is necessary to choose a concept, based on which the model will determine what certain actions should be used in order to realize the project. The concept of Applied Research was chosen as the basis for the realization of the research. It is a type of scientific approach focused on achieving result in applied problems and forming an integrated solution in different fields of activity. A systematic approach is determined for finding optimal solutions to existing problems by integrating theoretical knowledge into these results is the main objective of Applied Research. The solution itself relies on the development of experimental or simulation models that take into account the actual and environmental conditions (Bell, Harley & Bryman, 2022).

The main parameter for Applied Research is to gather and obtain qualified data (Mishra & Alok, 2022). Objectivity and conditions of the actual performance of the model depend on these data. Physical parameters should be reliable and accurate. In further stages, the data should be loaded into the models and empirically validated through experimental runs or simulations.

## 4.2 Stages of the Applied Research

Applied Research enables a process aimed at the practical application of theoretical scientific knowledge to solve specific particular problems. In order to realize research, several sequential steps have to be followed, consisting of two main groups of stages: Planning and Execution (Bickman and Rog, 2009).

The two groups have to be interpreted as a broad set of actions for implementing an Applied Research study. The scheme represents four main sequential stages, which are illustrated in Figure 4.

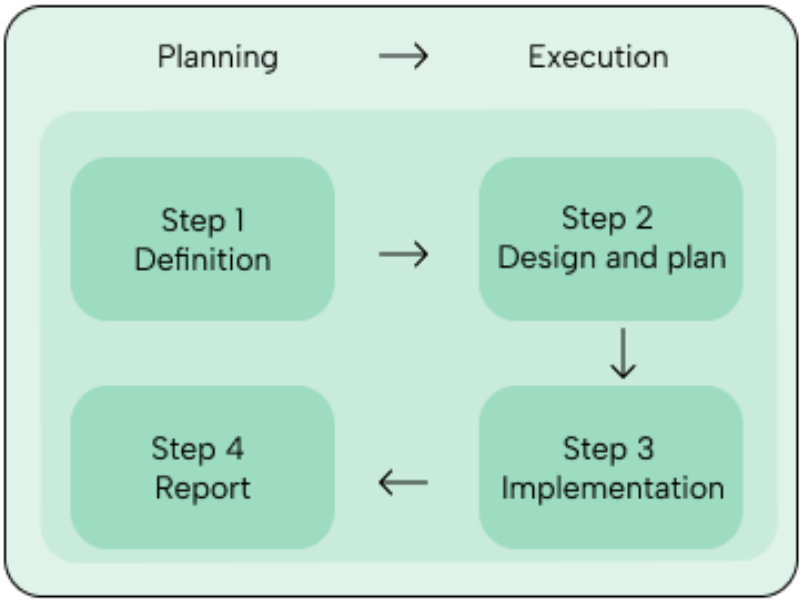


Figure 4 - The main stages of the Applied Research

Review each step of the process in more detail according to the work of Bickman and Rog (2009).

**Stage 1: Definition**

In the initial stage is – Definition. There the research problem is identified, so that the objectives are formulated and the goals are clarified for the further research work. The parameter should be allocated by considering the key conditions affecting the research process of implementation. The constraints of the environment should also be noted for the further analysis in the next steps.

**Stage 2: Design and Planning**

In the stage of design and planning, the main method needs to be determined for solving the identified set of tasks and issues. By defining tools, the experiment or simulation model should be used to conduct the process, to establish the order of data collection and variables that are involved in the calculations. The evaluation criteria need to be defined for analyzing the effectiveness of the proposed actions and achieved results, this point is also required for Stage 4 – Reporting the results.

**Stage 3: Implementation**

Theoretical knowledge should be combined at the stage of Implementation. It should be applied in the practical task of the problem to test the hypotheses and operated with the gathered data. By conducting experiments in a selected software, the most optimal and appropriate solution of the task will be indicated and simulated. After these iterations, the goal of the problem, that was determined in the Stage 1- Definition will be achieved.

**Stage 4: Report the results**

Stage reporting requires an analysis of the obtained results of the simulation that was provided at Stage 3 - Implementation. The analysis should cover the question of applicability of the proposed solutions of the problem by testing the hypotheses and evaluating the expected effect based on achieved results. The evaluation should be conducted in accordance with the effectiveness criteria of the proposed actions. Recommendations for the proposed implemented methods may be formed, and improvements for using the future models may be estimated. After these actions, a conclusion about accomplished work needs to be formulated.

The procedure of performing a consecutive regular operation in a process is the concept of the Applied Research methodology. These actions aim to integrate hypotheses based on scientific theoretical substantiation, by using software tools to analyze and search for the solutions to concrete specific problems.

### 4.3 Applied Research application example

This paper uses the Applied Research methodology based on the steps described in 1.2 and focusing on the key theoretical and mathematical models of the Ant Colony Optimization (ACO) algorithm. The combined scientific knowledge with a practical example will help to determine the adaptability of the ACO algorithm to the current conditions of the logistics system. Also, by implementing a simulation model, it will be possible to verify whether the ACO method can provide an optimal optimization route by considering and including fuel prices.

The main aim is to compare the ACO method as it is related to the Greedy Algorithm that belongs to the heuristic class. Selecting two different algorithms will enable to test which method finds the most optimal route based on its different approaches of solving the problem. It will be possible to evaluate how the ACO method works under uncertainty

conditions by comparing it with the algorithm from another classes. By creating a model simulation, the results will be gathered and the work of the method will be evaluated.

Following each step of the methodology and describing the exact stages in detail in order to structure the progress of the work and guarantee a consistent order in implementing the ACO research project.

**Stage 1: Definition**

The aim of the Applied Research is to solve technical problems in a practical way. Remler and Van (2014) note that the problem should be correctly formulated, and determine on what the new model needs to be developed or to adapt an existing one based on the created task. Further, considered the type of model which will be used to establish and describe the parameters that are needed during the work.

Therefore, in this paper the goal is to develop a task to test the ACO algorithm.

The practical application of the problem is formulated as follows: A company wants to optimize their delivery routes between towns in the Wesel District in order to minimize fuel costs and minimize the total traveled distance. The practical solution should be found for an optimal route that lies along all the towns in the region and returns back to the starting town. The example is planned to be solved under uncertainty and incomplete conditions, where no constraints are given in terms of delivery time and time-windows, information on traffic jams and other factors affecting the existing road traffic situation. An essential point is to consider on how ACO can improve route optimization and does it provide cost and delivery efficiency by finding an optimal path.

**Stage 2: Design and Planning**

The implementation of this stage will be carried out by using the measurement method. Using this method will provide to get the initial data and apply the set in the model. This method helps to gather information into an array of data for further work and evaluate them based on variable characteristics for further analysis in order to obtain results on the posed questions (Kabir SMS, 2016). The data will be collected through measurement tools in the context of this work by using auxiliary platforms.

To create an own case study, it is necessary to collect data on fuel tariffs and distances between towns, and then to create a matrix of routes and fuel costs between them. As this study is not associated with a specific company, data collection will be determined by Google Maps and Tankerkoenig platforms. Google Maps (2025) will gather enough

information about distances between towns. The accuracy of the maps is achieved by storing real time data using historical parameters (Mehta, Kanani and Lande, 2019).

The usage of Tankerkoenig (2025) platform will help to indicate and gather data of the current fuel tariffs in Germany. The developed case study in this research will be modeled based on the real existing conditions with updated data on fuel tariffs in Germany for the selected region.

**Stage 3: Implementation**

The simulation method was selected for its a systematic study of the object of the system in order to conduct the results of an experiment. The behavior of the model is based on the set of variables that simulate the process by its activity in the real circumstances of the environment (Banks, 2014).

The development of the ACO model's program code will be performed in Java language as a console application visualization. The code will be created based on an interpretation of the mathematical model of the ACO and contain strict parameters of the model construction.

The main components and criteria of the method should be considered by providing an experiment and simulate the ACO algorithm to find the optimal route to obtain the final results. A program code should be developed for Greedy algorithm for further interpretation of the reached results and providence of comparative analysis of two methods. By testing ACO and Greedy Algorithm, the program code will verify the general correctness of the provided data outputs and the accuracy of their work.

By implementing the experiment, into the ACO and Greedy models will be loaded the data information into the main class of the program. Algorithms will be located in different classes of the program, for better visual composition of the program code, and to implement the mathematical formulas by avoiding errors in the process of writing.

**Stage 4: Report the results**

At the stage of the reporting, achieved results of the simulation will be analyzed in terms of the effectiveness of the method and how the work was optimized in general. By providing an analysis of the outputs, results should be indicated in terms of the applicability of the ACO method in logistic sector for finding optimal route. Simulation results should demonstrate the suitability of the input data and indicate the effectiveness of the ACO method by contributing the comparison analysis.

The comparison method will be used to confirm the optimization hypothesis of this paper. Bolbakov, Sinitsyn and Tsvetkov (2020) note that comparison is a research method that has a set of analytical and intellectual procedures aimed at finding relationships or inconsistencies between the objects. The initial analysis provides a preliminary result, identifying regularities and comparison of the results of the two models of ACO and Greedy.

The concluding stage will describe the observed behaviors and formed findings based on the achieved results. In the same section recommendations for the application of the software usage on real conditions will be described.

# 5. Practical part

In the previous chapters the main aim was focused on the theoretical components and the development of a step-by-step plan of action to implement and to test the ACO method on own defined case-study parameters. The practical part of this paper will be divided into stages based on the previously outlined application of Applied Research in Chapter 4.3 and on the identified research methods. Based on this basis, the further research and the provided information will follow a structured order, including a description to each stage in details.

## 5.1 Stage 1: Definition

A major aim of this research is to evaluate the effectiveness of the ACO method involving that method to solve the routing problem. Thus, in Chapter 4.3, own given task was specified according to selected conditions that are needed to specify for creating own example. In stage 5.1 these tasks will be extended and more details will be clarified.

An important component is the choice of the research area, for which region the calculations will be formed and on the information of which data the model will be constructed and simulation will be implemented. The purpose of this research is to test the performance of the Ant Colony algorithm. There is why in the given conditions no strict requirements on the choice of observed area are given, so that as well for the density of cities and distances between them. The research should show the correct results even if the distances between cities kilometrage is small. Therefore, the region was chosen according to the most familiar in terms of geographical data and transport network - Wesel district. Based on Kolkmann et al. (2023) the region includes 13 towns. This number is optimal for further ACO measurements and experiments.

In forming an own example for model ACO testing, it should be also specified that the transport needs to visit each of the observed points. These points are the town indicators in the Wesel district. The transport in the task should return back to the starting point and show the optimized route. For that purpose, in searching for an optimal path in the selected area the problem can be represented as in the form of graphs. Towns are going to be the vertices in the graph and edges between vertices might be counted as the lengths of routes. It should be noted that there is no exact specification of transported goods. However, it should be kept in mind that the goods are not be perishable, so the routes do not have fixed or prioritized delivery and exact time of arrival to a particular city. This clarification is important for the ACO algorithm to work, as the ants will proceed their own logic rather than following already known set paths.

In Figure 5 the location of the towns in the selected region is represented on the map. The titles of the towns will be further summarized in the combined tables in Chapter 5.2.

Figure 5 - The observed research area for a case study

For the case-study application, assume that a company wants to optimize its routes operations. Using the ACO method, it is necessary to reduce economic costs, in this research the focus will be on the fuel costs and it is necessary to find an optimal route. In

this context, variables can be either a group of factors or a single parameter. In this study, the observed variables are: distances between towns, tariffs and fuel costs in Germany.

The initial other method will be used the Greedy algorithm. This step is necessary to examine the efficiency by implementing the ACO model in the created system. Moreover, to check how well the method was able to reduce fuel costs and reduce the overall total distance of the route.

This research will be conducted in pure experimental conditions to exclude the possible influence of the external factors. The term of uncertainty will consider those conditions where the impact of the external environment on the experiments and during the gathering of numerical indicators are not taken into account. This will allow to evaluate the algorithm and its performance in general, so that the facilitates of assessment of the ACO and Greedy algorithms are going to be under the same conditions. Moreover, both models can be adapted to any variables to make the evaluation and calculation more flexible and personalized.

The environment condition can be further customized and elements such as time intervals or information about traffic congestion and other necessary factors can be added into the observed model. However, at this moment it is the important to realize the mathematical model of the ACO algorithm and to check what results the model will demonstrate during the experimentation and simulation phase.

According to Eldosouky, Ibrahim and Mohammed (2014) under uncertainty terms additional 5% variation should be considered. To calculate the price fuel expenses 1.05 will be added to the basic amount. The calculations will be presented in more detail in Chapter 5.2.

### 5.2 Stage 2: Design and Planning

At the stage 5.2 it is necessary to collect the data set that is needed for the further work with the implementing and testing the ACO method. For this purpose, by using the measurement tools the numerical indicators will contain information about distances between cities and current fuel tariffs. In order to gather numerical indicators of outlined parameters, the decision was made to use open resources where the accuracy materials are provided.

The distances between town will be stored and collected as a matrix using the Google Maps platform (2025). The table was formed as follows: the rows on the left side contain

information about the town from which the movement starts and are acting as departure towns. The column top headings contain information about to which town the arrival in town occurs.

Table 1 is the representation of the information about corresponding distances. At the moment of gathering data information, distances between town were measured by using an existing transportation network of the region. It means that the kilometers of distances between town were not gathered by air. In completing the table, if multiple routes are available between observed towns, the length to each distance was chosen based on the average value. Thus, between some points the return routes may have small differences in kilometers in comparing to direct routes.

| | Wesel | Dinslaken | Hamminkeln | Kamp-Lintfort | Moers | Neukirchen-Vluyn | Rheinberg | Voerde | Xanten | Alpen | Hünxe | Schermbeck | Sonsbeck |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Wesel | | 15 | 11 | 25 | 32 | 38 | 19 | 9 | 19 | 15 | 14 | 21 | 22 |
| Dinslaken | 15 | | 30 | 31 | 24 | 33 | 28 | 11 | 30 | 26 | 9 | 21 | 33 |
| Hamminkeln | 11 | 30 | | 32 | 39 | 39 | 27 | 19 | 26 | 22 | 23 | 28 | 29 |
| Kamp-Lintfort | 25 | 31 | 32 | | 13 | 9 | 10 | 30 | 32 | 11 | 37 | 44 | 24 |
| Moers | 32 | 24 | 39 | 13 | | 7 | 12 | 34 | 30 | 23 | 37 | 51 | 28 |
| Neukirchen-Vluyn | 38 | 33 | 39 | 9 | 7 | | 16 | 40 | 35 | 18 | 48 | 57 | 31 |
| Rheinberg | 19 | 28 | 27 | 10 | 12 | 16 | | 25 | 18 | 8 | 30 | 38 | 24 |
| Voerde | 9 | 11 | 19 | 30 | 34 | 40 | 25 | | 24 | 20 | 11 | 22 | 27 |
| Xanten | 19 | 30 | 26 | 32 | 30 | 35 | 18 | 24 | | 13 | 28 | 35 | 9 |
| Alpen | 15 | 26 | 22 | 11 | 23 | 18 | 8 | 20 | 13 | | 26 | 34 | 11 |
| Hünxe | 14 | 9 | 23 | 37 | 37 | 48 | 30 | 11 | 28 | 26 | | 12 | 32 |
| Schermbeck | 21 | 21 | 28 | 44 | 51 | 57 | 38 | 22 | 35 | 34 | 11 | | 38 |
| Sonsbeck | 22 | 33 | 29 | 24 | 28 | 31 | 24 | 27 | 9 | 11 | 32 | 38 | |

Table 1 - Distance matrix, km

Another important indicator which is involved in the calculations is fuel tariff information. Data on current tariffs in Germany in the Wesel district was collected according to the Tankerkoenig platform (2025). The data was gathered on the period of 18.02.2025. While gathering information, in some towns there were multiple options of tariffs with different prices. The decision was made to take the average tariffs, in order to ensure that further calculations would be fair and do not inflate the cost for specific towns, avoiding the most expensive and lowest rates. Table 2 provides information on fuel tariffs in euros, which is required to construct the matrix.

| Towns title | Price, euro |
|---|---|
| Wesel | 1,599 |
| Dinslaken | 1,609 |
| Hamminkeln | 1,629 |
| Kamp-Lintfort | 1,579 |
| Moers | 1,579 |
| Neukirchen-Vluyn | 1,619 |
| Rheinberg | 1,589 |
| Voerde | 1,619 |
| Xanten | 1,639 |
| Alpen | 1,579 |
| Hünxe | 2,199 |
| Schermbeck | 1,639 |
| Sonsbeck | 1,649 |

Table 2 - Fuel tariffs, euro

It should be mentioned that the gathered data denotes the cost of fuel at a petrol stations to a particular town. In other words, to form the matrix the inverse data are also required, where exactly the transport has been fueled.

One option on how exactly to create the fuel matrix was to use the average values from the departure and arrival towns. The logic of this action would be so, that it is not defined in which town the vehicle has refueled. In that case, it is necessary to take into account this probability of refueling vehicles in two towns at once. In creating the cost of an average route both values should be taken into calculations. However, this method of data collection has a significant weakness. In forming the matrix using that principle, the tariffs would be symmetrical. Then the differences in path selection for the algorithms would be problematic and would not show the exact result as for in choosing the concrete sequence of towns and also for optimizing the economic benefit and component in this case-study.

To solve this disadvantage, it was decided to use the following matrix design principle. While filling in the table, the fuel price was considered from the departure town. This means that, for example, if the vehicle travelled on the route Wesel-Dinslaken, the price is indicated from the starting city Wesel 1,599. For the reverse Dinslaken-Wesel route also include a price from the departure city, that is Dinslaken and is equal to 1.609. In this way, it is possible to construct direct and reverse routes that reflect different information. As a result, it allows ants to take into consideration this difference in cost and make decisions more correctly in choosing paths. These numerical cost measures are summarized in Table 3.

| | Wesel | Dinslaken | Hamminkeln | Kamp-Lintfort | Moers | Neukirchen-Vluyn | Rheinberg | Voerde | Xanten | Alpen | Hünxe | Schermbeck | Sonsbeck |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Wesel | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 | 1,599 |
| Dinslaken | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 | 1,609 |
| Hamminkeln | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 | 1,629 |
| Kamp-Lintfort | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 |
| Moers | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 |
| Neukirchen-Vluyn | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 |
| Rheinberg | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 | 1,589 |
| Voerde | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 | 1,619 |
| Xanten | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 |
| Alpen | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 | 1,579 |
| Hünxe | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 | 2,199 |
| Schermbeck | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 | 1,639 |
| Sonsbeck | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 | 1,649 |

Table 3 - Fuel tariff matrix, euro

Moreover, for further calculations it is necessary to understand what is the average fuel consumption of the vehicle will be. For the created task assume that a company uses a MAN TGL model with a D0834 engine with a load capacity of medium level. According to the technical manual Maintenance and Service Product Recommendation MAN Truck (2024) fuel consumption is 17 liters per 100 km.

The fuel cost calculating is based on the formula according to Burnham et al (2021):

$$\text{Fuel Cost} = \left(\frac{\text{Distance} \times \text{Fuel Consumption}}{100}\right) \times \text{Fuel Price} \qquad 6)$$

As it was mentioned earlier additional 5% should be added to the formula in order to calculate the full cost under the chosen conditions:

$$\text{Fuel Cost} = \left(\left(\frac{\text{Distance} \times \text{Fuel Consumption}}{100}\right) \times \text{Fuel Price}\right) \times 1{,}05 \qquad 7)$$

To provide the calculations based on formula 7 the data from Table 3 would be used for further work. The new developed numbers will also be arranged in a form of matrix and represent the fuel cost in euros.

| | Wesel | Dinslaken | Hamminkeln | Kamp-Lintfort | Moers | Neukirchen-Vluyn | Rheinberg | Voerde | Xanten | Alpen | Hünxe | Schermbeck | Sonsbeck |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Wesel | | 4,3 | 3,1 | 7,1 | 9,0 | 10,8 | 5,5 | 2,7 | 5,3 | 4,3 | 4,0 | 6,0 | 6,2 |
| Dinslaken | 4,3 | | 8,6 | 9,0 | 6,9 | 9,4 | 8,1 | 3,0 | 8,5 | 7,6 | 2,6 | 6,0 | 9,5 |
| Hamminkeln | 3,1 | 8,7 | | 9,4 | 11,3 | 11,3 | 7,7 | 5,6 | 7,4 | 6,5 | 6,7 | 8,2 | 8,4 |
| Kamp-Lintfort | 7,0 | 8,8 | 9,1 | | 3,5 | 2,6 | 2,7 | 8,6 | 9,1 | 3,1 | 10,3 | 12,3 | 6,8 |
| Moers | 8,9 | 6,7 | 11,0 | 3,5 | | 1,9 | 3,5 | 9,5 | 8,4 | 6,5 | 10,3 | 14,4 | 7,9 |
| Neukirchen-Vluyn | 10,9 | 9,4 | 11,2 | 2,7 | 2,0 | | 4,6 | 11,5 | 10,0 | 5,1 | 13,8 | 16,6 | 8,9 |
| Rheinberg | 5,5 | 8,0 | 7,5 | 2,7 | 3,5 | 4,5 | | 7,1 | 5,0 | 2,2 | 8,4 | 10,8 | 6,9 |
| Voerde | 2,7 | 3,1 | 5,5 | 8,8 | 9,7 | 11,5 | 7,2 | | 6,8 | 5,9 | 3,3 | 6,3 | 7,8 |
| Xanten | 5,4 | 8,7 | 7,5 | 9,4 | 8,7 | 10,2 | 5,2 | 6,9 | | 3,9 | 8,3 | 10,2 | 2,6 |
| Alpen | 4,2 | 7,4 | 6,3 | 3,1 | 6,5 | 5,0 | 2,1 | 5,7 | 3,7 | | 7,3 | 9,5 | 3,1 |
| Hünxe | 5,5 | 3,6 | 9,1 | 14,3 | 14,4 | 18,8 | 11,6 | 4,5 | 11,1 | 10,2 | | 4,7 | 12,6 |
| Schermbeck | 6,1 | 6,1 | 8,2 | 12,7 | 14,9 | 16,8 | 11,1 | 6,4 | 10,2 | 9,9 | 3,3 | | 11,2 |
| Sonsbeck | 6,4 | 9,7 | 8,5 | 7,1 | 8,2 | 9,1 | 7,2 | 8,0 | 2,6 | 3,2 | 9,4 | 11,2 | |

Table 4 - Total fuel costs, euro

For the next step 5.3, the data regarding the distances between towns from Table 1 and the fuel cost values from Table 4 will be used. Based on this data, the mathematical models from Chapter 3 will be applied to the simulation. Overall actions will allow to correctly conduct experiments and to test the ACO algorithm, as well as the algorithm for comparison – Greedy. In next section solution of the created example about finding the optimal route and reducing economic costs will be procedure.

### 5.3 Stage 3: Implementation

In the current phase of implementation is being performed to create the code for the algorithms that will be used for the optimization calculations. In the algorithm, the Ant Colony method is implemented as the observed model. It will be used as a monitored behavior method for receiving calculations in finding optimal routes taking into account the fuel cost. Moreover, an advanced EAS method of the ACO will be also implemented in a code to get more accurate results. Greedy Algorithm will be used as a comparative model to the ACO and EAS. In a basic view, Greedy algorithm is introduced in many other heuristic algorithms and act as a competitive to ACO in the speed of getting calculations.

The Java language was chosen for the application model design. Code creation was conducted according to own knowledge, but with the help to Java sources and manual. Moreover, to the Java library and specification, loops, methods, commands and basic functions as are presented in the manual of Gosling et al (2015). Accessing to the manual was necessary to avoid errors in a phase of the code writing, to correctly implement mathematical expressions and to realize the operation of the algorithms in a comprehensive way.

In the program code there were created blocks, where each of them has a key role in the mechanism's functioning. In total, 7 classes were created, where in each of them the main stages of algorithms and different steps to optimize the necessary operations of the process of the corresponding models are being performed. The class is the framework that provides information about the applied methods describing the behavior of the object and the variables that are involved (Samoylov, 2018). The interaction of all classes allows to create a simulation and to test the algorithms based on the gathered numerical data for the Wesel region.

In the program, the developed Main.java class serves as the basis for communication between other classes in the system and also presents outputs of the results of the program in the compiler for further analysis.

For the creating the simulation, it is essential to note which role each of the classes perform in the process according their functions. Moreover, below will be described the classes and the structure of the program. Figure 6 includes the process of interactions in the system and serves to be as a diagram for better understanding of how the simulation works. That is the representation of the designed system with a particular logic of interaction between created classes. It means that the relationship between classes is responsible in functioning the whole designed code, where the program accurately and correctly realizes the work of algorithm implementation. The software is simulating corresponding algorithms under created conditions and tasks.

### 5.3.1 Program classes

In the implemented software package, several key actions between the created classes can be distinguished:

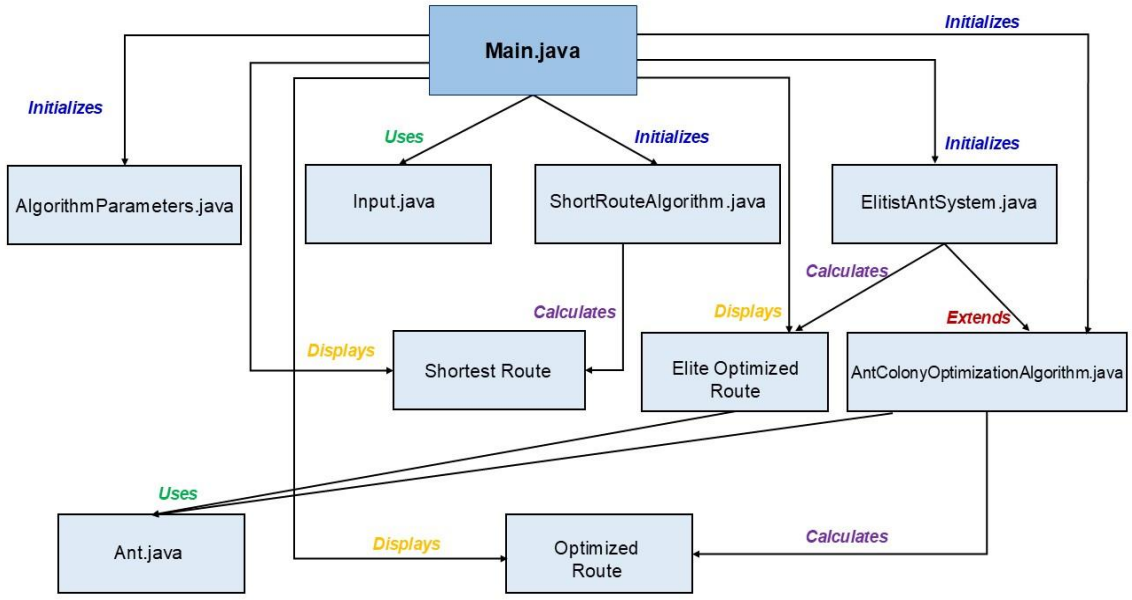


Figure 6 - Implemented code framework

*Uses* –operating with input data of the class and use it for the further purposes;

*Initializes* – a class creates an object to which this action is directed;

*Calculates* - mathematical calculations are performed, in this example the directions are addressed to the action of finding an optimal solution based on mathematical models of ACO, EAS and Greedy algorithms;

*Displays* - the calculated result is shown;

*Extends* - this action acquires the capabilities and features of the class to which this action is addressed.

As it is illustrated in Figure 6, the above actions are addressed to the key modules in the program. Each of them is created to carry out the optimization process and to implement the simulation on the developed example. By considering each of the modules of the programmed code separately, it will be determined what information they contain in the blocks.

1. The simulation starts with creation of the Main.java class. It serves to be in a program as a main unit where all the provided operations are occurred. The main functions are to start the simulation model, to extract, read file and use input data, to organize consistency in coordination between the created classes; to run all the necessary methods and models that are involved in the process of achieving optimization and to provide the obtained results to the program compiler.

In the Main class the particular objects are formed for further use of the models' capabilities functionality: AlgorithmParameters, AntColonyOptimizationAlgorithm, ElitistAntSystem and ShortRouteAlgorithm. From the Input class, the data are used and processed on towns - vertices, distances and fuel cost matrices from Section 5.2.

The use of the classes of ShortRouteAlgorithm, AntColonyOptimisationAlgorithm and ElitistAntSystem, is procedure in order to operate with the methods that calculate the optimal routes on a given created example in the Wesel region.

The outputs from the calculations are displayed in the program compilator. The received results conduct information about the calculated best options of the route for the observed towns in the district. At the same time the outputs denote the length of the path in kilometers and fuel costs in euro. These parameters are obtained through implementation of the ACO, Greedy Algorithm and EAS models.

2.AlgorithmParameters.java is an additional set of parameters that is required for the implementation of the ACO and EAS models. It was decided to create a separate class of AlgorithmParameters due to its flexible configuration and work convenience. Based on

that step there is no need to recompile the file if some parameters should be changed or corrected in the program. Also, the class acts as a calibration for the designed model before running the algorithm. It means that a synchronization of functions with determined parameters are occurred in a system. These outlined parameters that are affecting to the work are:

$\alpha$ – a pheromone importance indicator which is used to find the assumed optimal route;

$\beta$ – an indicator of the heuristic data used in the search of the estimated optimal route;

ρ – coefficient of the pheromone evaporation;

Q – parameter of the total amount of pheromone being left behind by the ant;

$v$ – the number of elite ants.

Chapter 5.3.2 will demonstrate in more detail how the numerical measures were selected to ensure the effectiveness of the algorithms ACO and EAS.

3. AntColonyOptimisationAlgorithm is the class that performs the function of route construction and cost calculation using the ACO algorithm model. The process itself is built with the help of mathematical formulas and step-by-step structure of ants' behavior, the application of those components was described in Chapter 3. The class consists of the creating virtual ants that are searching the appropriate route based on the pheromone level. The number of ants depends of the amount of the observed vertices and in the provided simulation where these parameters were chosen to be equal to each other. These multi-level processing is focusing on the route optimization and further analysis of the obtained results.

4. ElitistAntSystem is the class which indicates to be as the extended version of the standard ACO algorithm - AntColonyOptimisationAlgorithm. It implies a modernized order of updating pheromone trails on the paths. The principle is to add to the shortest paths an additional number of pheromones that belong to elite ants. Parameters from class AlgorithmParameters are used in ElitistAntSystem operation.

5. ShortRouteAlgorithm is a class of Greedy Algorithm. Creating a route depends on the shortest distances between cities. The principle of the algorithm realizes by defining a starting point on the map. At this moment at each step the selection will be made due to of the nearest unvisited city. It is necessary to take into account the point that has not been taken before.

6. Ant is a class where a structure about one ant in the ACO system contains. Moreover, the class acts as a virtual ant in the system. The virtual ant helps to keep tracking information about its route, intermediate data about visited towns and stores dynamic factors about the object.

7. Input is a class where input data for the program stores. The collected matrices from Chapter 5.2 are contained in this class. Thus, data on distances, fuel costs and town titles are taken for providing calculations and for reading information of the files.

Interaction between modules in the program can be represented as synchronized flow. Figure 7 shows the step-by-step operations of all classes and their functioning of the system can be visualized in the following sequence:

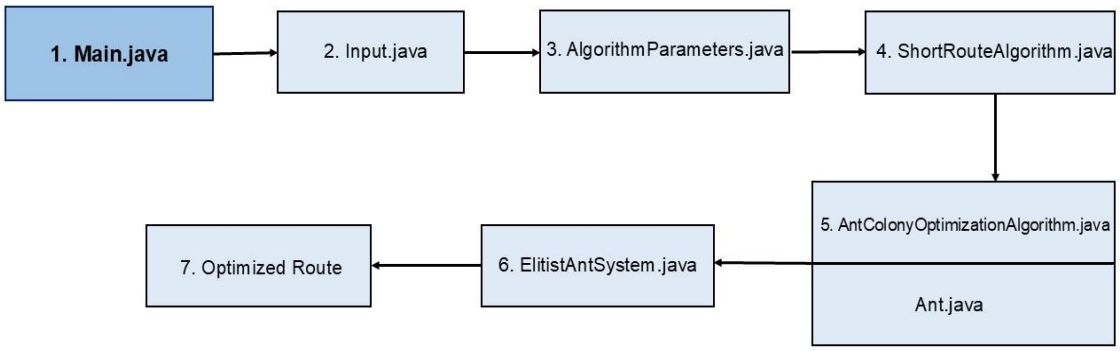


Figure 7 – Java classes flow

1. The work of the program starts in Main, where class runs the simulation;

2. Input loads the collected data set. Main manipulates the data from Input.java and uses information about kilometers and fuel costs.

3. To operate with the coefficients that are essential for the ACO and EAS models Main create a class of the AlgorithmParameters. The class is responsible in providing operations to a system models.

4 – 6. The Main class creates objects of classes ShortRouteAlgorithm, AntColonyOptimisationAlgorithm, ElitistAntSystem. Main provides calculations based on the data passed by Input. The behavior for a single ant is used from the class Ant.

7. ElitistAntSystem, AntColonyOptimisationAlgorithm, ShortRouteAlgorithm represent the calculated solution of the optimal route and information about fuel cost are displayed in the Main.

The provided connections between the classes are needed for the realization of the algorithms. By providing a simulation, algorithms presenting a solution for a given problem by using a collected data set. On the other hand, an important part in realization of the simulation is the selection of correct parameters in the AlgorithmParameters for the appropriate chosen conditions. In that case these factors may affect to the basic operations of the observed ACO algorithm.

### 5.3.2 Parameters configuration

The main part in realizing and working with the ACO algorithm and it is also connected to its modernized extended version EAS is to determine the suitable parameters. In order to maximize the performance from the algorithms in this research, it is needed to find the appropriate numerical parameters that affecting to the overall process of the system. Shrivastava and Kumar (2019) identify the coefficients that are influencing to the optimization by implementing the models of ACO and EAS. In their study the authors outline the parameters as: Alpha (α) - pheromone coefficient; Beta (β) - heuristic data coefficient; Rho (ρ) - pheromone evaporation coefficient. Additionally, other important parameters such as Q - coefficient of the total amount of leaving pheromone will also be involved in the test validation. According to Pasquier et al. (2007), in searching for the correct coefficients of Q, suppose that it does not affect much on the work of the system. This judgement will be tested on own gathered data set and then the effect will be seen. For the functioning of the EAS system, it is necessary to determine the correct number of v - elite ants that should be added to the algorithm. In searching the corresponding elite ants, the effect of these number on the distance will be observed.

In order to determine the values, the formulas 1-5 from Chapter 3 were used for those purposes in parameter's calculation. The data from Table 1 was needed in the testing process. The provided below mechanism of calculation was implemented based on the corresponding data to Wesel town with an index of 0 in the code realization. The calculations were realized in cycles with gradual increases in them. To start the calculations, the initial value from which the process beginning should be determined. Firstly, Alpha and Beta parameters were changed gradually with the interval of 0.5. The initial checking numbers were chosen to be 0.5 and the final number was 5. The Evaporation - $\rho$ parameter was changing by the step of 0.1, starting with a value of 0.1

and a maximum was value of 10. Parameter Q was varied with step of 50, tests are procedure further with starting number 50.

During the calculation process was created 4 cycles. The first cycle was calculated by changing the parameter Q, where Alpha Beta and Evaporation were at initial values. There the Q parameter was changed in increments of 50 until the value of 1000 was reached. After that, by increasing only Q the obtained parameters for distances for ACO and EAS algorithms were reached.

In the next calculation cycle the increase was in the Evaporation parameter with unchanged Alpha and Beta. As it was done in the previous stage the Q parameter still changing with the interval of 50. This mechanism of actions was the following as was described above. It means each of the involved parameters was changed according to this logic.

After all calculations have been performed, the average values for that case should be realized. This action is necessary to reduce the resulting data set of rows and emphasize only the necessary information that is contained there. Thus, the data is determined and hat a structure of a parameter for the algorithms and the route in kilometers related to this parameter.

To start presenting the obtained results, the operations of the parameters should be considered in for details. For the Alpha parameter it is necessary to take all the values at which Alpha is equal to 0.5 and divide by the number of elements that are involved in the calculation. Thus, the average value for the distance to ACO and EAS models is found. The calculated indicators can be formed and presented in the form of a table, which is shown below in Table 5 according to each coefficient for Alpha.

| Alpha | ACO Distance | EAS Distance |
|---|---|---|
| 0,5 | 166,93275 | 166,3622 |
| 1 | 166,77195 | 166,64425 |
| 1,5 | 172,1807918 | 170,7675678 |
| 2 | 173,6975312 | 172,4334484 |
| 2,5 | 174,4227362 | 173,3495735 |
| 3 | 175,0519155 | 173,9993866 |
| 3,5 | 175,3351651 | 174,6485752 |
| 4 | 175,6035336 | 175,2131676 |
| 4,5 | 175,9199721 | 175,6173665 |
| 5 | 176,0347509 | 176,0381081 |

Table 5 - Alpha results calculation

The similar calculation mechanics will be applied to all other parameters. The tables with the all results information will be demonstrated in Chapter Appendices 8. In a graphical representation the observed gathered values are analyzing due to its performance and role in the calculations.

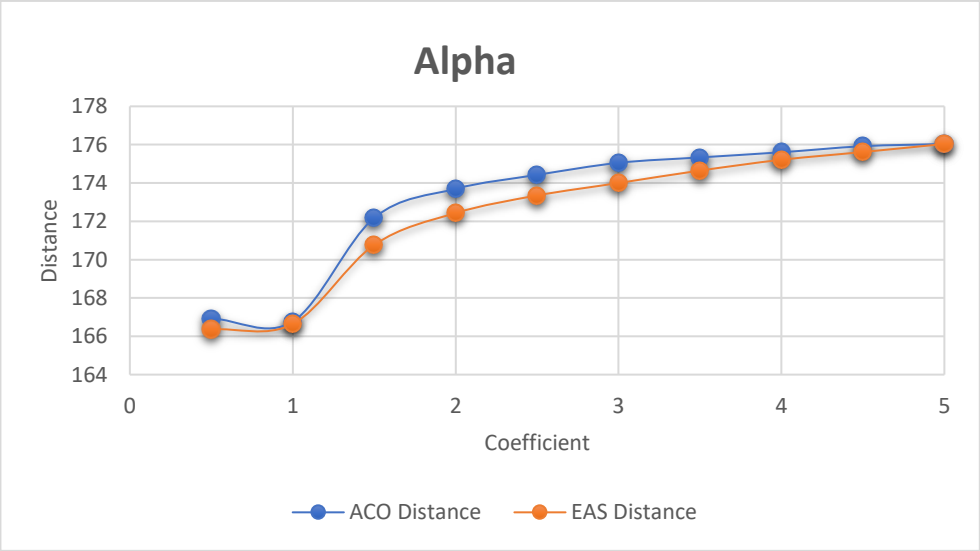


Figure 8 - Alpha configuration

Figure 8 shows that for Alpha the smallest length distances on the routes are demonstrated at values Alpha in the interval of 0.5 - 1. That is why the choice of Alpa should be based on the logic above and be selected in the lower range. This conclusion is made due to the fact where the algorithms show the most optimal distances for the researched dataset for the region. On the other hand, there is no reason to choose higher values of Alpha as they show the more longer length distances.

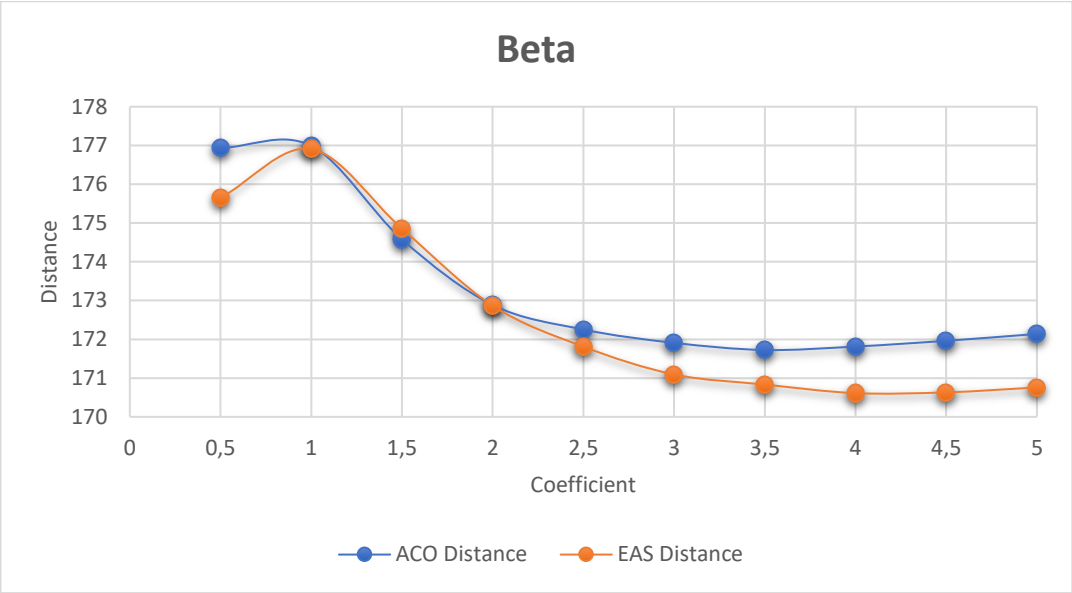


Figure 9 - Beta configuration

Figure 9 demonstrates the data for the Beta parameter. As can be seen in the graph, the choice of selecting correct Beta coefficient follows a different logic. The smaller value of the Beta, the worse the ACO and EAS algorithms work. The more Beta is, the less travelled kilometrage appears. This indicates that the value is going to be between of 3-5 for the observed data. In the simulation model, it was decided use the coefficient of 4.

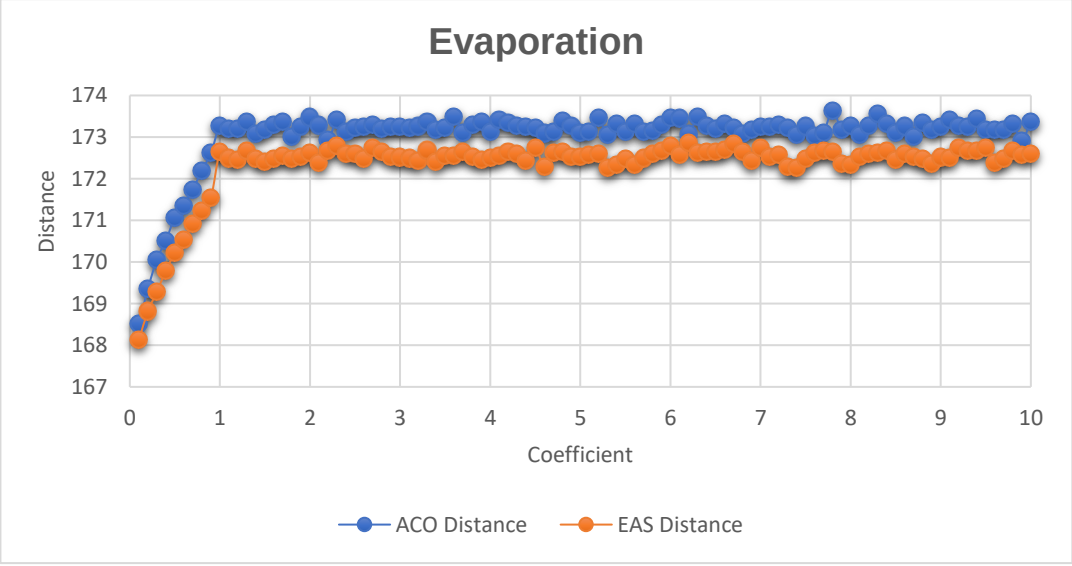


Figure 10 - Evaporation configuration

Figure 10 shows the Evaporation parameter, from which the value performs better at the minimum number. The more coefficient tries to increase, the longer the distance length becomes. This means that for our dataset the coefficient should be at the minimal level. The value 0.1 will be used in future calculation simulation.

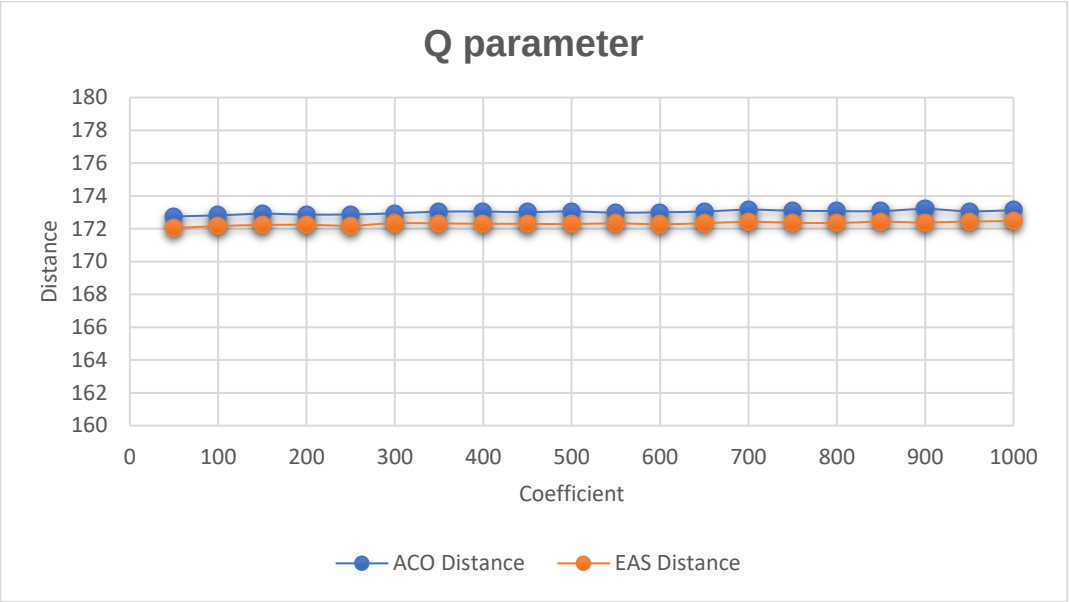


Figure 11 - Q configuration

The parameter Q is demonstrated in Figure 11. When the parameter is changing, the route lengths are remaining approximately the same during the whole distance length. It can be concluded that changing Q has almost no effect on the overall calculations. The choice therefore can be made in the direction of minimum or to maximum. In the simulation the parameter will be equal to 500.

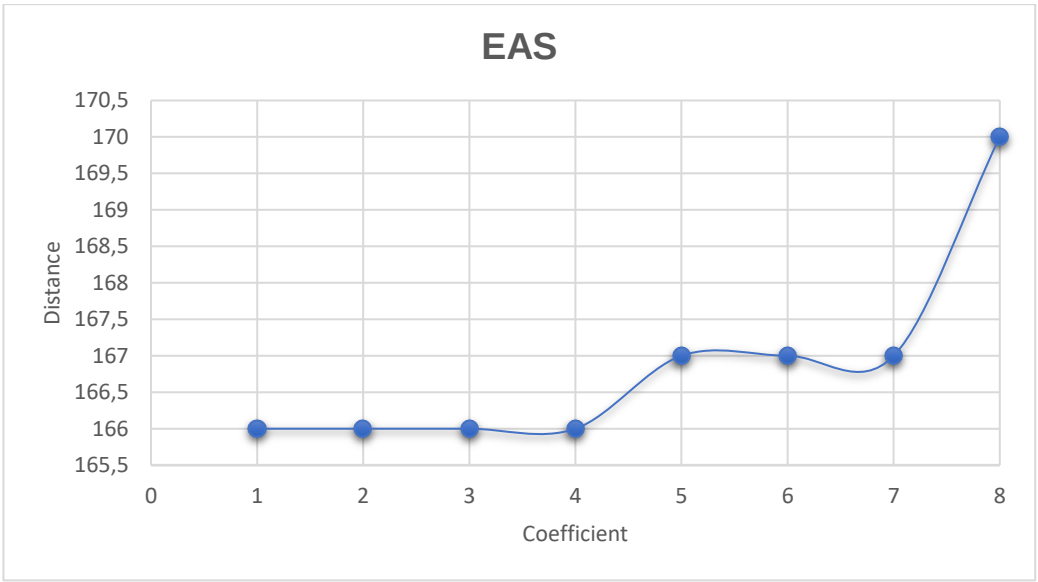


Figure 12 - Ant number effect

In solving the optimization problem in this research, a system of elite ants also has been implemented. However, to work efficiently, it is required to determine what is the exact number of elite ants to create. Figure 12 describes the effect of the number of ants to the length distance. As the number of created virtual ants increases, the kilometrage on the

route increases as well. It leads to a poor proposal. Stably for this task, the algorithms work when up to 5 ants are introduced into the system. In the code 4 elite ants will be involved to search the solution.

In the context of the program application, these identified parameters are contained in the AlgorithmParameters class. The view in the program compiler is represented in Figure 13.

```
public class AlgorithmParameters {
    // Constants for the ACO algorithm
    3 usages
    public double Alpha = 1.0;
    3 usages
    public double Beta = 4.0;
    3 usages
    public double Evaporation = 0.1;
    4 usages
    public double Q = 500.0;
    1 usage
    public double PheromoneStartValue = 100.0;
    3 usages
    public int maxIterationsNumber = 1000;
    2 usages
    public int iterationsWithoutImprovementCounter = maxIterationsNumber;
}
```

Figure 13 - Algorithm parameters class

As it is shown in Figure 13, there are 3 more additional parameters that are involved in the work of AlgorithmParameters class. The parameter pheromonStartValue is the value of initial pheromone level indicates at the beginning to each vertices. It should be noted the above provided calculations calculation of all parameters was procedure for this number of pheromones. The number itself was chosen arbitrarily, from which other further parameter calculations were made.

MaxIterationNumber is a parameter denoting how many times the algorithm will be running in the program in searching the optimal computation results. Shrivastava and Kumar (2019) determine that when the number of iterations is large, the ants in the system are starting to following the same route without further optimization. It means that for an algorithm with a large value MaxIterationNumber, only an unchanged solution will be displayed in the output over the time. To avoid this situation, the MaxIterationNumber parameters was set in that way to prevent the algorithm from working pointlessly. The set the parameter equal to 1000.

The value iterationsWithoutImprovementCounter = maxIterationsNumber is needed to optimize the algorithm of ACO and EAS. The parameter means that the algorithm is running many times and if there is no improvement, the calculations stop the working process. This value is equal to the maximum number of iterations, meaning to stop calculations if the optimal route could not be long changed. This is why in order to avoid a looping problem, the number is equal to its iterations value. This helps the algorithm to work faster and not to waste time on further long calculations.

### 5.3.3 Code realization

The main task in implementing the algorithms into a program code is to correctly apply the theoretical basis into a practical application. The code is based on the statements from Chapter 3, taking into account the sequence of algorithms, mathematical formulas for calculations and approaches in finding a solution to the problem. In this section, the particular realization of the algorithms will be observed regarding to the code writing in Java language.

#### 5.3.3.1 Greedy Algorithm

In the class of ShortRouteAlgorithm the Greedy model is creating through *public static List<Integer> calculate(int numCities, double[][] values, int startCityIndex)*. A list of integers *List<Integer>* by the method *calculate* performs calculations using the number of cities – *numCities,* also collected data in the matrix *double[][] values* and starting its route in the selected initial city *startCityIndex.* Having created *List<Integer> route = new ArrayList<>()* data about visited cities will be stored in it. The checked city due to *visited[startCityIndex] = true* is assigned to have been *true* value, adding it to the route map list *route.add(startCityIndex).* By setting the way of int *currentCityIndex = startCityIndex* store the received information about the current city from which the movement is made.

Introducing the loop *while (route.size() < numCities)* is performed, until all the cities *numCities* have been reached. Initially, the starting city is assigned to the value *int nextCityIndex = -1*, as the decision which city will be the starting city has not been made and selected yet. Since the shortest path is being explored, the variable with the minimum *double minValue* stores information about the minimum distance. In this case, *Double.MAX_VALUE* is the starting value, which will serve as the maximum possible indicator. When the first distance is found *minValue* will update its setting. In the loop operation, a condition is checked *if (!visited[i] && values[currentCityIndex][i] <*

*minValue)*. When both of these conditions are met, then *visited[nextCityIndex] = true* happens, which takes this city into account and adds it to the map list route.add(nextCityIndex). In a such way, the Greedy algorithm runs in the searching through all the cities by looking for the closest one and stops the selection process when all cities are visited.

#### 5.3.3.2 Ant Colony Algorithm

In this section the implementation of the AntColonyOptimizationAlgorithm class will be considered. The important elements are the functioning of mathematical formulas 1 - 2 in this class. The code is created by *public AntColonyOptimizationAlgorithm(int numberOfAntsCities, double[][] values, AlgorithmParameters algorithmParameters)*. The quantity of ants *numberOfAntsCities* participating in the algorithm is equivalent to the number of cities being considered, *double[][] values* are the matrix notation and adjustable parameters for functioning of the ACO are the *algorithmParameters.* The array *pheromones = new double[antsAndCitiesCount][antsAndCitiesCount]* contains the level of pheromones between the vertices of the matrix whose size is given by *antsAndCitiesCount.* For each *city for (int i = 0; i < antsAndCitiesCount; i++)*, that is identified as *i,* a starting vertices is selected from where the ants begin their journey. The ants follow *for (int j = 0; j < antsAndCitiesCount; j++)* all possible routes and leave an initial pheromone level that is equal to number from the parameter *pheromones[i][j] = algorithmParameters.PheromoneStartValue.*

Using the method *calculate(int startCityIndex)* the computational work of the ACO algorithm is performed. To find the optimal route, variables are created such as: *bestRouteValue = Double.MAX_VALUE* is a parameter that stores the cost of the best suitable route, set of *List<Integer> bestRoute = new ArrayList<>()* will hold the information of the best-found path. Further *int iterationsNumberWithoutImprovement = 0* represents the measurement of the number of replays in the system without any progression. In the class AlgorithmParameters, this parameter has been assigned to *maxIterationsNumber* for avoiding looping moment in the ACO. As the cycle is repeating for multiple times, in order to store data and cost about all routes a list was created *List<List<Integer>> allRoutes* and *List<Double>allRoutesValues.*

The behavior of the ant is read from the class *Ant*. For each ant a*ntsAndCitiesCount* follows its own route from the initial given point *startCityIndex*. The search is done by introducing the method *createRoute* based on ant *a* и *startCityIndex*. For length

calculations using *double totalValue = getTotalValueForRoute(route)* obtain the relevant data and add it in the *allRoutes.add(route)* distance list and to the *allRoutesValues.add(totalValue)* cost list.

To determine whether the found route has become optimal and shorter, these conditions should be checked. In that case, for veryfing conditions the data for the length in accordance with *bestRoute = route*, and same for the cost *bestRouteValue = totalValue* should be updated. If the best option is identified *iterationsNumberWithoutImprovement = 0*, meaning be reset to the initial parameters to zero indicator. At this point the pheromone level is also updating according to *allRoutes* and *allRoutesValues*.

A suitable improved route is not found for a long period of time, then i*f (iterationsNumberWithoutImprovement>=algorithmParameters.iterationsWithoutImprovementCounter)* the algorithm must be finished in that terms, otherwise the change in the value of number of iterations must be ineffectively increased. The algorithm is completed and is being returned to the *bestRoute*.

Creating *getTotalValueForRoute(List<Integer> route)* in order to calculate the sum of the length of the distances, starting its calculation with the total = 0.0. By adding the distances between each node of the vertices *total += values[route.get(i)][route.get(i + 1)* these values are adding to the total sum. In the task we have a rule that an ant needs to return back from the final city to the initial city. In this case it is necessary to consider this return distance too by applying *total += values[route.get(route.size()-1)][route.get(0)]*.

Forming one ant by *createRoute(Ant ant, int startCityIndex)* promotes to start the ant to search for the best solution beginning its route with s*tartCityIndex*. The reached city is marked in a system by *ant.visitCity(startCityIndex),* and this city *currentCity = startCityIndex* is memorized and the next step is going to be from it. The movement continues until the ant has created its way through all the given cities. The ant chooses the following city based on pheromone level *by int nextCity = nextCity(currentCity, ant)*. The moving process of an ant stops, when the ant reaches all the points on that created path.

The decision to travel to a particular city depends on the transition probability of the choice. The total sum of probability starts with zero data with *double totalProbability = 0.0*. Formula 1 is implemented in such a way, where *i* – is a representative value describing particular cities and if ant has not visited *!ant.getVisited(i)* the corresponding city yet, the probability is calculated by:

```
for (int i = 0; i < antsAndCitiesCount; i++) {
   if (!ant.getVisited(i)){
      probabilities[i] = Math.pow(pheromones[currentCity][i],
algorithmParameters.Alpha) *
            Math.pow(1.0 / values[currentCity][i], algorithmParameters.Beta);
      totalProbability += probabilities[i];
   }
}
```

The data of the parameters are set in the *AlgorithmParameters* class и are used in *AntColonyOptimizationAlgorithm* class to calculate the transition probability. The need to sum to *totalProbability += probabilities[i]* is required in order to normalize the probability. This action is needed due to the fact that further the algorithm will find the next city more conveniently and the number will be in the interval between 0 and 1.

The algorithm needs to update the level of pheromones over the iterations. Formula 2 provides the mathematical application how to proceed the action of updating the pheromones. In a code realization the formula is represented by the *public void updatePheromones(List<List<Integer>> routes, List<Double> routeValues)*, where the leaving pheromones between the vertices of cities *i* and *j* get the value of *pheromones[i][j] *= (1 - algorithmParameters.Evaporation).*

It is important to note, in order to regulate the pheromone level, we need to set an interval to the pheromones evaporation in order to prevent the dropping it to the 0 indicator. That is why the appropriate bounds should be set. When the value gets minimal value of *(pheromones[i][j] < 0.01)* equate this value to pheromones*pheromones[i][j] = 0.01*. The update happens on each route and for each ant. The number of pheromones *pheromoneAdd = algorithmParameters.Q / routeValue* is kept on this path according to the length on the distance and occurs up to the last city. Pheromones are leaving on the routes according to their direction *pheromones[from][to] += pheromoneAdd* and to *pheromones[to][from] += pheromoneAdd.*

#### 5.3.3.3 Elite Ant System

The EAS version is an extended algorithm of the standard model of the ACO model. The working principle is similar to the ACO, but acts in deeper with using a set of features in the form of elite ants. The algorithm is created by *public ElitistAntSystem(int numberOfAntsCities, double[][] values, AlgorithmParameters algorithmParameters, int*

*eliteAntsCount)*, the variables contain the same information as in the ACO. However, a parameter that indicates the number of adding elite ants is performed by *eliteAntsCount*. Since the ACO model is the foundation framework for the EAS it is essential to use method *super(numberOfAntsCities, values, algorithmParameters)*, sending the implied information there avoiding the moment of code duplication.

The pheromone updating has a form of *double pheromoneAdd = algorithmParameters.Q / bestRouteValue*, where at shorter lengths on node edges more pheromone should be added. If ants have passed through all the cities, their importance in accounting of pheromones on routes is *super.updatePheromones(routes, routeValues)*. By adding the pheromones also takes into account the number of created elite ants, increasing the influence of pheromone level on routes and considered in the direction of *pheromones[from][to] += eliteAntsCount * pheromoneAdd* and in the other way *pheromones[to][from] += eliteAntsCount * pheromoneAdd*. The number of elite ants that are being added in the code, was defined as *eliteAntsCount = 4.*

#### 5.3.3.4 Main

The Main class contains the major steps that are implemented there. By creating the method *calculateTotalCost(double distance, double fuel_cost, double fuel_consumption)*, the cost of fuel expenses can be counted and calculated. The calculation is realixed by the formula 7 of *1.05 * distance * fuel_cost * fuel_consumption / 100.0,* where the coefficient 1.05 indicates a factor of uncertain events while the other components reflect to the fuel cost, distance and fuel consumption. The input data from the distance and fuel cost matrices are loaded into the class in order to fulfill and create a total cost matrix *double[][] total_costs = new double[rows][cols]*. The *total_costs[i][j]* matrix is being filled by entering information with full summarized data into it.

By entering the name of a city in a line, the compiler will process the given and contained information about this city and all results calculations will be provided for it. For example, in case for the town Sonsbeck the written line in the code has the following view: *int startCityIndex = cities.indexOf("Sonsbeck").*

For the observed algorithms, the found routes are displayed in the console. Then for each of them the total price that is spent on fuel is calculated. For each of the algorithms the calculation has the form *calculateTotalCost(input.distances[path.get(i)][path.get(i+1)],input.fuel_costs[path.get(i)][path.get(i+1)], input.fuel_consumption).* For the distances results to each of the

algorithms the shortest route should be displayed (Integer integer : shortRoutePath) and we results will be visible System.*out*.print(cities.get(integer). In the compilator if we want to receive the overall calculated results the method *System.out.print* is used. For example, the output result for ACO has the form System.out.println(String.format('Total distance is: %.1f; fuel cost is: %.2f Euro', acoDistance, acoPriceCalculated)), where the information includes the traveled kilometers and the calculated cost.

The written implemented code is demonstrated in Chapter 7.

### 5.4 Stage 4: Results

This Chapter 5.4 presenting the obtained results from the experiment simulation model realized in the Java programming language. In the program compiler in the Main class outputs information about the optimized routes. This information includes the distance kilometers to the implemented three algorithms with the corresponding fuel cost. At the same time, the titles of the towns and in what order they should be visited are showed in the display. The provided results can be seen below. The program provides the results as follows and the routes combinations are taken from the complier. For example, the output results if the start and end point is the town of Dinslaken are:

**Short Distance Algorithm Route**: Dinslaken->Hünxe ->Voerde->Wesel->Hamminkeln->Alpen->Rheinberg->Kamp-Lintfort->Neukirchen-Vluyn->Moers->Sonsbeck->Xanten->Schermbeck->Dinslaken

Total distance is: 189,0; fuel cost is: 55,62 Euro

**Optimized Ant Colony Optimization Route**: Dinslaken-> Hünxe ->Schermbeck->Voerde->Wesel->Hamminkeln->Xanten->Sonsbeck->Alpen->Rheinberg->Kamp-Lintfort->Neukirchen-Vluyn->Moers->Dinslaken

Total distance is: 167,0; fuel cost is: 49,32 Euro

**Elite Ants Colony Optimization Route**: Dinslaken->Moers->Neukirchen-Vluyn->Kamp-Lintfort->Rheinberg->Alpen->Sonsbeck->Xanten->Hamminkeln->Wesel->Voerde->Schermbeck-> Hünxe ->Dinslaken

Total distance is: 166,0; fuel cost is: 48,79 Euro

The information obtained from the software compiler is useful for assessing the quality of the received routes. The full list of results with ordinal allocation of towns is presented in Appendices 8. For a complete analysis of the calculations, a combined table was created

for all total results. Table 6 summarizes all outputs from the simulation model for the 13 cities in the Wesel region. In determining the results, each of the cities acted as starting points from where the route started and ended. The algorithms have therefore been shown to work for different vertices for good evaluation for their effectiveness in general.

| **Towns title** | **Greedy** | | **ACO** | | **EAS** | |
|---|---|---|---|---|---|---|
| | distance, km | fluel price, euro | distance, km | fluel price, euro | distance, km | fluel price, euro |
| Wesel | 181 | 53,38 | 166 | 48,78 | 166 | 48,78 |
| Dinslaken | 189 | 55,62 | 167 | 49,32 | 166 | 48,79 |
| Hamminkeln | 183 | 53,98 | 167 | 49,32 | 166 | 48,79 |
| Kamp-Lintfort | 176 | 52,06 | 169 | 49,86 | 166 | 48,79 |
| Moers | 181 | 53,54 | 168 | 49,71 | 166 | 48,79 |
| Neukirchen-Vluyn | 198 | 58,32 | 167 | 49,32 | 166 | 48,79 |
| Rheinberg | 210 | 61,83 | 166 | 48,79 | 166 | 48,79 |
| Voerde | 192 | 56,67 | 166 | 48,79 | 166 | 48,79 |
| Xanten | 181 | 53,31 | 167 | 49,32 | 166 | 48,79 |
| Alpen | 205 | 60,38 | 167 | 49,15 | 166 | 48,79 |
| Hünxe | 179 | 52,48 | 166 | 48,79 | 166 | 48,79 |
| Schermbeck | 179 | 52,48 | 166 | 48,79 | 166 | 48,79 |
| Sonsbeck | 186 | 54,77 | 168 | 49,71 | 166 | 48,79 |

Table 6 – Combined Java outputs

The table contains data on distances in kilometers and fuel prices in euros. For the purpose of analysis, the numerical totals are divided into 3 groups: in group 1 are located the results of Greedy model calculations, Group 2: results of ACO algorithm, Group 3: results of EAC method implementations.

To start with the analysis, the obtained data should be examined according to the question of which group of classes heuristic or metaheuristic showed the best performance in the field of optimization. When we start comparing the data, we can see that the results are small in the considered region. However, as mentioned in Chapter 5.1 this fact was known in advance, as the chosen region does not have a long distance between towns. Despite this factor, the difference between the considered algorithms is significant and can be seen from the Table 6.

Comparing criterion of the distance on the created routes it can be seen how the ACO model and the EAS version exceeds the Greedy method. The distance between cities with Greedy model takes on average 180-200 kilometers. In contrast, the ACO method shows a kilometrage on average166-170 kilometers. For a more detailed view of how the routes

have been shorten, let us present the data in terms of percentage difference of optimized routes were combined in Table 7. By detailed reviewing at the distance factor, the smallest difference is observed in the town of Kamp-Lintfort with 4.0%, however, in the remaining towns the difference is more noticeable and averages improvement was in the interval between 7.2% and 9.7%. The most markable difference is achieved in the intervals from 15.7% - 21.0%. The difference between these rates demonstrates how the ACO model was able to improve overall kilometers and reduce the total length for all town in the region.

| **Difference between Greedy and ACO** | | |
|---|---|---|
| **Towns title** | **distance, %** | **fluel price, %** |
| Wesel | 8,3% | 8,6% |
| Dinslaken | 11,6% | 11,3% |
| Hamminkeln | 8,7% | 8,6% |
| Kamp-Lintfort | 4,0% | 4,2% |
| Moers | 7,2% | 7,2% |
| Neukirchen-Vluyn | 15,7% | 15,4% |
| Rheinberg | 21,0% | 21,1% |
| Voerde | 13,5% | 13,9% |
| Xanten | 7,7% | 7,5% |
| Alpen | 18,5% | 18,6% |
| Hünxe | 7,3% | 7,0% |
| Schermbeck | 7,3% | 7,0% |
| Sonsbeck | 9,7% | 9,2% |

Table 7 - Percentage difference in results

Comparing criterion of the fuel cost, it is important to realize that each route is built individually with its own unique set of town sequences. The calculated cost depends on this sequence, in some cases the calculated cost may differ from each other even for the same kilometers. Greedy Algorithm shows a cost in the range of 52.06 euro - 61.83 euro, for ACO algorithm 48.78 euro – 49.86 euro. In percentage terms, the smallest difference is noticeable in Kamp-Lintfort 4.2%, the average difference between values is observed in the interval from 7.0%-11.3%. The highest difference is achieved from 13.9% to 21.1%. This confirms the fact that it is possible to reduce the costs expenses, as in this research regarding the fuel costs.

It should be noted that the difference between the results of ACO and EAS is not very significant and in some cases, it shows the same result. In fact of the output results, it can be seen how the EAS has even improved the ACO routes. Based on the Table 6, the EAS

model demonstrates stable results to all towns within the region with the lowest kilometrage of 166 kilometers. The minimum result for the fuel cost can be seen between 48.78 euro and 48.79 euro. The EAS model shows stable optimization performance throughout the whole process of finding the optimal solution for the given data set.

The simulation outputs have proved that even for small numerical values, the metaheuristic methods in the form of ACO with EAS showed a better result in finding the shortest path and reducing the cost. At the same time the algorithms showed flexibility and remained robust in the results when the start and end vertices were changed. The efficiency of the proposed solution for the determined parameters have increased and the simulation model realized in Java could be used in further experiments.

# 6. Conclusion

## 6.1 Overall research findings

To summarize the overall research results of the ACO algorithm, many aspects of the method's performance have been studied and demonstrated, including both the theoretical fundamentals and the practical application through a developed case study. In the thesis paper the basic concepts were reviewed, also theory was considered the step-by-step operations of both observed algorithms. The Greedy Algorithm was used as a method to compare and determine the working performance of the ACO. Additionally, in the thesis an improved EAS version of the basic ACO method was introduced with its mathematical model on how virtual elite ants are formed.

The thesis paper also conducted a study on the role of parameters in the ACO method combined with EAS. The parameters we performed and their relation to the effect in implementation of the optimization process for a gathered data set. In this section it was determined the coefficients in order to reach the optimality in efficient functioning of the algorithms. Then, the overall gather information was simulated according to test the functionality based on the self-developed tool.

On the basis of the mathematical expressions of the observed methods, the software tool was created in Java language. With the help of which it was possible to realize a simulation experiment according to the actual collected data in the Wesel region. The solution of the logistic problem using the ACO method demonstrated that the created program shows stable solutions and can be adapted to changing parameters. Simulation showed us that the method copes with the optimization of a set of facilities as in finding

an efficient route and reducing the costs. Testing was carried out in an environment where information on traffic congestion, time windows and other elements affecting the existing road conditions were not taken into account.

At the same time, the flow of the experiment showed that EAS also improved the overall results when elite ants were introduced. EAS during the entire experiment displayed unchanged and regular numerical performance. Moreover, it was demonstrated that with the right number of elite ants, has a sustainable function in the overall system. At the same time constantly correctly and accurately indicating the shortest route lengths with the corresponding fuel cost.

All research questions have been covered and explained in the thesis paper. The basic principles of algorithms were revealed, parameters and variables of influence on the ACO group were considered and studied. The created simulation model demonstrated the functioning of ACO in reality and on the basis of the routing problem even under incomplete environmental conditions. At the same time, the performance of the ACO algorithm with the EAS was demonstrated on small values in the region. The effectiveness is visible due to the comparison with the Greedy algorithm. Greedy method was acting as competitive in terms of computation speed. On the other hand, the ACO model solves more complex and fundamental problems, the results of which are credible and might be used in the professional sphere.

The objectives of the thesis during the research were achieved and the functionality of the ACO algorithm was tested on a logistic developed example. The created software model is ready to use and prepared for future testing. This research can be served as a basic point for further experiments by using that developed tool with other variables and input data.

### 6.2 Future plans and ideas

In this thesis an important step in the study of ACO algorithm was understanding the functionality of the algorithm and what results ACO would demonstrate. In the research work it was possible to find optimal routes, optimize the total length and reduce the fuel costs. It should be noted that in general terms, the reduction of distance is one of the most important parameters in improving relation between cities, because it can directly affect to 3 indicators at once: time, economic performance and reduction of emissions into the environment. However, travelling time does not always correlate with the distance. For example, during transportation process there may be situations in which a short distance

may cause a different and higher travelling time. This may happen in the context of emergency situations, traffic congestion and changing or adjustments to an existing route.

Distance also has a direct impact on costs as well. In this thesis paper, the performance of ACO in terms of reducing fuel costs was tested and, in some towns, it helped to reduce costs by 13.9% to 21.1%. This provides an opportunity to introduce in future other relevant economic data into the existing model such as total cost of transportation, accounting for insurance costs, taxes, total spend value and more other factors.

The influence of distance on emissions is not quite unambiguous and does not have a direct correlation. But we can assume that obvious then the shorter time of transport engine works - the lower emission level of harmful substances is in the air. An important role there plays the traffic congestion caused by the formation of road accidents on sections of the road network infrastructure. Traffic congestions length can be many kilometers long, so that the amount of exhaust gases significantly increases. These figures will depend on the type of road and the geographical location of the research study area.

At the same time, all the above-mentioned extended proposals can be implemented on the existing developed Java program model. The implementation itself can be realized in different ways, without changing the program structure itself, but by setting new input data in Input class.

For the further realization, in order to implement next the problem statements for the deliveries where the goods will have to be transported in certain time intervals, an input matrix has to be created. The matrix should take into account in what exact intervals the goods should be transported to the customer or to other distribution centers. For that purposes the corresponding coefficients are need to be created. Moreover, it should be reflected to the delivery time or to denote a certain interval, in which the goods must be delivered to a particular city or to a given area. By multiplying the distances by the created coefficients, we will not only set time windows, but also be able to set the priority in the directions where, for example, transporting goods need to be delivered first. When the algorithms start their work, the process of finding the optimal solution will take into account this important condition, to which city by prioritization the transport should go first and will consider whether the delivery fits into this frame intervals or not.

The implementation of weighting multipliers coefficients can be created in such way, there the time interval reflecting to a delivery process. When it needs to be processed the interval range should be divided by the total number of working days. It means we can

consider when the staff are working and in a such logic a multiplier values are obtained. In the integrated process, the indicators can either be served as a separate input matrix or be already concluded into a common matrix with the corresponding numerical distance data.

In considering another aspect the further mechanism should be noted. Implementing the traffic situation, it is not possible to fully predict the traffic congestion without GPS systems that would illuminate the online traffic situation. However, it is also possible to select weighting factors that will denote where and between which sections traffic congestion is observed. In searching for optimal routes, the ACO algorithm will take this condition into account. In the input data we can also include the road type information between the cities. Also, to take into the matrix the data of the geographical conditions and specify the terrain in the regions.

Another option is how to consider traffic jams is to use another logic. In this thesis research, the calculation of fuel consumption was taken as 17 liters of fuel per 100 kilometers. Assuming that in the study area if there are certain sections or regions where congestion is collected, then fuel consumption decreases in the moments of waiting time in traffic jams. As a consequence, it is possible to observe the congestions through the fuel consumption on different sections of the road between cities or in another area. Then in the input data there may be an additional matrix that takes into account these features.

In the ACO model the time to certain cities can also be considered regarding the developed program. The time can be combined with the traffic jams in a such way, where the traveling time might be added into the input matrix with the average speed on the routes. In that case the traffic congestions can also be observed. It means that after calculating the average speed and collecting distance data, the input data will include the set of the travelling time. The realization of calculation will be developed by dividing the distance by the average speed. In that case by the calculation get the time for which the transport will reach the points of interest. These operations are another proposed action of how to use new data in order to optimize routes and also take into account the type of road and terrain region in the existing Java model.

In a such manner the input data models can create a more complex task for the further research and different set conditions can be added to the program. In this thesis a basic ACO model has been developed and even in the first version of the program code each result reflected to be an improved. On the basis of the written code, it is possible to solve

the tasks such as searching the right time intervals, route optimization and to determine in which direction it is more profitable to travel and at the same time to reduce the total cost.

Into the ACO model these suggested different options can be added in the future work. In that case a good calculator can be created and also applied in a real work experiment. As it was showed, the program in Java is not concentrated to a particular data. The input parameters can be different according to the new designed tasks. The above options were suggested in order to present how the developed tool might be working without changing the program code itself. It was described on how with the help of input data ACO model can be performed in the future work and calculations.

In the future experiments the model itself can be improved. As was already mentioned the idea for the ACO is to create a model that would take into account the real traffic situation with the help of GPS. Another suggestion is to implement in the model the use of artificial intelligence (AI). The use of AI would also increase the efficiency and help the program quickly react to a new incoming updated information. In order to develop such calculator, it is necessary to understand how the new suggested tools are working. It should be studied in details of how to implement these ideas into a code and how the ACO algorithm will be working in a new area. These are uncovered future points for the further research work.

## 7. References


Banks, J. (2014). Discrete-event system simulation. Harlow: Pearson.

Bell, E., Bryman, A. and Harley, B. (2022). Business Research Methods. 6th ed. Oxford: Oxford University Press.

Bhavya, R. and Elango, L. (2023). Ant-Inspired Metaheuristic Algorithms for Combinatorial Optimization Problems in Water Resources Management. Water, [online] 15(9), p.1712. doi:https://doi.org/10.3390/w15091712.

Bickman, L. and Rog, D. (n.d.). Applied Research Design A Practical Approach Planning Applied Social Research. [online] Available at: https://us.sagepub.com/sites/default/files/upm-binaries/23770_Ch1.pdf.

Birattari, M., Caro, G.D. and Dorigo, M. (2000). For a formal foundation of the Ant Programming approach to combinatorial optimization: Part 1: The problem, the representation, and the general solution strategy. Technical Report TR-H-301, ATR-Human Information Processing Labs, Kyoto, Japan.

Bolbakov, R.G., Sinitsyn, A.V. and Tsvetkov, V.Y. (2020). Methods of comparative analysis. Journal of Physics: Conference Series, 1679(5), p.052047. doi:https://doi.org/10.1088/1742-6596/1679/5/052047.

Burnham, A., Gohlke, D., Rush, L., Stephens, T., Zhou, Y., Delucchi, M., Birky, A., Hunter, C., Lin, Z., Ou, S., Xie, F., Proctor, C., Wiryadinata, S., Liu, N. and Boloor, M. (2021). Comprehensive Total Cost of Ownership Quantification for Vehicles with Different Size Classes and Powertrains. doi:https://doi.org/10.2172/1780970.

Christopher, M. (2016). Logistics and Supply Chain Management ePub eBook. Pearson UK.

Colorni, A., Dorigo, M., Vittorio Maniezzo, Varela, F. and P. Bourgine (1992). Distributed Optimization by Ant Colonies. European Conference on Artificial Life, pp.134–142.

Dang, Y., Singh, M. and Allen, T.T. (2021). Network Mode Optimization for the DHL Supply Chain. INFORMS Journal on Applied Analytics, 51(3). doi:https://doi.org/10.1287/inte.2020.1046.

Dorigo (1992) Optimization Learning and Natural Algorithms. Unpublished PhD Thesis, Dipartimento di Eletronica, Politecnico di Milano, Milano.

Dorigo, M. and Gambardella, L.M. (1997). Ant colonies for the travelling salesman problem. Biosystems, 43(2), pp.73–81. doi:https://doi.org/10.1016/s0303-2647(97)01708-5.

Dorigo, M. and StützleT. (2004). Ant colony optimization / Ant colony optimization. Cambridge, Mass.: Mit Press.

Eldosouky, I.A., Ibrahim, A.H. and Mohammed, H.E.-D. (2014). Management of construction cost contingency covering upside and downside risks. Alexandria

Engineering Journal, [online] 53(4), pp.863–881. doi:https://doi.org/10.1016/j.aej.2014.09.008.

Florez, E., Gomez, W. and Lola Bautista, MSc. (2013). An Ant Colony Optimization Algorithm For Job Shop Scheduling Problem. International Journal of Artificial Intelligence & Applications, 4(4), pp.53–66. doi:https://doi.org/10.5121/ijaia.2013.4406.

Gaertner, D. and Clark, K.L. (2005). On Optimal Parameters for Ant Colony Optimization Algorithms. Proceedings of the 2005 International Conference on Artificial Intelligence, ICAI 2005, Las Vegas, Nevada, USA, June 27-30, 2005, Volume 1, [online] 1, pp.83–89. Available at: https://www.researchgate.net/publication/220835272_On_Optimal_Parameters_for_Ant_Colony_Optimization_Algorithms.

Goel, A. and Gruhn, V. (2008). A General Vehicle Routing Problem. European Journal of Operational Research, 191(3), pp.650–660. doi:https://doi.org/10.1016/j.ejor.2006.12.065.

Google (2024). Google Maps. [online] Google Maps. Available at: https://www.google.com/maps.

Gosling, J., Joy, B., Steele, G.L., Bracha, G. and Buckley, A. (2015). The Java Language Specification, Java SE 8 Edition. Addison-Wesley Professional.

Guo, S. (2023). Research on Logistics Distribution Route Optimization Based on Ant Colony Algorithm. pp.84–88. doi:https://doi.org/10.1109/eebda56825.2023.10090583.

Han, T., Foo, G., Yi, C., Yee, B., Zer, K., Wong, J., Jie, Lun, N., Arabee, Z. and Salam, A. (2023). Optimizing ACO Algorithm for the TSP Through Parameter Modification. Journal of Applied Technology and Innovation, [online] 7(3), p.52. Available at: https://jati.sites.apiit.edu.my/files/2023/07/Volume7_Issue3_Paper9_2023.pdf [Accessed 2 Apr. 2025].

Jirawat Thongpiem, Narong Punkong, Chiabwoot Ratanavilisagul and Somkiat Kosolsombat (2024). Hybrid Genetic Algorithm and Ant Colony Algorithm for Solving Travelling Salesman Problem. IEEE 9th International Conference on Computational Intelligence and Applications , pp.69–73. doi:https://doi.org/10.1109/iccia62557.2024.10719116.

Kabir SMS (2016) Basic guidelines for research: An introductory approach for all disciplines. Zone Publication, 4(2), 168-180.

Kizilateş, G. and Nuriyeva, F. (2013). On the Nearest Neighbor Algorithms for the Traveling Salesman Problem. Advances in Intelligent Systems and Computing, [online] pp.111–118. doi:https://doi.org/10.1007/978-3-319-00951-3_11.

Kolkmann, M., Tebest , Dr.R., Westrich, H. and Petroff , U. (2023). Kreis Wesel Örtliche Planung nach dem Alten-und Pflegegesetz NRW. [online] Available at: https://www.kreis-wesel.de/system/files/2023-11/Pflegeplan_des_Kreises_Wesel.pdf [Accessed 3 Mar. 2025].

Kossmann, D. and Stocker, K. (2000). Iterative dynamic programming. ACM Transactions on Database Systems, 25(1), pp.43–82. doi:https://doi.org/10.1145/352958.352982.

Kuldeep Singh Kaswan, Jagjit Singh Dhatterwal and Kumar, A. (2023). Swarm Intelligence. John Wiley & Sons.

Li, N. and Yu, Y. (2022). Design and Application of Improved Ant Colony Algorithm in E-Commerce System. Wireless Communications and Mobile Computing, 2022, pp.1–12. doi:https://doi.org/10.1155/2022/9467095.

Liu, D., Hu, X. and Jiang, Q. (2023). Design and optimization of logistics distribution route based on improved ant colony algorithm. Optik, 273, p.170405. doi:https://doi.org/10.1016/j.ijleo.2022.170405.

Ma, X. and Liu, C. (2024). Improved Ant Colony Algorithm for the Split Delivery Vehicle Routing Problem. Applied Sciences, 14(12), pp.5090–5090. doi:https://doi.org/10.3390/app14125090.

Maintenance and service product recommendation 2024 MAN Truck and MAN Individual. (n.d.). Available at: https://public.man.eu/media/service/asp/media/en/auftritte/service_dokumente_/wartungs__und_betriebsstoffvorschrift/bezugsbremskraefte_2/Maintenance_and_service_product_recommendation.pdf [Accessed 10 Feb. 2025].

Mccool, M., Arch Robison and Reinders, J. (2012). Structured parallel programming : patterns for efficient computation. Waltham, Ma: Morgan Kaufmann.

Mehta, H., Kanani, P. and Lande, P. (2019). Google Maps. International Journal of Computer Applications, [online] 178(8), pp.41–46. doi:https://doi.org/10.5120/ijca2019918791.

Mishra, S.B. and Alok, S. (2022). Handbook of Research Methodology. Educreation Publishing

Omid Bozorg-Haddad (2017). Advanced Optimization by Nature-Inspired Algorithms. Springer.

Pasquier, J.L., Balich, I.K., Carr, D.W. and Cuauhtémoc López-Martín (2007). A Comparative Study of Three Metaheuristics Applied to the Traveling Salesman Problem. pp.243–254. doi:https://doi.org/10.1109/micai.2007.14.

Remler, D.K. and Van, G.G. (2014). Research Methods in Practice. SAGE Publications.

Pop, P.C. (2012). Generalized Network Design Problems. Walter de Gruyter.

Rizzoli, A.E., Montemanni, R., Lucibello, E. and Gambardella, L.M. (2007). Ant colony optimization for real-world vehicle routing problems. Swarm Intelligence, 1(2), pp.135–151. doi:https://doi.org/10.1007/s11721-007-0005-x.

Roberto, P., Da Costa, O., Rhuggenaath, J., Zhang, Y., Akcay, A., Pan, S. and Sugiyama, M. (2020). Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep

Reinforcement Learning. Proceedings of Machine Learning Research, [online] 129, pp.465–480. Available at: https://proceedings.mlr.press/v129/costa20a/costa20a.pdf [Accessed 19 Feb. 2025].

Salam, Z., Ahmed, J. and Merugu, B.S. (2013). The application of soft computing methods for MPPT of PV system: A technological and status review. Applied Energy, 107, pp.135–148. doi:https://doi.org/10.1016/j.apenergy.2013.02.008.

Samoylov, N. (2018). Introduction to programming : learn to program in Java with data structures, algorithms, and logic. Birmingham: Packt.

Samson Ejim (2016). Implementation of Greedy Algorithm in Travel Salesman Problem. ResearchGate. [online] doi:https://doi.org/10.13140/RG.2.2.23921.48485.

Selvi, V. and Umarani, Dr.R. (2010). Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques. International Journal of Computer Applications, 5(4), pp.1–6. doi:https://doi.org/10.5120/908-1286.

Shrivastava, K. and Kumar, S. (2019). The Effectiveness of Parameter Tuning on Ant Colony Optimization for Solving the Travelling Salesman Problem. The International Conference on Communication Systems and Network Technologies (CSNT2018). [online] doi:https://doi.org/10.1109/CSNT.2018.17.

Solnon, C. and Firm, P. (2010). Ant colony optimization and constraint programming. London: Iste. Available online via HSRW library from: https://ebookcentral.proquest.com/lib/hrw/reader.action?docID=1143621/ [Accessed: 20 November 2024].

Tankerkoenig.de. (2025a). Tankerkönig Homepage. [online] Available at: https://tankerkoenig.de/ [Accessed 11 Feb. 2025].

Tankerkoenig.de. (2025b). Tankerkönig Homepage. [online] Available at: https://www.tankerkoenig.de/ [Accessed 12 Feb. 2025].

Toloo, M., Talatahari, S. and Rahimi, I. (2022). Multi-objective combinatorial optimization problems and solution methods. London: Academic Press.

Tomar, V., Bansal, M. and Singh, P. (2024a). Metaheuristic Algorithms for Optimization: A Brief Review. Engineering Proceedings, [online] 59(1), p.238. doi:https://doi.org/10.3390/engproc2023059238.

Viswanathan, V., Sen, A. and Chakraborty, S. (2011). Stochastic Greedy Algorithms A Learning-Based Approach to Combinatorial Optimization. [online] Available at: https://www.thinkmind.org/articles/soft_v4_n12_2011_1.pdf [Accessed 25 Mar. 2025].

Wang, Y. (2023). Review on greedy algorithm. Theoretical and natural science, 14(1), pp.233–239. doi:https://doi.org/10.54254/2753-8818/14/20241041.

Wei, X. (2014). Parameters Analysis for Basic Ant Colony Optimization Algorithm in TSP. International Journal of u- and e-Service, Science and Technology, 7(4), pp.159–170. doi:https://doi.org/10.14257/ijunesst.2014.7.4.16.

Wu, Y. and Wang, J. (2018). Algorithm Design Practice for Collegiate Programming Contests and Education. CRC Press.

Yang, X.-S. (2014). Nature-inspired optimization algorithms. London [England]: Elsevier.
Yue, Y. meng and Wang, X. (2015). An Improved Ant Colony Optimization Algorithm for Solving TSP. International Journal of Multimedia and Ubiquitous Engineering, 10(12), pp.153–164. doi:https://doi.org/10.14257/ijmue.2015.10.12.16.

Zarei, H., Yousefikhoshbakht, M. and Khorram, E. (2016). A hybrid modified meta-heuristic algorithm for solving the traveling salesman problem. [online] Journal of Industrial and Systems Engineering Vol. 9, No. 3, pp 57-69 . Available at: https://www.jise.ir/article_13968_da70337d040a93df6acf54667bfa7d5c.pdf [Accessed 19 Feb. 2025].

Zang, H., Zhang, S. and Hapeshi, K. (2010). A Review of Nature-Inspired Algorithms. *Journal of Bionic Engineering*, 7(S4), pp.S232–S237. doi:https://doi.org/10.1016/s1672-6529(09)60240-7.

Zhanar Bimurat, Kim, Y., Rauza Ismailova and Bimurat Sagindykov (2023). METHODS OF NAVIGATING ALGORITHMIC COMPLEXITY: BIG-OH AND SMALL-OH NOTATIONS. Scientific journal of Astana IT University (Print), [online] pp.160–181. doi:https://doi.org/10.37943/15dnlb5877.

# 8. Appendices

## 8.1 Parameters determination

| Q calculation | | |
|---|---|---|
| **Q** | **ACO Distance** | **EAS Distance** |
| 50 | 172,7546 | 172,04137 |
| 100 | 172,82015 | 172,1677 |
| 150 | 172,91816 | 172,23185 |
| 200 | 172,866 | 172,24396 |
| 250 | 172,87567 | 172,19036 |
| 300 | 172,94401 | 172,34418 |
| 350 | 173,04554 | 172,32777 |
| 400 | 173,04932 | 172,30915 |
| 450 | 173,01765 | 172,30214 |
| 500 | 173,05934 | 172,28996 |
| 550 | 172,98201 | 172,33224 |
| 600 | 173,00925 | 172,27772 |
| 650 | 173,05237 | 172,34266 |
| 700 | 173,16576 | 172,4265 |
| 750 | 173,10176 | 172,37037 |
| 800 | 173,08127 | 172,35901 |
| 850 | 173,07677 | 172,43003 |
| 900 | 173,20906 | 172,38607 |
| 950 | 173,06767 | 172,4443 |
| 1000 | 173,12834 | 172,46914 |

Table 8 - Q results calculation

| Beta calculation | | |
|---|---|---|
| **Beta** | **ACO Distance** | **EAS Distance** |
| 0,5 | 176,93515 | 175,6466 |
| 1 | 176,98921 | 176,9074 |
| 1,5 | 174,57741 | 174,8544 |
| 2 | 172,88641 | 172,8679 |
| 2,5 | 172,25298 | 171,8103 |
| 3 | 171,90914 | 171,0935 |
| 3,5 | 171,7244 | 170,8335 |
| 4 | 171,81385 | 170,6122 |
| 4,5 | 171,9622 | 170,6317 |
| 5 | 172,13895 | 170,7575 |

Table 9 - Beta results calculation

| Number of elite ants calculation | |
|---|---|
| **Elite Ants Count** | **Distance** |
| 1 | 166 |
| 2 | 166 |
| 3 | 166 |
| 4 | 166 |
| 5 | 167 |
| 6 | 167 |
| 7 | 167 |
| 8 | 170 |

Table 10 - EAS results calculation

| Evaporation calculation | | | | | |
|---|---|---|---|---|---|
| Evaporation | ACO Distance | EAS Distance | Evaporation | ACO Distance | EAS Distance |
| 0,1 | 168,50676 | 168,12268 | 5,1 | 173,1353 | 172,5726 |
| 0,2 | 169,35003 | 168,81472 | 5,2 | 173,4713 | 172,5915 |
| 0,3 | 170,06335 | 169,28557 | 5,3 | 173,0434 | 172,2648 |
| 0,4 | 170,51489 | 169,79202 | 5,4 | 173,3163 | 172,346 |
| 0,5 | 171,0723 | 170,22607 | 5,5 | 173,1291 | 172,4803 |
| 0,6 | 171,36415 | 170,53364 | 5,6 | 173,333 | 172,3368 |
| 0,7 | 171,74443 | 170,90565 | 5,7 | 173,1227 | 172,4885 |
| 0,8 | 172,20157 | 171,22887 | 5,8 | 173,1499 | 172,6056 |
| 0,9 | 172,61155 | 171,53524 | 5,9 | 173,3089 | 172,6716 |
| 1 | 173,27556 | 172,64625 | 6 | 173,4594 | 172,796 |
| 1,1 | 173,20765 | 172,49126 | 6,1 | 173,4554 | 172,5653 |
| 1,2 | 173,1967 | 172,44231 | 6,2 | 173,071 | 172,8685 |
| 1,3 | 173,37084 | 172,67777 | 6,3 | 173,4807 | 172,612 |
| 1,4 | 173,07393 | 172,47317 | 6,4 | 173,2737 | 172,6458 |
| 1,5 | 173,18231 | 172,39847 | 6,5 | 173,1916 | 172,6481 |
| 1,6 | 173,29715 | 172,48136 | 6,6 | 173,3297 | 172,6981 |
| 1,7 | 173,35769 | 172,55616 | 6,7 | 173,2122 | 172,8352 |
| 1,8 | 173,0104 | 172,47072 | 6,8 | 173,086 | 172,6362 |
| 1,9 | 173,25328 | 172,52681 | 6,9 | 173,1691 | 172,4378 |
| 2 | 173,47337 | 172,61304 | 7 | 173,2508 | 172,7383 |
| 2,1 | 173,28147 | 172,37713 | 7,1 | 173,239 | 172,539 |
| 2,2 | 172,94344 | 172,67161 | 7,2 | 173,3027 | 172,5668 |
| 2,3 | 173,41221 | 172,79444 | 7,3 | 173,2193 | 172,2802 |
| 2,4 | 173,12739 | 172,58885 | 7,4 | 173,0594 | 172,2677 |
| 2,5 | 173,21856 | 172,59638 | 7,5 | 173,2593 | 172,4678 |
| 2,6 | 173,23838 | 172,47458 | 7,6 | 173,034 | 172,6129 |
| 2,7 | 173,29451 | 172,7456 | 7,7 | 173,1188 | 172,6623 |
| 2,8 | 173,20283 | 172,65579 | 7,8 | 173,6416 | 172,6411 |
| 2,9 | 173,25232 | 172,5385 | 7,9 | 173,169 | 172,3544 |
| 3 | 173,24164 | 172,51836 | 8 | 173,2787 | 172,3463 |
| 3,1 | 173,2269 | 172,49371 | 8,1 | 173,0614 | 172,5279 |
| 3,2 | 173,26284 | 172,43005 | 8,2 | 173,2725 | 172,6051 |
| 3,3 | 173,36393 | 172,68689 | 8,3 | 173,559 | 172,6177 |
| 3,4 | 173,17166 | 172,40926 | 8,4 | 173,3118 | 172,6649 |
| 3,5 | 173,2187 | 172,54347 | 8,5 | 173,1146 | 172,4615 |
| 3,6 | 173,49918 | 172,56403 | 8,6 | 173,2638 | 172,6002 |
| 3,7 | 173,1181 | 172,6561 | 8,7 | 173,0159 | 172,5214 |
| 3,8 | 173,30516 | 172,53074 | 8,8 | 173,3492 | 172,4659 |
| 3,9 | 173,35749 | 172,45123 | 8,9 | 173,1779 | 172,37 |
| 4 | 173,12411 | 172,50957 | 9 | 173,2553 | 172,5134 |
| 4,1 | 173,42249 | 172,54312 | 9,1 | 173,4089 | 172,5025 |
| 4,2 | 173,35468 | 172,65298 | 9,2 | 173,2714 | 172,7368 |
| 4,3 | 173,26917 | 172,59924 | 9,3 | 173,253 | 172,6658 |
| 4,4 | 173,24741 | 172,43052 | 9,4 | 173,4326 | 172,6667 |
| 4,5 | 173,22829 | 172,73839 | 9,5 | 173,1873 | 172,7346 |
| 4,6 | 173,08886 | 172,28799 | 9,6 | 173,1686 | 172,3924 |
| 4,7 | 173,1209 | 172,62691 | 9,7 | 173,175 | 172,4858 |
| 4,8 | 173,38048 | 172,64419 | 9,8 | 173,3117 | 172,6634 |
| 4,9 | 173,27451 | 172,5305 | 9,9 | 172,9224 | 172,5787 |
| 5 | 173,10181 | 172,5353 | 10 | 173,3762 | 172,6102 |

Table 11 - Evaporation results calculation

## 8.2 Java application code

### 8.2.1 Main.java

```
    public static double calculateTotalCost(double distance, double
fuel_cost, double fuel_consumption) {

        return 1.05 * distance * fuel_cost * fuel_consumption / 100.0
;
    }

    public static void main(String[] args) {
        Input input = new Input();
        int rows = input.distances.length;
        int cols = input.distances[0].length;
        double[][] total_costs = new double[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                total_costs[i][j] =
calculateTotalCost(input.distances[i][j], input.fuel_costs[i][j],
input.fuel_consumption) ;
            }
        }
        List<String> cities = new ArrayList<>(List.of( "Wesel",
"Dinslaken", "Hamminkeln", "Kamp-Lintfort","Moers",
                "Neukirchen-Vluyn", "Rheinberg", "Voerde", "Xanten",
"Alpen", "Hünxe", "Schermbeck", "Sonsbeck"));

        int startCityIndex = cities.indexOf("Sonsbeck");
        int numCities = input.distances.length;
        // Path
        ShortRouteAlgorithm shortRoute = new ShortRouteAlgorithm();
        List<Integer> shortRoutePath = shortRoute.calculate(numCities,
input.distances, startCityIndex );
        System.out.print("Short Distance Algorithm route:" );
        for (Integer integer : shortRoutePath) {
            System.out.print(cities.get(integer) + "->");
        }
        System.out.print(cities.get(startCityIndex));
        System.out.println();
        double totalDistance =
shortRoute.calculateTotalValue(shortRoutePath, input.distances);
        double shortPathPrice = 0.0;
        for (int i = 0; i < shortRoutePath.size() - 1; i++) {
            shortPathPrice +=
calculateTotalCost(input.distances[shortRoutePath.get(i)][shortRoutePa
th.get(i+1)],
input.fuel_costs[shortRoutePath.get(i)][shortRoutePath.get(i+1)]
,input.fuel_consumption);
        }
        shortPathPrice +=
calculateTotalCost(input.distances[shortRoutePath.get(shortRoutePath.s
ize() - 1)][shortRoutePath.get(0)],
input.fuel_costs[shortRoutePath.get(shortRoutePath.size() -
1)][shortRoutePath.get(0)], input.fuel_consumption);

        System.out.println(String.format("Total distance is: %.1f;
fuel cost is: %.2f Euro", totalDistance, shortPathPrice));

        AlgorithmParameters params = new AlgorithmParameters();
        AntColonyOptimizationAlgorithm aco = new
AntColonyOptimizationAlgorithm(numCities, input.distances, params);
```

```
        List<Integer> path = aco.calculate(startCityIndex);
        System.out.print("Optimized Ant Colony Optimization Route:" );
        for (Integer integer : path) {
            System.out.print(cities.get(integer) + "->");
        }
        System.out.print(cities.get(startCityIndex));
        double acoPrice = 0.0;
        for (int i = 0; i < path.size() - 1; i++) {
            acoPrice +=
calculateTotalCost(input.distances[path.get(i)][path.get(i+1)],
input.fuel_costs[path.get(i)][path.get(i+1)], input.fuel_consumption);
        }
        acoPrice +=
calculateTotalCost(input.distances[path.get(path.size() -
1)][path.get(0)], input.fuel_costs[path.get(path.size() -
1)][path.get(0)], input.fuel_consumption);
        double acoDistance = aco.getTotalValueForRoute(path);

        AntColonyOptimizationAlgorithm acoF = new
AntColonyOptimizationAlgorithm(numCities, total_costs, params);
        double acoPriceCalculated =
acoF.getTotalValueForRoute(acoF.calculate(startCityIndex));

        System.out.println();
        System.out.println(String.format("Total distance is: %.1f;
fuel cost is: %.2f Euro", acoDistance, acoPriceCalculated));

        int eliteAntsCount = 4;
        ElitistAntSystem eas = new ElitistAntSystem(numCities,
input.distances, params, eliteAntsCount);
        List<Integer> pathEas = eas.calculate(startCityIndex);
        System.out.print("Elite Ants Colony Optimization Route:" );
        for (Integer integer : pathEas) {
            System.out.print(cities.get(integer) + "->");
        }
        System.out.print(cities.get(startCityIndex));
        double totalDistanceEas = eas.getTotalValueForRoute(pathEas);

        double easPrice = 0.0;
        for (int i = 0; i < path.size() - 1; i++) {
            easPrice +=
calculateTotalCost(input.distances[pathEas.get(i)][pathEas.get(i+1)],
input.fuel_costs[pathEas.get(i)][pathEas.get(i+1)],
input.fuel_consumption);
        }
        easPrice +=
calculateTotalCost(input.distances[pathEas.get(pathEas.size() -
1)][pathEas.get(0)], input.fuel_costs[pathEas.get(pathEas.size() -
1)][pathEas.get(0)], input.fuel_consumption);

        ElitistAntSystem easF = new ElitistAntSystem(numCities,
total_costs, params, eliteAntsCount);
        double eafPriceCalculated =
easF.getTotalValueForRoute(easF.calculate(startCityIndex));

        System.out.println();
        System.out.println(String.format("Total distance is: %.1f;
fuel cost is: %.2f Euro", totalDistanceEas, eafPriceCalculated));

    }
}
```

### 8.2.2 ShortRouteAlgorithm.java

```
import java.util.ArrayList;
import java.util.List;

public class ShortRouteAlgorithm {
    public static List<Integer> calculate(int numCities, double[][]
values, int startCityIndex) {
        List<Integer> route = new ArrayList<>();
        boolean[] visited = new boolean[numCities];
        visited[startCityIndex] = true;
        route.add(startCityIndex);
        int currentCityIndex = startCityIndex;
        while (route.size() < numCities) {
            int nextCityIndex = -1;
            double minValue = Double.MAX_VALUE;

            for (int i = 0; i < numCities; i++) {
                if (!visited[i] && values[currentCityIndex][i] <
minValue) {
                    minValue = values[currentCityIndex][i];
                    nextCityIndex = i;
                }
            }
            visited[nextCityIndex] = true;
            route.add(nextCityIndex);
            currentCityIndex = nextCityIndex;
        }
        return route;
    }

    public double calculateTotalValue(List<Integer> route, double[][]
values) {
        double total = 0;
        for (int i = 0; i < route.size() - 1; i++) {
            int cityFromIndex = route.get(i);
            int cityToIndex = route.get(i + 1);
            total += values[cityFromIndex][cityToIndex];
        }
        total += values[route.get(route.size() - 1)][route.get(0)];
        return total;
    }
}
```

### 8.2.3 AlgorithmParameters.java

```
public class AlgorithmParameters {
    // Constants for the ACO algorithm

    public double Alpha = 1.0;
    public double Beta = 4.0;
    public double Evaporation = 0.1;
    public double Q = 500.0;
    public double PheromoneStartValue = 100.0;
    public int maxIterationsNumber = 1000;
    public int iterationsWithoutImprovementCounter =
maxIterationsNumber;
}
```

### 8.2.4 Ant.java

```
import java.util.ArrayList;

public class Ant {
    private final ArrayList<Integer> route;
    private final boolean[] visited;

    public Ant(int numCities) {
        this.route = new ArrayList<>();
        this.visited = new boolean[numCities];
    }
    public void visitCity(int city) {
        route.add(city);
        visited[city] = true;
    }
    public ArrayList<Integer> getRoute() {
        return route;
    }
    public boolean getVisited(int city) {
        return visited[city];
    }
}
```

### 8.2.5 AntColonyOptimizationAlgorithm.java

```
import java.util.ArrayList;
import java.util.List;

public class AntColonyOptimizationAlgorithm {

    public AlgorithmParameters algorithmParameters;
    public int antsAndCitiesCount;
    public double[][] values;
    public double[][] pheromones;

    public AntColonyOptimizationAlgorithm(int numberOfAntsCities,
double[][] values, AlgorithmParameters algorithmParameters) {
        this.antsAndCitiesCount = numberOfAntsCities;
        this.values = values;
        this.algorithmParameters = algorithmParameters;

        pheromones = new
double[antsAndCitiesCount][antsAndCitiesCount];
        for (int i = 0; i < antsAndCitiesCount; i++) {
            for (int j = 0; j < antsAndCitiesCount; j++) {
                pheromones[i][j] =
algorithmParameters.PheromoneStartValue;
            }
        }
    }

    public List<Integer> calculate(int startCityIndex) {
        double bestRouteValue = Double.MAX_VALUE;
        List<Integer> bestRoute = new ArrayList<>();
        int iterationsNumberWithoutImprovement = 0;
        for (int iter = 0; iter <
algorithmParameters.maxIterationsNumber; iter++) {
            List<List<Integer>> allRoutes = new ArrayList<>();
            List<Double> allRoutesValues = new ArrayList<>();


            for (int ant = 0; ant < antsAndCitiesCount; ant++) {
                Ant a = new Ant(antsAndCitiesCount);
```

```
                List<Integer> route = createRoute(a, startCityIndex);
                double totalValue = getTotalValueForRoute(route);
                allRoutes.add(route);
                allRoutesValues.add(totalValue);

                if (totalValue < bestRouteValue) {
                    bestRouteValue = totalValue;
                    bestRoute = route;
                    iterationsNumberWithoutImprovement = 0;
                }
            }
            updatePheromones(allRoutes, allRoutesValues);

            if (iterationsNumberWithoutImprovement >=
algorithmParameters.iterationsWithoutImprovementCounter) {
                break;
            }

            iterationsNumberWithoutImprovement++;
        }
        return bestRoute;
    }
    public double getTotalValueForRoute(List<Integer> route) {
        double total = 0.0;
        for (int i = 0; i < route.size() - 1; i++) {
            total += values[route.get(i)][route.get(i + 1)];
        }
        total += values[route.get(route.size()-1)][route.get(0)];
        return total;
    }

    public List<Integer> createRoute(Ant ant, int startCityIndex) {
        ant.visitCity(startCityIndex);
        int currentCity = startCityIndex;
        while (ant.getRoute().size() < antsAndCitiesCount) {
            int nextCity = nextCity(currentCity, ant);
            if (nextCity == -1)
            {
                break;
            }
            ant.visitCity(nextCity);
            currentCity = nextCity;
        }
        return ant.getRoute();
    }

    private int nextCity(int currentCity, Ant ant) {
        double totalProbability = 0.0;
        double[] probabilities = new double[antsAndCitiesCount];

        for (int i = 0; i < antsAndCitiesCount; i++) {
            if (!ant.getVisited(i)){
                probabilities[i] =
Math.pow(pheromones[currentCity][i], algorithmParameters.Alpha) *
                        Math.pow(1.0 / values[currentCity][i],
algorithmParameters.Beta);
                totalProbability += probabilities[i];
            }
        }



        for (int i = 0; i < antsAndCitiesCount; i++) {
```

```
            probabilities[i] /= totalProbability;
        }
        double rand = Math.random();
        double currProbability = 0.0;
        int nextCityIndex = -1;
        for (int i = 0; i < antsAndCitiesCount; i++) {
            if (!ant.getVisited(i)) {
                currProbability += probabilities[i];
                if (currProbability >= rand) {
                    nextCityIndex = i;
                    break;
                }
            }
        }
        return nextCityIndex;
    }
    public void updatePheromones(List<List<Integer>> routes,
List<Double> routeValues) {
        for (int i = 0; i < antsAndCitiesCount; i++) {
            for (int j = 0; j < antsAndCitiesCount; j++) {
                pheromones[i][j] *= (1 -
algorithmParameters.Evaporation);
                if (pheromones[i][j] < 0.01) {
                    pheromones[i][j] = 0.01;
                }
            }
        }

        for (int ant = 0; ant < antsAndCitiesCount; ant++) {
            List<Integer> route = routes.get(ant);
            double routeValue = routeValues.get(ant);
            double pheromoneAdd = algorithmParameters.Q / routeValue;

            for (int i = 0; i < route.size() - 1; i++) {
                int from = route.get(i);
                int to = route.get(i + 1);
                pheromones[from][to] += pheromoneAdd;
                pheromones[to][from] += pheromoneAdd;
            }
            int from = route.get(route.size()-1);
            int to = route.get(0);
            pheromones[from][to] += pheromoneAdd;
            pheromones[to][from] += pheromoneAdd;
        }
    }

}
```

### 8.2.6 Input.java

```
import java.util.ArrayList;
import java.util.List;

public class Input {
    public double[][] distances = {
            {0, 15, 11, 25, 32, 38, 19, 9, 19, 15, 14, 21, 22},
            {15, 0, 30, 31, 24, 33, 28, 11, 30, 26, 9, 21, 33},
            {11, 30, 0, 32, 39, 39, 27, 19, 26, 22, 23, 28, 29},
            {25, 31, 32, 0, 13, 9, 10, 30, 32, 11, 37, 44, 24},
            {32, 24, 39, 13, 0, 7, 12, 34, 30, 23, 37, 51, 28},
            {38, 33, 39, 9, 7, 0, 16, 40, 35, 18, 48, 57, 31},
            {19, 28, 27, 10, 12, 16, 0, 25, 18, 8, 30, 38, 24},
            {9, 11, 19, 30, 34, 40, 25, 0, 24, 20, 11, 22, 27},
            {19, 30, 26, 32, 30, 35, 18, 24, 0, 13, 28, 35, 9},
            {15, 26, 22, 11, 23, 18, 8, 20, 13, 0, 26, 34, 11},
            {14, 9, 23, 37, 37, 48, 30, 11, 28, 26, 0, 12, 32},
            {21, 21, 28, 44, 51, 57, 38, 22, 35, 34, 11, 0, 38},
            {22, 33, 29, 24, 28, 31, 24, 27, 9, 11, 32, 38, 0}
    };

    public double[][] fuel_costs = {
            {1.599, 1.599, 1.599, 1.599, 1.599, 1.599, 1.599, 1.599,
1.599, 1.599, 1.599, 1.599, 1.599},
            {1.609, 1.609, 1.609, 1.609, 1.609, 1.609, 1.609, 1.609,
1.609, 1.609, 1.609, 1.609, 1.609},
            {1.629, 1.629, 1.629, 1.629, 1.629, 1.629, 1.629, 1.629,
1.629, 1.629, 1.629, 1.629, 1.629},
            {1.579, 1.579, 1.579, 1.579, 1.579, 1.579, 1.579, 1.579,
1.579, 1.579, 1.579, 1.579, 1.579},
            {1.579, 1.579, 1.579, 1.579, 1.579, 1.579, 1.579, 1.579,
1.579, 1.579, 1.579, 1.579, 1.579},
            {1.619, 1.619, 1.619, 1.619, 1.619, 1.619, 1.619, 1.619,
1.619, 1.619, 1.619, 1.619, 1.619},
            {1.589, 1.589, 1.589, 1.589, 1.589, 1.589, 1.589, 1.589,
1.589, 1.589, 1.589, 1.589, 1.589},
            {1.619, 1.619, 1.619, 1.619, 1.619, 1.619, 1.619, 1.619,
1.619, 1.619, 1.619, 1.619, 1.619},
            {1.639, 1.639, 1.639, 1.639, 1.639, 1.639, 1.639, 1.639,
1.639, 1.639, 1.639, 1.639, 1.639},
            {1.579, 1.579, 1.579, 1.579, 1.579, 1.579, 1.579, 1.579,
1.579, 1.579, 1.579, 1.579, 1.579},
            {2.199, 2.199, 2.199, 2.199, 2.199, 2.199, 2.199, 2.199,
2.199, 2.199, 2.199, 2.199, 2.199},
            {1.639, 1.639, 1.639, 1.639, 1.639, 1.639, 1.639, 1.639,
1.639, 1.639, 1.639, 1.639, 1.639},
            {1.649, 1.649, 1.649, 1.649, 1.649, 1.649, 1.649, 1.649,
1.649, 1.649, 1.649, 1.649, 1.649}
    };

    public double fuel_consumption = 17.0;
    List<String> cities = new ArrayList<>(List.of("Wesel",
"Dinslaken", "Hamminkeln", "Kamp-Lintfort","Moers",
            "Neukirchen-Vluyn", "Rheinberg", "Voerde", "Xanten",
"Alpen", "Hünxe", "Schermbeck", "Sonsbeck"));
}
```

### 8.2.7 ElitistAntSystem.java

```
import java.util.ArrayList;
import java.util.List;
public class ElitistAntSystem extends AntColonyOptimizationAlgorithm {
    private final int eliteAntsCount;
    public ElitistAntSystem(int numberOfAntsCities, double[][] values,
AlgorithmParameters algorithmParameters, int eliteAntsCount) {
        super(numberOfAntsCities, values, algorithmParameters);
        this.eliteAntsCount = eliteAntsCount;
    }
    public void updatePheromones(List<List<Integer>> routes,
List<Double> routeValues, List<Integer> bestRoute, double
bestRouteValue) {
        super.updatePheromones(routes, routeValues);

        // Additional pheromone deposit by elite ants on the best
route
        double pheromoneAdd = algorithmParameters.Q / bestRouteValue;
        for (int i = 0; i < bestRoute.size() - 1; i++) {
            int from = bestRoute.get(i);
            int to = bestRoute.get(i + 1);
            pheromones[from][to] += eliteAntsCount * pheromoneAdd;
            pheromones[to][from] += eliteAntsCount * pheromoneAdd;
        }
        int from = bestRoute.get(bestRoute.size() - 1);
        int to = bestRoute.get(0);
        pheromones[from][to] += eliteAntsCount * pheromoneAdd;
        pheromones[to][from] += eliteAntsCount * pheromoneAdd;
    }
    @Override
    public List<Integer> calculate(int startCityIndex) {
        double bestRouteValue = Double.MAX_VALUE;
        List<Integer> bestRoute = new ArrayList<>();
        int iterationsNumberWithoutImprovement = 0;
        for (int iter = 0; iter <
algorithmParameters.maxIterationsNumber; iter++) {
            List<List<Integer>> allRoutes = new ArrayList<>();
            List<Double> allRoutesValues = new ArrayList<>();
            for (int ant = 0; ant < antsAndCitiesCount; ant++) {
                Ant a = new Ant(antsAndCitiesCount);
                List<Integer> route = createRoute(a, startCityIndex);
                double totalValue = getTotalValueForRoute(route);
                allRoutes.add(route);
                allRoutesValues.add(totalValue);
                if (totalValue < bestRouteValue) {
                    bestRouteValue = totalValue;
                    bestRoute = route;
                    iterationsNumberWithoutImprovement = 0;
                }
            }
            updatePheromones(allRoutes, allRoutesValues, bestRoute,
bestRouteValue);

            if (iterationsNumberWithoutImprovement >=
algorithmParameters.iterationsWithoutImprovementCounter) {
                break;
            }
            iterationsNumberWithoutImprovement++;
        }
        return bestRoute;
    }
```

## 9. Declaration of Authenticity

I, Anna Pogrebnyak, hereby declare that the work presented herein is my own work completed without the use of any aids other than those listed. Any material from other sources or works done by others has been given due acknowledgement and listed in the reference section. Sentences or parts of sentences quoted literally are marked as quotations; identification of other references with regard to the statement and scope of the work is quoted. The work presented herein has not been published or submitted elsewhere for assessment in the same or a similar form. I will retain a copy of this assignment until after the Board of Examiners has published the results, which I will make available on request.