Delft University of Technology

FACULTY MECHANICAL, MARITIME AND MATERIALS ENGINEERING
Department Marine and Transport Technology

Mekelweg 2
2628 CD Delft
the Netherlands
Phone +31 (0)15-2782889
Fax +31 (0)15-2781397
www.mtt.tudelft.nl

Specialization:	Transport Engineering and Logistics
Report number:	2015.TEL.7916
Title:	**Ant Colony Optimization for the Travelling Salesman Problem**
Author:	M.C.M. van Tol, BSc

Title (in Dutch)	Mieren Kolonie Optimalisatie voor het Handelsreiziger Probleem

Assignment:	Research
Confidential:	No
Supervisor:	ir. M.B. Duinkerken
Date:	February 26, 2015

Delft University of Technology

FACULTY OF MECHANICAL, MARITIME AND MATERIALS ENGINEERING
Department of Marine and Transport Technology

Mekelweg 2
2628 CD Delft
the Netherlands
Phone +31 (0)15-2782889
Fax +31 (0)15-2781397
www.mtt.tudelft.nl

Student:	M.C.M. van Tol, BSc	Assignment type:	computer assignment
Mentor:	M.B. Duinkerken	Report number:	2015.TEL.7916
Specialization:	TEL	Confidential:	No
Creditpoints (EC):	15		

Subject: **Ant Colony Optimization for the Travelling Salesman Problem**

The Travelling Salesman Problem (TSP) is a well known problem in the field of transport engineering and logistics. It plays an important role in the planning for transport and other logistic systems. A promising approach to solve TSPs is Ant Colony Optimization (ACO).

Ant Colony Optimization is a 'nature inspired' heuristic approach that can be used for combinatorial optimization. One of the earliest problems that was solved with ACO is the well known Travelling Salesman Problem. In the ACO heuristic intelligent agents (so-called 'ants') find their way in a network leaving a trail of 'pheromones'. If a certain path in the network brings the ant closer to its goal, a stronger trail is left behind. This trail will attract other ants. While in the first iterations all routes in the network are equally attractive to the ants, after a number of iterations the stronger pheromone-trails are the routes that leads to the goal. Eventually, only one trail remains which represents the optimal solution.

Your assignment is to implement an ACO-algorithm for the Travelling Salesman Problem. The algorithm must be able to solve existing TSP-instances. Part of the project is to study the usability of OpenStreetMap data to represent the TSP-instance (see http://www.openstreetmap.org).

Studying relevant literature, developing and implementing a model, verification and validation of the model, experimenting with different scenario's, presenting solid conclusions and recommendations and reporting the research work are all part of this assignment.

The report should comply with the guidelines of the section. Details can be found on the website.

The supervisor,

M.B. Duinkerken

NOMENCLATURE

Symbol	
a	Degree of non-linearity of edge selection
b	Degree of attraction to edge without pheromone trail
d	Distance (cost)
E	Set of edges
f	Frequency
G	Graph
h	Hour
k	Number of nearest neighbours
L	Length (cost)
m	Number of ants
m	Minute
n	Number of nodes
p	Probability
Q	Set of nodes
s	Second
t	Time
T	Tour (route)
V	Set of vertices
w	Search width
α	Influence of pheromone trail intensity
β	Influence of heuristic information
δ	Difference in pheromone trial intensity
η	Reciprocal of d
λ	Influence of difference in pheromone trial intensity
λ_i	λ-branching factor of node i
ρ	Persistence of pheromone trail
σ	Standard deviation
τ	Pheromone trail intensity

Sub- or superscript	
avg	Average
b	Best
ij	Node i to node j
max	Maximum
min	Minimum
opt	Optimal
$theo$	Theoretical

Acronym	
ACO	Ant Colony Optimization
ACS	Ant Colony System
ANTS	Approximated Non-deterministic Tree Search
AS	Ant System
ATSP	Asymmetric Travelling Salesman Problem
BA	Bat Algorithm
B&B	Branch and Bound
CPU	Central Processing Unit
CVRP	Capacitated Vehicle Routing Problem
GB	Global Best
HCP	Hamiltonian Cycle Problem
IB	Iteration Best
MDS	Multidimensional Scaling
MMAS	Max-Min Ant System
OSM	OpenStreetMap
PSO	Particle Swarm Optimization
PTS	Pheromone Trail Smoothing
PTU	Pheromone Trail Update
SOP	Sequential Ordering Problem
TSP	Travelling Salesman Problem
TSP LIB	TSP Library
URL	Uniform Resource Locator

CONTENTS

ABSTRACT

Nature is a source of inspiration for humans in many ways. An interesting source of inspiration is natural swarm intelligence. Natural swarm intelligence entails the intelligent behaviour of a group of individuals as a whole, with individuals having only a limited intelligence. The intelligence of the individuals is judged limited because they follow very simple rules and there is no centralized control structure to organize the behaviour of the individuals. Natural swarm intelligence inspires many artificial (exhibited by machines or software) swarm intelligence applications. One of these applications is based on the colony behaviour of ants.

Ants are almost blind animals, but they are able to find the shortest path between their nest and a food source. During foraging, the ants communicate to each other via depositing pheromone. When an ant locates a food source, it carries the food to the nest and deposits a pheromone trail. The path of the other foraging ants is influenced by this pheromone trail. Stronger pheromone trails attract more ants and shorter paths accumulate faster in pheromone intensity. This positive feedback mechanism biases the ants to shorter paths. Eventually, all the ants follow the shortest path to the food source. This simple mechanism of communicating via pheromones forms the basis of Ant Colony Optimization (ACO). ACO has many applications like solving scheduling, vehicle routing and Travelling Salesman Problems (TSPs).

The main objective of this research is obtaining knowledge and experience in the field of Ant Colony Optimization. The emphasis thereby is on the performance in solving Travelling Salesman Problem (TSP) instances. The TSP is a classical problem in combinatorial optimization problems and has many applications within logistic systems. The simplicity of formulation is deceitful. Globally, algorithms to solve the TSP can be divided in exact algorithms and approximate algorithms. Exact algorithms can only be used for instances of small size. In many real-world problems, the number of nodes is too large for an exact solution within a reasonable time. Approximation (heuristic) algorithms provide a balance between a near-optimal solution and a reasonable computation time. ACO is an example of such a heuristic algorithm.

Many different types of ACO algorithms exist. One type is selected: Max-Min Ant System (MMAS). But even within one type of ACO algorithm one could think of various variants for implementation. How and why different elements of the algorithm are implemented is discussed in detail. MMAS algorithm is equipped with conventional heuristic algorithms to improve performance, like nearest neighbour and local search. The MMAS algorithm is implemented in Delphi 2010 in combination with the discrete simulation package TOMAS. However, the runs performed with the implemented MMAS algorithm are not discrete event simulations. The reason why TOMAS still is used is that it provides many useful tools to process elements like queues. In the computer program it is possible to turn on and off various elements and adjust the parameters of the MMAS algorithm. To be able to compare the performance of the MMAS algorithm, a second algorithm is required. Therefore, the exact algorithm Branch and Bound (B&B) is implemented in Delphi 2010.

The source of TSP instances used for experiments is TSP Library (often denoted as TSPLIB). TSP LIB is a library of sample instances for the TSP. The advantage of TSP LIB is that for most of the instances the optimal solution in terms of cost and route is known. Above that, these instances are used throughout literature. So it provides a manner for comparison in performance between different methods and thus can be considered as benchmark instances. As smaller TSP instances provided by TSP LIB were not available in the desired format, it was chosen to convert these instances. This conversion is done by means of an approximation algorithm called multidimensional scaling (MDS). For completeness, an introduction is provided in how OpenStreetMap (OSM) could be used to represent TSP instances.

Various experiments will be performed with the implemented MMAS and B&B algorithms. Both models are verified and validated. In the available literature concerning MMAS, extensive experiments are performed on parameter settings, model setups and settings. However, the emphasis of this research is on the performance in solving TSP instances with respect to time. It is concluded that the advantage of the heuristic MMAS approach with respect to the exact solver B&B starts to pay off for TSP instances of around 40 nodes and larger. A good solution (within 1% of the optimal solution) can be found with MMAS within a small amount of time compared to the run time required by B&B.

1

INTRODUCTION

1.1. SWARM INTELLIGENCE

Nature is a source of inspiration for humans in various ways. A interesting source of inspiration is natural swarm intelligence. Natural swarm intelligence entails the intelligent behaviour of a group of individuals as a whole, with individuals having only a limited intelligence. The intelligence of the individuals is judged limited because they follow very simple rules and there is no centralized control structure to organize the behaviour of the individuals. Natural swarm intelligence forms the inspiration for many artificial (exhibited by machines or software) swarm intelligence applications. A few examples of swarm intelligence will be introduced.

Flocking and schooling of birds and fish formed the idea behind Particle Swarm Optimization (PSO). PSO is an algorithm to determine the best point or surface in an n-dimensional space. The algorithms works with a population (swarm) of candidate solutions (particles). The particles move around in the search space by obeying simple rules. The movements of particles are influenced by the best known position in the search space of the particle as the best known position in the search space of the swarm. If better positions are discovered, these will influence the movements of the swarm. This process is repeated. PSO is a heuristic, so it is not guaranteed that an optimal solution is ever found. An example of the application of PSO is antenna radiation field design.

Bat algorithm (BA) is a heuristic algorithm based on the echo locating behaviour of bats. An animal emits a sound to the environment. The sound waves return from objects. The animal listens to the echoes and this enables the animal to locate and identify the objects. Echolocation behaviour contains varying sound properties in terms of frequency, pulse rates of emission and loudness. The algorithm can be summarized as follows. There is a swarm of bats and each bat flies around randomly with a velocity at a certain position with varying sound properties. The bat searches for a prey. If the bad finds a prey, it changes sound properties. The echolocation behaviour of bats is formulated in such a way that the best performing bats are selected and that it can be used to optimize an (multiple) objective function. An example of the application of BA is the ergonomic screening of office workplaces.

The last example considers the colony behaviour of ants. Ants are almost blind animals, but they are able to find the shortest path between their nest and a food source. During foraging, the ants communicate to each other via depositing pheromone. When an ant locates a food source, it carries the food to the nest and deposits a pheromone trail. The path of the other foraging ants is influenced by this pheromone trail. Stronger pheromone trails attract more ants and shorter paths accumulate faster in pheromone intensity. This positive feedback mechanism biases the ants to shorter paths. Eventually, all the ants follow the shortest path to the food source. This simple mechanism of communicating via pheromones forms the basis of Ant Colony Optimization (ACO). ACO has many applications like solving scheduling, vehicle routing and Travelling Salesman Problems (TSPs).

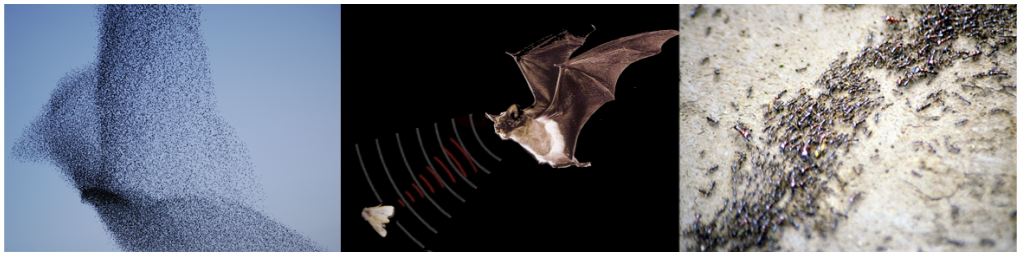

Figure 1.1: Example of artificial intelligence (a) Bird flock (b) Bat hunting (c) Ant colony

1.2. PROBLEM STATEMENT

The main objective of this research is obtaining knowledge and experience in the field of Ant Colony Optimization for the section Transportation Engineering and Logistics. The section deals with scheduling and routing problems, including the Travelling Salesman Problem. The Travelling Salesman Problem is a classical problem in combinatorial optimization problems and has many applications within logistic systems. The simplicity of its formulation is deceitful. Ant Colony Optimization seems a promising approach to solve Travelling Salesman Problem instances. This research is a first step in exploring the potential of Ant Colony Optimization in solving these complex problems. In future developments Ant Colony Optimization may form a shackle in a large simulation environment or real time planning procedure for example. The main research question is formulated as follows:

> *How does an Ant Colony Optimization algorithm perform compared to an exact algorithm in solving Travelling Salesman Problem instances?*

In order to answer this main research question, the following sub questions are proposed:

- What is the Travelling Salesman Problem?
- How does an exact algorithm to solve the Travelling Salesman Problem work?
- How does Ant Colony Optimization work?
- How can an Ant Colony Optimization algorithm to solve the Travelling Salesman Problem be implemented?
- How can the performance of an Ant Colony Optimization algorithm and an exact algorithm in solving Travelling Salesman Problem instances be compared?

1.3. APPROACH

The research is started with a literature survey. First the Travelling Salesman Problem and an exact algorithm will be considered. Subsequently, an overview of Ant Colony Optimization will be given. Herein different algorithms will be introduced. One Ant Colony Optimization algorithm from literature will be selected. The literature concerning this algorithm will be reviewed in detail. The next part focuses on how the information from literature can be translated to implementation. For implementation the software package Delphi 2010 in combination with the discrete simulation package TOMAS [Veeke and Ottjes, 2010] will be used. Finally, the Ant Colony Optimization algorithm is verified and validated. Several experiments are performed to benchmark its performance. This report will follow the same outline as described above.

The Ant Colony Optimization algorithm is build from scratch. In building this model, (mathematical) concepts from literature are used. The algorithm must be able to solve existing TSP instances from literature. For this research two algorithms are required: a heuristic and an exact algorithm. For the exact algorithm, use will be made of public resources. To eliminate a difference in performance due to programming language, both algorithms are implemented in Delphi 2010.

2

TRAVELLING SALESMAN PROBLEM

In this chapter the Travelling Salesman Problem will be discussed in detail. First a mathematical formulation of the problem is given. Subsequently, the origin of its complexity is explained. The Travelling Salesman Problem has various real-world applications. Some examples will be given. During research an open source library of Travelling Salesman Problem instances is used. This library is introduced. Globally, algorithms to solve the Travelling Salesman Problem can be divided in exact algorithms and approximate algorithms. One exact algorithm is discussed in detail.

2.1. PROBLEM DESCRIPTION

A travelling salesman must visit each city in a certain area exactly once. The journey starts and ends at his home town. Given the cost between each pair of cities, find the route of the journey with minimal cost. This is called the travelling salesman problem (TSP). Wei and Yuren [2010] formulates the TSP mathematically as follows: let $G = (V, E)$ be a graph that consists of the set of vertices (also called nodes, e.g. cities) V and the set of edges (e.g. roads) E. Find a shortest Hamiltonian cycle of G. A Hamiltonian cycle is a closed path whereby each vertex in a graph is visited exactly once (named after the Irish mathematician Sir Hamilton). The cost between each pair of nodes is given in a cost matrix D with elements d_{ij}, which represent the cost between node i and node j. If $d_{ij} = d_{ji}$ for all i, j, then the cost matrix is symmetric. Otherwise, the cost matrix is asymmetric. A TSP instance does not necessarily have only one unique optimal solution. An optimal solution of an example TSP instance is shown in Figure 2.1.

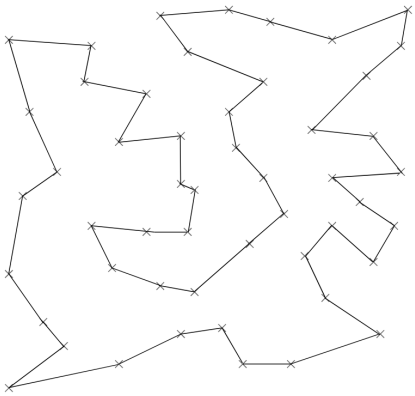

Figure 2.1: Example optimal solution TSP [Zhou et al., 2012]

The TSP is a classical problem in combinatorial optimization problems. The TSP has been studied extensively in academic circles and still attracts many scholars [Wei and Yuren, 2010]. Laporte [1992] states that hundreds of articles have been written on the TSP. The simplicity of the formulation of the problem is deceitful. For the first node, there are n possibilities, for the second node $n-1$ possibilities, for the third node $n-2$ possibilities etc. The route start and ends at the same node and thus the route is closed. Therefore, it does not matter from which node the journey starts. The number of possible routes for n nodes can be expressed as follows:

$$\frac{n(n-1)(n-2)...}{n} = (n-1)! \tag{2.1}$$

If the cost matrix is symmetrical, the number of possible routes is counted double in Equation 2.1. The number of possible routes for n nodes in this case is:

$$\frac{n(n-1)(n-2)...}{2n} = (1/2)(n-1)! \tag{2.2}$$

Globally, algorithms to solve the TSP can be divided in exact algorithms and approximate algorithms. Exact algorithms, such as Branch & Bound (see section 2.3), Branch & Cut and Dynamic Programming can only be used for instances of small size since the time to find a solution increases exponentially with the number of nodes [Shokouhifar and Sabet, 2012]. For example, $n = 25$ gives $(15-1)!$ possibilities, that is over 87 billion possibilities. The TSP has become a benchmark for many (newly) developed algorithms [Shi et al., 2008]. In many real-world problems, the number of nodes is too large for an exact solution within a reasonable time. Approximation (heuristic) algorithms provide a balance between a near-optimal solution and a reasonable computation time.

The TSP has many applications within logistic systems. Some applications can be converted to the TSP, for example by introducing a dummy node. Laporte [1992] proposes a few applications of the TSP:

- Computer wiring: The pins in a computer need to be linked by means of wires. The objective is to minimize the total required wire length.
- Hole drilling: In various manufacturing industries holes are drilled in for example boards or metal sheets. The objective is to minimize the distance the drill has to move over the sheet.
- Job sequencing: Various jobs must be performed on a machine. Different tools can be used on the machine. To change tools, different change-over times are required. The objective is to minimize the time required to change tools.

2.2. TSP LIBRARY

TSP library (often denoted as TSPLIB) is a library of sample instances for the TSP (and related problems) from various sources and of various types [Reinelt, 1995]. For most of the instances the optimal solution in terms of cost and route is known. The instances and optimal solutions can be downloaded from the website (see bibliography for URL). This is very useful in validating your developed method to solve these instances. Above that, these instances are used throughout literature. So it provides a manner for comparison in performance between different methods and thus can be considered as benchmark instances. Also in this research the TSP LIB is used. TSPLIB provides instances for the following classes of problems:

- Symmetric travelling salesman problem (TSP)
- Asymmetric travelling salesman problem (ATSP)
- Hamiltonian cycle problem (HCP)
- Sequential ordering problem (SOP)
- Capacitated vehicle routing problem (CVRP)

An overview of the symmetric travelling salesman problems provided by TSP LIB with their solution bounds (a single number indicates that the optimal solution is known) is given in Appendix A. The number in the file name indicates the number of nodes of the instance. Instance size varies from 17 to almost 86000 nodes. Every TSP file has the same setup as will be described below. Each file consists of two parts. In the first part the instance is specified with name, type, dimension, edge weight type and edge weight format (all strings). The edge weight type specifies how the edge weights (or distances) are given: either implicitly (e.g. Euclidean distance, Manhattan distance, geographical distance, 2D, 3D) in the form of a node list or explicitly in the form of a cost matrix. Appendix B provides both an explicit as an implicit example. If the cost matrix is given explicitly, an edge weight format is given. This describes the format of the edge weights, e.g. a full matrix, a lower triangular matrix or a function. Please note that the edge weight type is essential in case the cost matrix is given implicitly. If the user does not stick to this specification, the found optimal solution will not correspond to the optimal solution given by TSP LIB. The second part of the file prescribes the data of the instance (integers or reals). A few TSP files contain both a list of node coordinates as a cost matrix. One could also think of other open sources for TSP instances. section 2.4 provides an introduction to how OpenStreetMap could be used to represent these instances.

2.3. EXACT ALGORITHM

To save time, use is made of public resources to implement an exact algorithm for solving TSP instances. It should be noted that the availability of such exact algorithms for Delphi is very limited. Above that, not all available algorithms are able to read TSP instances from the TSP LIB. Thus in most cases the source code must be available. A Branch and Bound (B&B) algorithm is selected. B&B was first published by Little et al. [1963]. The following algorithm is used: Stony-Brook-University [] (see bibliography for URL). This university collects algorithms for various combinatorial optimization problems. The algorithms are made available for research or educational use. The B&B source code is edited and made suitable to read instances from TSP LIB. The execution time of this branch and bound grows with the size of the network. In the description of the algorithm it is stated that the worst-case time complexity of this algorithm could be as bad as $\mathcal{O}(n!)$. Despite Little's B&B is able to handle asymmetric TSP instances, this algorithm is only able to handle symmetric TSP instances. In the remainder of this section more information on Little's B&B algorithm will be given.

As stated in section 2.1, Branch & Bound (B&B) is one of the exact algorithms to solve TSP instances. Little et al. [1963] describes B&B as follows. The idea is to break up all feasible routes into increasing smaller subsets by a procedure called branching. Figure 2.2 provides an example of a branch and bound tree. For each subset, a lower bound on the cost (length) of the included routes is calculated. These bounds guide the branching of the subsets. Eventually, a subset is found that contains a single route whose length is less than or equal to the lower bound of all other subsets. That route is optimal. The subsets of routes are conveniently represented as the nodes of a tree and the process of partitioning as a branching of the tree. Hence the name of the method: branch and bound.

For this research Little's B&B algorithm is implemented in Delphi. The B&B algorithm can be described more pragmatic as follows. The algorithms starts with an nxn weight matrix for a certain graph. The first step is reduction: in every row and column of the matrix a zero has to be created by subtracting constants. The sum of these subtracted constants is a lower bound (indicated in Figure 2.2 top left of the circles) for the length of all TSP routes from the matrix. Subsequently, the branch and bound process is started. Branching takes place at a certain edge (indicated as a pair of nodes in Figure 2.2 in the circles). One branch contains all routes with a certain pair of nodes, the other branch contains all routes without a certain pair of nodes. For both branches a new matrix has to be formed, for one branch the matrix is reduced in size and for the other branch the matrix weights are modified. Subsequently, reduction is applied again on these new matrices. This results in a (new) lower bound value per matrix. Based on these lower bound values, the search in the branch and bound tree for the optimal solution is guided. For further clarification on the algorithm, an example provided in the paper by Little et al. [1963] should be considered. Please note that, in a worse-case scenario, the algorithm may end up examining all possible solutions.

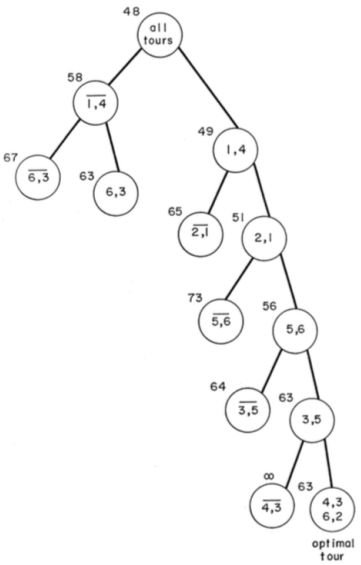

Figure 2.2: Example of branch and bound tree [Little et al., 1963]

2.4. OPENSTREETMAP

As indicated in section 1.2 the Travelling Salesman Problem has many applications within logistic systems. At the end of section 2.1 a few applications of the TSP are proposed. However, many other applications of the TSP are physical transport problems with cities and roads involved. These physical transport problems can be represented on a street map. OpenStreetMap (OSM) could be used for this representation. It provides nodes and edges to digitally represent cities and roads. In this section an introduction to the usability of OSM is given. OpenStreetMap is an initiative to create and provide free geographic data to anyone [OpenStreetMap, 2014a]. The OpenStreetMap foundation was founded in 2004 and is a non-profit organization. The updating of the maps is completely performed by volunteers and thus is very comparable to Wikipedia.

Elements (also known as data primitives) are the basic components of OpenStreetMap's conceptual data model of the physical world and consist of [OpenStreetMap, 2014b]:

- Nodes
- Ways
- Relations

A node represents a specific point on the surface of the earth. To each node are at least an id number and a pair of coordinates (latitude and longitude) attached. Nodes can be used to define standalone point features, like a park bench or a water well. A way is an ordered list of nodes (2 – 2 000 nodes) that defines a poly-line (a connected series of line segments). Ways are used to represent linear features such as rivers and roads, but can also represent the boundaries of areas such as buildings or forests. Open and closed ways can be distinguished. An open way is a way which does not share a first and last node. A closed way is a way for which the last node of the way corresponds with the first node. A relation is a multi-purpose data structure that documents a relationship between two or more data primitives (nodes, ways, and/or other relations). For example, a way of more than 2 000 nodes, cannot be represented by a single way. Instead, the feature will require a relation data structure.

The data from OpenStreetMap can be downloaded in a number of ways [OpenStreetMap, 2014c]. The entire planet (40 GB compressed) or smaller areas can be selected. Data is available in different forms, one of them is in the form of XML formatted .osm files. Different tools (like Osmosis, osmconvert, and osmfilter) are available to extract specific data. OpenStreetMap has the ability to be used as a source to represent TSP instances. However, a conversion of the XML files is required. An interesting open source program is OSM2PO. It is both a converter and a routing engine. The program converts OpenStreetMap's XML-data and makes it routable. A widely used program to view geographic data is the open source program QGIS. There are also numerous plug-ins available to extend the possibilities of QGIS in the field of routing. Figure 2.3 shows an example of the usage of the pgRouting plug-in for QGIS to determine shortest path between a pair of nodes (based on Dijkstra's algorithm).

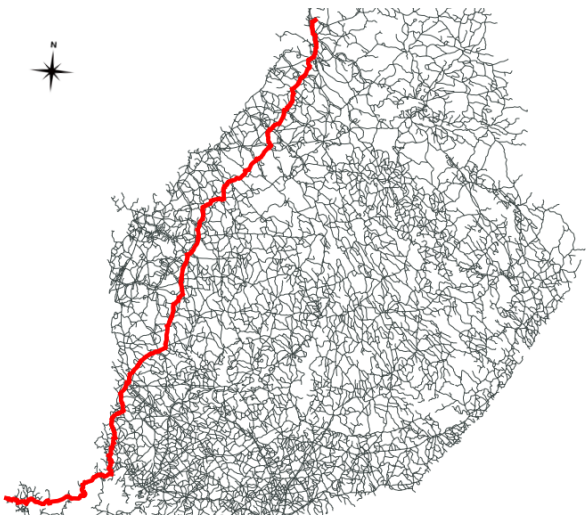

Figure 2.3: Example of shortest path between two nodes using pgRouting plugin [QGIS, 2014]

3

PRINCIPLE OF ANT COLONY OPTIMIZATION

As indicated in section 2.1, in many real-world problems the number of nodes of a TSP instance is too large for an exact solution within a reasonable time. In that case heuristic algorithms may be used. These approximation algorithms provide a balance between a near-optimal solution and a reasonable computation time. Ant Colony Optimization is such a heuristic algorithm. In this chapter the principle of Ant Colony Optimization is discussed. Various types of Ant Colony Optimization exist. A brief overview of these algorithms is given. Subsequently, one algorithm is selected for implementation.

3.1. ANT COLONY

An ant colony is a community of ants living close to together. Some ant species are able to collectively find the shortest path between two points [Solnon, 2010] (page 108), often the nest of the ant colony and a food source. When the path is obstructed or destroyed, a new alternative paths are found. These abilities are remarkable, since ants are almost blind animals [Dorigo et al., 1996]. For example Deneubourg et al. [1990] studied the foraging behaviour of ant species in order to develop a model to describe its behaviour. One of the experiments will be discussed in section 3.2. While an ant moves around, it deposits a pheromone trail on the ground. The pheromone is a chemical substance. An ant is able to detect the intensity of a pheromone trail. Initially ants randomly choose their path, but the probability of choosing a direction depends on the intensity of pheromone trails on the ground [Solnon, 2010] (page 108). So if an ant encounters a pheromone trail, the higher the probability the ant follows that direction. During following the ant reinforces the pheromone trail with its own pheromone. So a form of positive feedback takes place; the denser the pheromone trail, the larger the probability that an ant will follow the trail and will make it denser with its own pheromone. This principle is illustrated with an example.

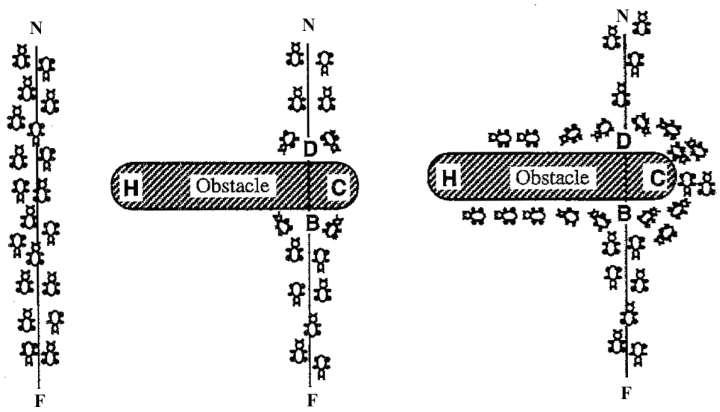

Figure 3.1: Example of ants [Dorigo et al., 1996]

This example proposed by Dorigo et al. [1996]. Assume that there is a path along which ants are moving from nest N to food source F and back, see left of Figure 3.1. Suddenly an obstacle is placed on the path, see middle of Figure 3.1. The ant that is going from N to F and arrives at D, has to decide whether to turn left or right. The same holds for the ant that is going from F to N and arrives at B. This choice is influenced by the intensity of the pheromone trail by preceding ants. For example, if the intensity of the left path is higher, the higher the probability the ant will turn left. The first ant that arrives at D (or B) has no predecessors and thus

the probability of turning left or right is equal. Assume that two ants leave at exact the same time from D and each choose a different path. The ant following path $D-C-B$ will arrive earlier at B than the ant following path $D-H-B$. The implication is that after this arrival, the first ant that arrives at B and is going from F to N will detect a pheromone trail and have a higher probability of taking path $B-C-D$ than path $B-H-D$. The result is that the number of ants following the shorter path is higher and also the intensity of the pheromone trail on the shorter path is higher. The final result is that almost all ants choose the shorter path. Some ants may not follow the highest pheromone trail and thus exhibit an exploratory behaviour [Solnon, 2010] (page 110).

3.2. DOUBLE BRIDGE EXPERIMENT

Deneubourg et al. [1990] studied the foraging behaviour of ant species. One of the experiments performed was the double bridge experiment with Argentine ants, see left of Figure 3.2. The nest is separated from the food by two bridges of equal length and initially free of pheromone. The ants explore the environment and eventually reach the food. Initially, the ants select one of the bridges randomly. However, after some time most of the ants follow one of the bridges. This is due to random fluctuations in path selection, which results in higher concentrations on one of the bridges and thus attracts more ants.

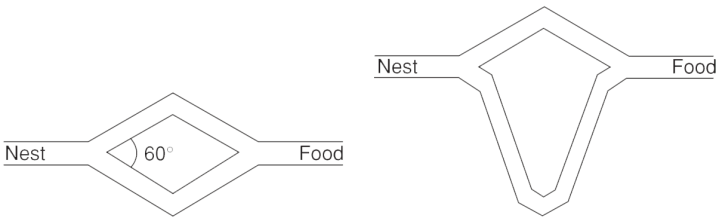

Figure 3.2: Double bridge experiment [Dorigo, 2006]

Goss et al. [1989] extended the double bridge experiment, see right of Figure 3.2. Now one bridge is longer than the other. Again, initially the ants select one of the bridges randomly. After some time most of the ants follow the shorter bridge. Random fluctuations still occur, however the ants following the shorter path arrive earlier at the food. Therefore the shorter bridge has a higher pheromone trail intensity from the moment the first ant arrives at the food and this trail will increasingly be reinforced, like described at the end of section 3.1. Goss et al. [1989] developed an empirical model to describe the observed behaviour. The probability for an ant to select bridge one and bridge two is given by:

$$p_1 = \frac{(m_1 + b)^a}{(m_1 + b)^a + (m_2 + b)^a} \tag{3.1}$$

$$p_2 = 1 - p_1 \tag{3.2}$$

where m_1 is the number of ants that have passed bridge one, m_2 is the number of ants that have passed bridge two. Equation 3.1 is a choice function. The equation quantifies in which way a higher pheromone trail intensity on bridge one gives a higher probability of choosing bridge one. Parameter a determines the degree of non-linearity of the choice. For example, a high value of a indicates that if one bridge has a slightly higher pheromone trail intensity than the other, the next ant will have a very high probability of choosing that branch. Parameter b indicates the degree of attraction of a bridge without a pheromone trail. Remember that ants initially wander randomly. The greater b, the greater the pheromone trail intensity must be to let choices become non-random. a and b are parameters that are fitted to experimental data. It was found that $a \approx 2$ and $b \approx 20$ provide a very good fit. The article presenting this equation does not provide restrictions on parameters a and b. However, in view of the meaning of these parameters the following restrictions seem plausible: $a \geq 1$ and $b \geq 0$.

3.3. ANT COLONY OPTIMIZATION

Ant Colony Optimization (ACO) is a meta-heuristic. A meta-heuristic is a general-purpose algorithmic framework that can be applied to different optimization problems with relatively few modifications [Dorigo, 2006]. Besides routing problems (like the TSP), ACO has many other applications. Like assignment problems (e.g. sequential ordering [Gambardella and Dorigo, 2000]), scheduling problems (e.g. course timetabling [Socha, Sampels, and Manfrin, Socha et al.]). The foraging behaviour of ants described in section 3.1 is a main source of inspiration for ACO algorithms. The ACO algorithm is a multi-agent system in which the agents are artificial ants [Zhao et al., 2008]. These artificial ants build solutions to a considered optimization problem [Dorigo, 2006]. These ants communicate in a way comparable to real ants. Different ACO algorithms exist. The first ACO algorithm was introduced by Dorigo et al. [1991], called Ant System (AS). Since then different other ACO algorithms are proposed. A list of early ACO algorithms is shown in Table 3.1. Solnon [2010] (page 163) states that ACO has been applied to a large number of combinatorial optimization problems and has shown to be very competitive in comparison with convenient approaches for many challenging problems.

Dorigo [2006] states that all ACO algorithms share the same idea and illustrates this idea by introducing an application to the travelling salesman problem. In ACO a number of artificial ants are simulated that move over edges. These edges connect nodes. To each edge is a pheromone variable attached, which indicates the intensity of the pheromone trail on that edge. This variable can be influenced by ants depositing pheromone and evaporation of the pheromone. Pheromone evaporation is a process whereby the pheromone trail intensity of the edges decreases over time. Constraints are applied such that the ants visit each node exactly once and return to the starting node. ACO is an iterative algorithm. Each ant builds each iteration a solution in constructive steps. So each iterations consists of multiple constructive steps. Each constructive step an ant chooses an edge and thus a node, see Figure 3.3. This choice is stochastically determined and is influenced by the intensity of the pheromone trails. At the end of an iteration, various solutions are build. Based on the quality of the solutions (in terms of cost), the pheromone variables are updated. Subsequently, a new iteration takes place with the new pheromone values. New solutions will be build that are similar to the best solutions from the previous iteration. The long term effect of the pheromone trails is to successively reduce the size of the search space by concentrating the search on a relative small number of edges [Stützle and Hoos, 1996].

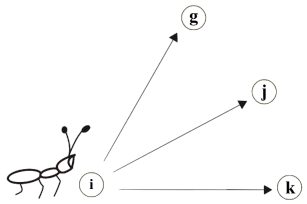

Figure 3.3: Ant at node i chooses a node to move to [Dorigo, 2006]

As the stated at the beginning of this section, the foraging behaviour of ants is a main source of inspiration for ACO algorithms. However, the reader may have noticed that this inspiration is limited. Many features are modified or added to the artificial ants, for example the memory containing which nodes have already been visited, when pheromone trails are deposited and the evaporation of pheromone. Above that, further improvements are realized by incorporating other heuristics, like local search procedures.

3.3.1. DEVELOPMENT

To get a feeling of the development in early ACO algorithms, some essential differences between these algorithms will be mentioned. To keep this comparison manageable and clear, it is limited to the algorithms that are closely related to Ant System. The two most important mathematical relations used in ACO algorithms are the relation describing the probability of moving from node i to node j and the relation describing the pheromone update (deposition and evaporation). Besides all mentioned aspects below, the algorithms contain modifications or extensions with respect to these two relations. The term “best ant” is used below. For explanation, the reader is referred to section 4.3. All differences are mentioned with respect to Ant System.

- In Ant System (AS) a blacklist is used. This blacklist contains all visited nodes. In choosing a node, the nodes on the black list are excluded. The blacklist thus prevents loops in the constructed path of an ant. In AS all ants are allowed to deposit pheromone on their constructed path.
  - In Ant Colony System (ACS) only the best ant is allowed to deposit pheromone on its corresponding best path. ACS also makes use of a candidate set. This candidate set contains the nodes that are close to a certain node. In choosing a node, the ants in the candidate set are considered first. Only if all candidate nodes are visited, the remaining nodes are considered.
  - In Max-Min ant System (MMAS) the pheromone trail intensities are restricted within a minimum and maximum value. Above that, only the best ant is allowed to deposit pheromone on its corresponding best path. Furhtermore, MMAS makes use of a pheromone trail smoothing mechanism.
  - In Rank-Based AS only the best ant is allowed to deposit pheromone on its corresponding best path. In this algorithm all solutions are ranked according to their cost. The pheromone deposition is directly proportional to the rank of the ant. The better the ranking, the more pheromone is deposited.
  - In Approximated Non-deterministic Tree Search (ANTS) a moving average of the cost over the iteration best solutions is registered. If the cost of an ant is lower than the moving average, pheromone concentrations are reduced. If the costs of an ant is higher than the moving average, pheromone concentrations are increased.

Table 3.1: Non-exhaustive list of early ACO algorithms [Dorigo, 2006]

Ant System	Dorigo et al.	1991
Elitist Ant System	Dorigo et al.	1992
Ant-Q	Gambardella and Dorigo	1995
Ant Colony System	Dorigo and Gambardella	1996
Max-Min Ant System	Stutzle and Hoos	1996
Rank-Based Ant System	Bullnheimer et al.	1997
Approximated Non-deterministic Tree Search	Maniezzo	1999
BWAS	Cordon et al.	2000
Hyper-cube Ant System	Blum et al.	2001

3.3.2. STAGNATION

At the end of section 3.3, it is mentioned that evaporation of pheromone is a feature added to artificial ants. Initial experiments with the double bridge (see section 3.2) indicated that ants rapidly converge to a solution and little time is spent on exploring alternative routes [Engelbrecht, 2005] (page 373). To force artificial ants to explore alternative routes, pheromone evaporation is performed every iteration. Pheromone evaporation causes ants to (partly) forget their search history.

The main problem with AS was that search stagnated (converged) prematurely for more complex problems, despite the presence of pheromone evaporation. A prematurely stagnated search means that the search for better solutions concentrates too early around suboptimal solutions. All ants follow the same route and construct the same solution over and over again, such that better solutions will not be found anymore [Stützle and Hoos, 2000]. If at each node the intensity of the pheromone trail of one of the edges is significantly higher than for all others, premature stagnation may occur. In section 4.5 a mathematical formulation is given to identify stagnation. Whether the detected stagnation is premature is often unknown, since for most problems the optimal solution is unknown.

The algorithms developed after AS addressed the problem of premature stagnation. It was attempted to provide a better exploitation-exploration trade-off. So on one hand, the search process should favour actions that were found and proven effective in the past, thereby exploiting the obtained knowledge about the search space. On the other hand, the search process has to investigate previously unseen actions, thereby

exploring the search space [Engelbrecht, 2005] (page 376). So pheromone evaporation favours exploration on alternative routes. However, as already stated in this subsection that is not sufficient. The algorithms developed after AS all provide one or more different additional methods to address premature stagnation. Stützle and Hoos [2000] states that the key to achieve best performance of ACO algorithms is to combine improved exploitation of the best solutions found during search with an effective mechanism for avoiding premature stagnation.

3.4. SELECTION OF ACO ALGORITHM

After the first ACO algorithm was proposed in 1991, it has been widely concerned by scholars continuously aiming for improvements [Zeng et al., 2012]. This led to a tremendous number of different ACO algorithms in literature, from which a few are listed in Table 3.1. For this research, one ACO algorithm has to be selected. This selection is mainly based on findings in literature.

The main successful algorithms proposed are Ant Colony System (ACS) and Max-Min Ant System (MMAS) [Zhang and Feng, 2009], [Dorigo, 2006]. It is shown that ACS and MMAS outperform AS [Stützle and Hoos, 1997], [Dorigo and Gambardella, 1997]. Often, other proposed algorithms take ACS or MMAS as a baseline for comparison of performance with their proposed algorithm. These algorithms show improvement with respect to ACS and MMAS, e.g. [Guo, 2006], [Zhang et al., 2007], [Hong and Bian, 2008] and [Zhang and Li, 2008]. However, no consensus which performs best on the TSP is found in literature. Above that, the complexity increases with every improvement.

Therefore, the selection of the algorithm for implementation is limited to ACS and MMAS. Both require comparable user-specified parameters values. Stützle and Hoos [1997] states that MMAS performs at least at the same level of performance as ACS and in most cases gives a better average performance. Zhang and Feng [2009] states that MMAS is the most widely used ACO algorithm. If quantitative results for runs of the algorithm are compared, MMAS often performs better than ACS, e.g. [Zhao et al., 2008], [Zhang and Feng, 2009], [Chang et al., 2009], [Zhou et al., 2012].

All of this made to select MMAS for implementation. The arguments to select MMAS proposed in this section can be summarized as follows:

- Belongs to most successful algorithms.
- Often used as benchmark algorithm.
- Not too complex.
- Slight preference in performance with respect to ACS.

4

MAX-MIN ANT SYSTEM

The MMAS algorithm discussed in this chapter is based on the papers by Stützle and Hoos [1996], Stützle and Hoos [1997] and Stützle and Hoos [2000]. MMAS is an improved version of AS. Adjustments are made, but some elements remained the same. One of these adjustments is the usage of limits on the pheromone trail intensities. This clarifies the name of the algorithm. Stützle and Hoos [2000] states that MMAS has been specifically developed to combine improved exploitation of the best solutions found during search with an effective mechanism for avoiding premature stagnation.

The first sections of this chapter gives an overview of the MMAS algorithm. During reading of the sections of this chapter, the reader is able to locate the specific section in the algorithm. As stated in section 3.3, the two most important mathematical relations used in ACO algorithms are the relation describing the probability of moving from node i to node j and the relation describing the pheromone update (evaporation and deposition). These two relations and the principles involved are discussed in the subsequent sections: section 4.2 and section 4.3. In that point of view, the remaining sections provide refinements with respect to these relations: section 4.4 and section 4.5 provide refinements for the pheromone trail update, while section 4.6 provides a refinement for selection of a node. In section 4.7 the extension of MMAS with local search is discussed. The last section considers initialization of the algorithm. In this chapter a few times is referred to premature stagnation and the exploitation-exploration trade-off. For more information on these subjects, the reader is referred to subsection 3.3.2.

4.1. OVERVIEW OF ALGORITHM

After initialization of some parameters and the pheromone trails, a main loop is repeated until a termination conditions is met. This termination condition could be a maximum number of iterations for example. In the main loop, first the ants construct each a tour. In choosing a node, an ant chooses among the nodes which not have been visited yet. So if the ant returns to its home node, every node is visited exactly once. Subsequently, the performance of the ants (in terms of solution quality) is computed and compared. Then local search is applied to improve the constructed solutions. Then the pheromone trails are updated. Finally, two steps addressing premature stagnation are performed. All these steps enfold one iteration.

Initialize parameters
Initialize pheromone trails
repeat
    Initialize ants randomly at home node
    Construct complete tours for ants (section 4.2)
    Compute and compare the performance of ants
    Apply local search (section 4.7)
    Update the pheromone (section 4.3)
    Constrict pheromone to be within pheromone trail limits (section 4.4)
    If stagnation is detected, then apply pheromone trail smoothing (section 4.5)
    Number of iterations = Number of iterations + 1
until termination condition is met

4.2. PROBABILITY OF SELECTING NODE

For initialization, each ant is settled on some randomly selected node. Engelbrecht [2005] (page 378) states that this improves the exploration ability of the search algorithm. Each ant builds a TSP route in constructive steps from and to this node. For each single ant, the next node is selected by means of a probabilistic function. The probability that a certain edge is selected is proportional to the pheromone trail intensity of that edge and inversely proportional to the length of that edge. The mutual influence is determined by setting two parameters. The function for the probability of moving from node i to node j [Stützle and Hoos, 1996]:

$$p_{ij}(t) = \begin{cases} \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{k \text{ not visited}} \tau_{ik}^{\alpha}(t)\eta_{ik}^{\beta}(t)} & \text{if node } j \text{ is not visited yet,} \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where $\tau_{ij}(t)$ represents the intensity of pheromone trail on edge (i, j) at time step t. One could see this variable as the memory that contains previous search experience. $\eta_{ij}(t)$ represents the local heuristic function (a priori fixed value) for edge (i, j) at time step t. η_{ij} is defined as $1/d_{ij}$, where d_{ij} is the distance between nodes i and j. So p_{ij} is higher for lower d_{ij}. The sum in the denominator ensures that the sum of all probabilities of all "outgoing" edges sums up to 1. α and β represent the influence of the trail intensity and the heuristic information respectively. $\alpha \geq 0$, if $\alpha = 0$, the pheromone information is neglected. $\beta \geq 0$, if $\beta = 0$, the attraction to certain edges generated by the heuristic information is neglected. Node j may not be visited yet, because literature prescribes that loops must be prevented (see subsection 3.3.1). Engelbrecht [2005] (page 376) states that Equation 4.1 effectively balances the exploitation-exploration trade-off. The best balance between these two is achieved trough proper selection of α and β. Node k is selected from a set of feasible nodes. This set of nodes only contains immediate neighbour nodes of node i, like shown in Figure 3.3. To recognize nodes that are already visited, a list with visited nodes should be maintained for each ant.

4.3. PHEROMONE TRAIL UPDATE

The pheromone trail intensity of an edge used in the next iteration is equal to the current pheromone trail intensity times a factor plus a pheromone deposition. The pheromone trail is updated according to [Stützle and Hoos, 1996]:

$$\tau_{ij}(t) = \rho\tau_{ij}(t-1) + \Delta\tau_{ij}^{b} \tag{4.2}$$

where ρ is the persistence of the pheromone trail ($0 < \rho < 1$), so $(1-\rho)$ is the evaporation. Stützle and Hoos [1996] states that the evaporation mechanism helps to avoid unlimited accumulation of the pheromone trails. If an arc is not selected by ants, the evaporation enables forgetting "bad" solutions over time. So evaporation takes place on all arcs. $\Delta\tau_{ij}^{b}$ is the amount of pheromone added to edge (i, j) by the best ant moving over that edge. So the notation of b is not an exponent, but a superscript. $\Delta\tau_{ij}^{b}$ is defined as $1/L_b$, where L_b is the length of the best tour. For selecting 'best' there are different possibilities. Possibilities are selecting the best ant/tour of a single iteration (iteration-best, ib) or the best ant/tour of all performed iterations (global-best, gb). Stützle and Hoos [1996] states that iteration-best favours a higher degree of exploration, whereas global-best may lead to premature stagnation. In addition, Stützle and Hoos [2000] states that in general, using exclusively global-best not seems to be a very good idea in the case of MMAS. In experiments it is shown that the results for iteration-best are significantly better than for global-best [Stützle and Hoos, 1996]. However, one could also think of mixed strategies of iteration-best and global-best.

4.4. PHEROMONE TRAIL LIMITS

Stützle and Hoos [2000] states that one way to avoid premature stagnation is to influence the probability for choosing the next node. From Equation 4.1 it can be seen that this directly depends on the pheromone trail intensity and the heuristic information. The heuristic information as defined is problem-specific and fixed a priori. However, the dependency on the pheromone trail intensities can be influenced. This provides a tool to influence the relative influence during running the algorithm. This is achieved trough imposing explicit limits on the pheromone trails intensities for all edges [Stützle and Hoos, 1996]:

$$\tau_{min} \leq \tau_{ij} \leq \tau_{max} \tag{4.3}$$

These limits clarify the name of the algorithm and can be seen as a refinement of the pheromone trail update. It can be formally derived that the maximum possible pheromone intensity is asymptotically bounded [Stützle and Hoos, 2000]. Above that, it is shown that good values for the minimum pheromone intensity can be found. The equations for the upper and lower bound are given in Equation 4.4 and Equation 4.5 [Stützle and Hoos, 1996].

$$\tau_{max}^{theo} = \frac{1}{1-\rho}\frac{1}{L_{opt}} \tag{4.4}$$

$$\tau_{min} = \frac{\tau_{max}(1 - \sqrt[n-1]{p_{best}})}{(n/2)\sqrt[n-1]{p_{best}}} \tag{4.5}$$

where L_{opt} is the optimal tour length for the TSP and therefore gives the theoretical value of τ. If the optimal solution is not known, then L_{opt} is substituted by L_{best}^{global}. Thus, τ_{max} is adapted during the algorithm and is time dependent. n is the number of nodes in the graph. Stützle and Hoos [2000] states that when MMAS has converged, the best solution found is constructed with a probability p_{best}. In this situation, an ant constructs the best solution found if it makes at each "choice point" the "right" decision by choosing a solution component with maximum pheromone trail intensity τ_{max}. Engelbrecht [2005] (page 385) states that p_{best} is a user-specified parameter which needs to be optimized for each new application. Note that $p_{best} < 1$ to ensure that $\tau_{min} > 0$. Also, if p_{best} is too small, then $\tau_{min} > \tau_{max}$. In that case, MMAS sets τ_{min} equal to τ_{max}. Using Equation 4.3 and Equation 4.1 it can be seen that this case corresponds to usage of the heuristic information η_{ij} only in the solution construction.

Experiments are performed by Stützle and Hoos [2000] to investigate the influence of lower trail limits. It is shown that it is advantageous to use lower trail limits in terms of lower derivation of the optimal solution. Above that, it is stated that the importance of lower trail limits increases with increasing problem instance. τ_{max} is initialized at an arbitrarily high value, τ_{min} is initialized at 0.

4.5. PHEROMONE TRAIL SMOOTHING

Stützle and Hoos [1997] states that despite pheromone trail intensities are bounded, long runs of the MMAS still can stagnate prematurely. To address this problem, pheromone trail smoothing (PTS) is introduced. PTS can be seen as a refinement of the pheromone trail update. The reasoning proposed by Stützle and Hoos [1997] is that if premature stagnation occurs, not enough new tours are explored. Thus, the TPIs need to be adjusted such that new tours are explored (again) in a higher extent. When MMAS has converged are is almost going to, the pheromone trail intensities are increased proportionally to their difference with the maximum pheromone trail limit [Stützle and Hoos, 1997]:

$$\Delta\tau_{ij}(t) \propto (\tau_{max}(t) - \tau_{ij}(t)) \tag{4.6}$$

The learned trails so far are not completely forgotten. By increasing the TPIs in this way, and thus reducing (smoothing) the difference between high and low TPIs, exploration is stimulated. With this mechanism the solution quality for longer runs increased significantly [Stützle and Hoos, 1997]. To be able to apply PTS, stagnation needs to be identified. The λ-branching factor of node i (λ_i) is defined as the number of edges leaving node i with a TPI higher than [Gambardella and Dorigo, 1995]:

$$\lambda\delta_i + \tau_{i,min}, \text{ with} \tag{4.7}$$

$$\delta_i = \tau_{i,max} - \tau_{i,min} \tag{4.8}$$

where λ is a parameter, $0 \leq \lambda \leq 1$. $\tau_{i,max}$ is the maximum TPI of all edges leaving node i. $\tau_{i,min}$ is the minimum TPI of all edges leaving node i. Stagnation is identified with the mean λ_i-branching factor, as proposed

by Gambardella and Dorigo [1995]. The mean λ_i-branching factor gives an indication of the dimension of the search space and is defined as:

$$\frac{\sum_i \lambda_i}{n} \tag{4.9}$$

Stützle and Hoos [1997] states that he has found that for a value of $\lambda = 0.05$ and a mean 0.05-branching factor very close to 1, only very few (often only one) arc exit from a node have a very high selection probability and practically no new solutions are explored. This is assumed as stagnation. As stated in subsection 3.3.2, whether the detected stagnation is premature is often unknown, since for most problems the optimal solution is unknown.

4.6. CANDIDATE SETS

A large part of the run times of MMAS is due to the complexity of the iterations. Therefore it pays off to reduce the running times of these iterations. One possibility is the usage of candidate sets. Candidate sets can be seen as a refinement of the selection of a node. Stützle and Hoos [1997] states that this reduces the computation times by far. Above that, it also has an positive influence on the performance of MMAS. Especially when the number of iterations or time is limited.

The number of edges that needs to be considered every constructive step contributes to the complexity of these constructive steps. Standard implementation of an algorithm considers all the edges. However, with the naked eye one could see that most of the edges are not worth considering. Because they are too long to occur in short tours. Candidate sets basically reduce the number of edges that are considered to the promising ones. This reduced graph is called a sub graph. Before the algorithm starts, two sets are created for each node: a candidate set (often referred to as nearest neighbours) and a remainder set. The candidate set contains the promising nodes and the remainder set the remaining nodes. When constructing a tour, an ant chooses probabilistically the next node among the nodes in the candidate set according to Equation 4.1. If all the candidate nodes are already visited, the ant chooses among the nodes in the remainder set. In this latter case, the ant deterministically chooses the node for which the following equation is maximum [Stützle and Hoos, 2000]:

$$\tau_{ij}^{\alpha} \eta_{ij}^{\beta} \tag{4.10}$$

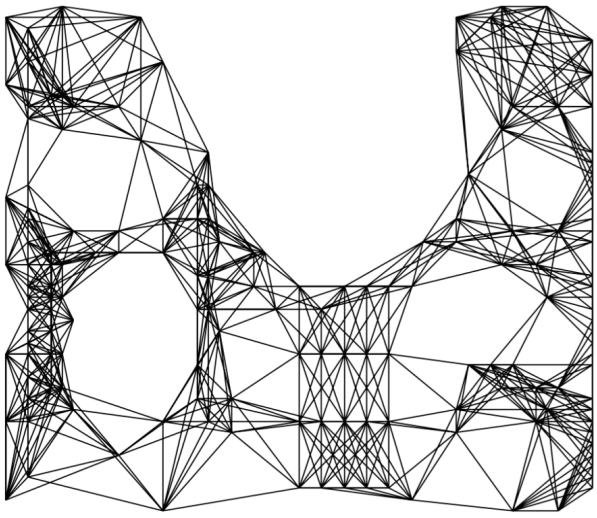

Figure 4.1: The 10 nearest neighbour sub graph for u159 [Reinelt, 1994] (page 64)

To speed up pheromone update (see section 4.3), Stützle and Hoos [2000] proposes to only apply pheromone evaporation to the arcs which are incorporated in the candidate sets. If this is implemented, it is important to notice that pheromone can still be deposited on all edges, and thus τ is correctly present in Equation 4.10.

However, it is not denoted that this also influences the identification of stagnation. The PTI on the arcs that are not incorporated in candidate sets can only increase due to pheromone deposition and thus not reach a value below the bound of Equation 4.7. So a mean 0.05-branching factor that is very close to 1 to identify stagnation is not applicable and an other value must be set.

There are different algorithms to determine the candidate sets, like exact methods (e.g. Delaunay triangulation) and heuristic methods. It might be interesting to notice that an optimal tour can be found within a low number of nearest neighbours. For example, an optimal solution for the problem pr2392 (see Appendix A and section 5.8) can be found within the 8 nearest neighbours and an optimal for the problem pcb442 (see Appendix A and section 5.8) can be found even within the 6 nearest neighbours [Reinelt, 1994] (page 64). An example of a sub graph with the 10 nearest neighbours is shown in Figure 4.1. This sub graph does contain an optimal route.

4.7. LOCAL SEARCH

Stützle and Hoos [1996] states that it is possible to find near optimal solutions for TSPs using only tour constructions with MMAS, however it cannot compete with more specialized algorithms for the TSP. To improve the tours constructed by ants, the MMAS algorithm is extended with local search. The goal of local search is faster convergence and an earlier detection of high quality solutions. Examples of local search heuristics are 2-opt exchange, 3-opt exchange and variable depth search ([Lin and Kernighan, 1973]).

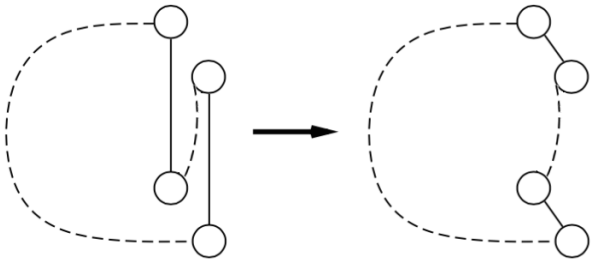

Figure 4.2: Example of a 2-opt exchange [Reinelt, 1994] (page 106)

After a solution route is constructed for a TSP instance, local search may be applied. So local search may be applied after every iteration. By altering the solution, the heuristic looks for an improvement. A closer look will be taken at 2-opt exchange, since this form of local search is the easiest to explain. In short, a 2-opt move consists of eliminating two edges and reconnecting the two resulting paths in a different way to obtain a new route (see Figure 4.2). The two edges that are eliminated may not have a node in common. It is important to notice that there is only one way possible to create a new route. Above that, a part of the new route will be passed in opposite direction in comparison with the old route. If the sum of the length of the new two edges is smaller than the length of the old two edges, this may be an improvement.

The number of possible 2-opt moves is given in Equation 4.11. For every node i, all the 2-opt exchanges with the edge between node i and its successor node in the route must be considered. If one or multiple improvement(s) are found, the best 2-opt exchange must be applied. So searching for 2-opt moves takes time $\mathcal{O}(n^2)$: all pairs of edges in route must be considered. As stated earlier, local search may be applied after every iteration. Similar to pheromone trail update (see section 4.3) one could think of different strategies for which ants are allowed to apply local search. For example, all the ants, only the iteration best ant or a mixed strategy. Above that, one could think of to limit the search for beneficial 2-opt moves, since some pairs of edges have more potential than others. For every node i, only the 2-opt exchanges for the edges connecting to nearest neighbours of node i are considered.

$$\frac{n(n-3)}{2} \tag{4.11}$$

4.8. INITIALIZATION

4.8.1. PHEROMONE TRAIL

The pheromone trail intensities are initialized by setting τ_{ij} to some arbitrarily high value. After the first iteration, the pheromone trail intensities will be set to τ_{max} by the imposed upper bound. A reasoning for this initialization is proposed by Stützle and Hoos [2000].

Taking the ratio of pheromone trail intensities from the zeroth and first iteration, it can be seen that this ratio is a factor ρ for very large initial value of τ_{ij}. For the second iteration it is a factor ρ^2 etc. For a smaller initial value of τ_{ij}, the ratio increases. Thus, for a smaller initial value of τ_{ij}, the relative difference between trail intensities increases more strongly. However, it is desired that the selection probabilities from Equation 4.1 evolve more slowly to support exploration during the first iterations. Hence, τ_{ij} is initialized by setting it to some arbitrarily high value.

One might have noticed that the candidate sets from section 4.6 can be seen as a form of initial solution. One could also think of creating an initial solution by depositing pheromone on certain edges. However the pheromone trail intensities on all the edges are initialized by setting τ_{ij} to some arbitrarily high value. This can be explained by the exploitation-exploration trade-off and the principle of premature stagnation. Stützle and Hoos [2000] states that the key to achieve best performance of ACO algorithms is to combine improved exploitation of the best solutions found during search with an effective mechanism for avoiding premature stagnation. One of the key aspects of MMAS is to initialized pheromone trails deliberately to achieve a higher exploration of solutions at the start of the algorithm. An initial solution in the form of depositing pheromone would undermine this aspect.

4.8.2. PARAMETERS

Various parameters are required at the beginning of the MMAS algorithm. Stützle and Hoos [1996] and Stützle and Hoos [2000] performed experiments with several parameter values. Stützle and Hoos [1996] states that ρ and m are the two largest instances.

Concerning ρ, this variable was varied between 0.7 and 0.99. It is observed that for a low number of iterations better (in terms of derivation of optimal solution) tours are found by using lower values of ρ. Stützle and Hoos [2000] states that this is due to the fact that for lower values of ρ the pheromone trails which are not reinforced decrease faster and therefore the search concentrates earlier on the best tours found so far. If ρ is high, too few iterations are performed to reach significant differences between the pheromone trails on arcs in high quality tours and those which are not. On the other hand, for a larger number of iterations better tours are found by using higher values of ρ.

Concerning α and β, these variables were varied between 1.0 and 5.0. These variables have considerable influence on the results. Stützle and Hoos [1996] states that high values for α have a negative influence on the performance of MMAS. On the other hand, higher values for β do not have much impact, and values around 1.0 to 5.0 seem appropriate. It is observed that optimal values for β are problem specific, while $\alpha = 1.0$ always seems best.

Concerning p_{best}, as stated earlier: this is a user-specified parameter which needs to be optimized for each new application. During experiments by Stützle and Hoos [1996] and Stützle and Hoos [2000] used a value of $p_{best} = 0.05$ or 0.005. Stützle and Hoos [2000] performed an experiment by systematically varying p_{best} (and thus the lower pheromone trail intensity) from 0.0005 to 0.5 and one series with $\tau_{min} = 0$. The results of this experiments are shown in Figure 4.3. The average tour length is given over 25 runs, the best results per instance are denoted in italic. Note that smaller values of p_{best} result in tighter trail limits.

Instance	$p_{\text{best}} = 0.0005$	$p_{\text{best}} = 0.005$	$p_{\text{best}} = 0.05$	$p_{\text{best}} = 0.5$	$\tau_{\text{min}} = 0$
eil51	428.5 (0.59%)	428.0 (0.46%)	427.8 (0.43%)	*427.7* (0.39%)	427.8 (0.43%)
kroA100	21344.8 (0.29%)	21352.8 (0.33%)	*21336.9* (0.26%)	21353.9 (0.34%)	21373.2 (0.43%)
d198	16024.9 (1.55%)	15973.2 (1.22%)	*15952.3* (1.09%)	16002.3 (1.41%)	16047.6 (1.70%)
lin318	42363.4 (0.80%)	*42295.7* (0.64%)	42346.6 (0.75%)	42423.0 (0.94%)	42631.8 (1.43%)

Figure 4.3: Computational results for different lower pheromone trail limits [Stützle and Hoos, 2000]

Concerning the number of ants m, this is an important parameter since the more ants are used, the more paths have to be constructed, the more pheromone deposits have to be calculated etc. But, ants communicate their experience about the search space with other ants. So the fewer ants are used, the less possibilities for the algorithm for exploration. This may cause bad solution quality. Dorigo et al. [1991] found with experiments for developing AS that that the optimal number of ants is equal to the number of nodes n. This is also used in the experiments by Stützle. Other applications in literature also tend to use this number of ants.

Concerning the candidate sets, Stützle and Hoos [1996] states that the size of a candidate set usually comprises between 10 to 30 nodes. During his experiments, a candidate list size of 20 nodes is used. Also in the experiments by Stützle and Hoos [2000] a candidate set of 20 nodes is used.

5
MODEL SETUP

In chapter 4 several variants of MMAS are presented. Several elements of this algorithm can be implemented in multiple ways. This chapter will give insight in how these elements are implemented and why these are selected. The corresponding sections are given the same name. In section 4.1 an overview of the MMAS algorithm was given. In addition to this, an overview of the implemented algorithm will be given. This should give a clear overview of the computer program. For completeness, a short introduction will be given on the input and output facilities of the computer program.

5.1. LOOPS

This section does not directly relate to a section of previous chapter. Since in a Hamiltonian cycle every node must be visited exactly once, it is necessary that no loops are included. In the implementation of MMAS the route of an ant is forced to a Hamiltonian cycle. So during selection of a node (see Equation 4.1), already visited nodes are excluded by means of a blacklist. Therefore, if the ant returns to the home node, no loops are included in the route. The implementation with a blacklist is prescribed by the literature concerning MMAS. However, one could also think of not forcing the route to a Hamiltonian cycle. The loops then have to be eliminated after the ant returned to the home node. Both approaches have their advantages and disadvantages. In literature no decisive answer is found which approach is best.

5.2. PHEROMONE TRAIL UPDATE

For depositing pheromone, a 'best' ant must be selected. It was chosen to use a mixed strategy of iteration-best and global-best ants. Experiments on larger instances have shown that using mixed strategies may yield a faster convergence of the algorithm and produce improved results [Stützle and Hoos, 2000]. It is stated that the best performance was obtained by using a mixed strategy with increasing usage of global-best ant over time. The following schedule is applied as proposed by Stützle and Hoos [2000], where f^{gb} is the frequency of usage of the global best ant to update the pheromone trail. So in the first 25 iterations only the iteration best ant is used. By gradually shifting the emphasis from the iteration-best ant to the global-best ant in updating the pheromone, a transition from a stronger exploration to a stronger exploitation is established. To be complete, the iteration-best only and global-best only strategies are also implemented.

Table 5.1: Schedule mixed strategy pheromone trail update

Iteration range	f^{gb}
0 - 25	0
26 - 75	5
76 - 125	3
126 - 250	2
251- ∞	1

5.3. PHEROMONE TRAIL SMOOTHING

In section 4.5 it is stated that for a value of $\lambda = 0.05$ and a mean 0.05-branching factor very close to 1, only very few (often only one) arc exit from a node have a very high selection probability and practically no new

solutions are explored [Stützle and Hoos, 1997]. This is assumed as stagnation. If the development of lambda mean is considered (see Figure 5.1), it can be seen that firstly lambda mean is stable on $n-1$. Subsequently, it decreases and could decrease from $n-1$ to 2 (two edges with PTI higher than Equation 4.7). Lambda mean first is stable on $n-1$ because none of the exiting edges from a node has a PTI lower than the bound mentioned in Equation 4.7. After a certain number of iterations, the PTI on all the edges that are not used up and to that moment has decreased below the bound. Consequently, lambda mean decreases significantly at that instance. How fast lambda mean decreases, depends on the selection the value of ρ. The higher the value of ρ, the faster lambda mean decreases. Often, lambda mean does not reach a value of 2 but fluctuates around a certain low value. Since a mean 0.05-branching factor very close to 1 to identify stagnation is not completely clear, a closer look is taken at this aspect. Several runs of TSP instances indicated that a mean 0.05-branching factor smaller than 2.5 is appropriate to identify stagnation. This value is implemented. From Figure 5.1 it can clearly be seen that stagnation is identified in iteration 30 and that PTS is applied.

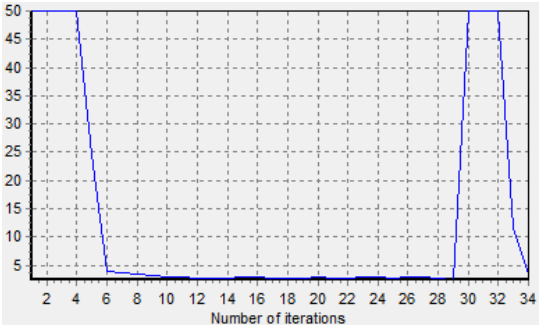

Figure 5.1: Example of development of lambda mean for TSP instance eil51

5.4. CANDIDATE SETS

Prior to the start of the algorithm, for a candidate set is determined for each node. A relative simple (with respect to e.g. Delaunay triangulation) heuristic as proposed by Reinelt [1994] (page 70) is implemented to determine the candidate sets. The basic idea is obtain the nearest neighbours based on their coordinates. The heuristic will be introduced only for the horizontal x-coordinates. First all the nodes are sorted in a list, based on their x-coordinate. For every node i, the nodes just above and just below are selected. This is realized by introducing search width w. Only the w nodes above and below node i are selected. Among these selected nodes, the k nearest neighbours are computed. The corresponding edges are marked as candidate edges. In the complete heuristic, vertical coordinates are also taken into account. The complete heuristic is described below. It might be helpful to notice that the algorithm works with overlap of quadrants in the x-y plane. Please note that if k is specified, that number of nearest neighbours is not necessarily found for every node. Referring back to Figure 4.1, the candidate sets obtained with $k = 10$ and $w = 20$ for this instance contained 94% of the 10 nearest neighbours shown [Reinelt, 1994] (page 70).

1. Two lists are created. One list contains the nodes sorted with respect to their x-coordinate. The other list contains the nodes sorted with respect to their x-coordinate. Let i_x and i_y denote the positions of node i these lists respectively.
2. For every node $i = 1, 2, \ldots, n$ do:
   (a) Create four sets: Q_1, Q_2, Q_3, Q_4
   (b) $Q_1 = \{j | j_x \in \{i_x + 1, ..., i_x + w\}$ and $j_y \in \{i_y + 1, ..., i_y + w\}\}$,
   $Q_2 = \{j | j_x \in \{i_x + 1, ..., i_x + w\}$ and $j_y \in \{i_y - 1, ..., i_y - w\}\}$,
   $Q_3 = \{j | j_x \in \{i_x - 1, ..., i_x - w\}$ and $j_y \in \{i_y - 1, ..., i_y - w\}\}$ and
   $Q_4 = \{j | j_x \in \{i_x - 1, ..., i_x - w\}$ and $j_y \in \{i_y + 1, ..., i_y + w\}\}$.
   (c) Create nearest neighbour set for node i. Add at maximum two nearest neighbours from every set Q_i for $i = 1, 2, 3, 4$ to the nearest neighbour set. Remove those nodes from their corresponding sets Q_i. The number of nodes added/removed in this step is denoted with l.

(d) Compute the $k-l$ nearest neighbours in the reduced set $Q_1 \cup Q_2 \cup Q_3 \cup Q_4$, and add those nodes to the nearest neighbour set of node i.

To guarantee stable behaviour of the nearest neighbour component, obey the following restrictions:

- Number of nearest neighbours $k \geq 8$
- Search width $w \geq k/2$

As stated at the end of section 4.6, to speed up pheromone update Stützle and Hoos [2000] proposes to only apply pheromone evaporation to the arcs which are incorporated in the candidate sets. Since this disables the possibility to use a mean 0.05-branching factor that is very close to 1 to identify stagnation and the pheromone evaporation is a relative simple operation, it is chosen to apply evaporation on all the edges.

5.5. LOCAL SEARCH

To extend the MMAS algorithm, 2-opt exchange is implemented as a local search algorithm. 2-opt exchange is a rather simple heuristic. Stützle and Hoos [1996] states that it is well known that other local search heuristics (like 3-opt exchange or variable depth search) give higher quality routes. 2-opt exchange, 3-opt exchange and variable depth search usually result in an average route length within 8-10%, 5% and 2% respectively from the optimal route length [Stützle and Hoos, 1996]. Since one of the purposes of this research is to explore the potential of ACO, 2-opt exchange is considered sufficient. Above that, 2-opt exchange is easier to implement. A procedure to perform 2-opt exchange is described below [Reinelt, 1994] (page 106).

1. Let T be a route constructed by an ant.
2. Perform the following until failure:
   (a) For every node $i = 1, 2, ..., n$:
   Examine all 2-opt exchanges involving the edge between node i and its successor node in the route. If possible: choose the best 2-opt exchange and update T.
   (b) If no improvements can be found, then declare failure.

A straightforward implementation of a local search procedure is computationally too expensive for larger TSP instances [Stützle and Hoos, 1996]. Therefore, the number of 2-opt exchanges checked for node i is limited to the edges connecting to nodes from the candidate set of node i (as described in section 5.4).

Similar to the pheromone trail update, the trade-off which ants are allowed to apply local search exists. Basically two options can be considered. One option allows all the ants to perform local search after every iteration. The other option allows only the iteration-best ant to improve its route by local search. Stützle and Hoos [1996] performed experiments to compare differences in performance between these two. It is stated that the first mentioned option is preferable for TSPs. During implementation it appeared that allowing all the ants performing local search as described above is computationally expensive. It does not outweigh the gain in improvement of solution quality (for more details on the performance, see tables presented in Stützle and Hoos [1996]). Therefore, the local search heuristic is only implemented for the iteration-best ant.

5.6. TERMINATION CONDITIONS

As stated in section 4.1, the main loop of the MMAS algorithm is repeated until a termination condition is met. The following termination conditions are implemented:

- Number of iterations = specified maximum number of iterations
- Run time = specified maximum duration
- Cost ≤ specified threshold value

The first one is used very often in literature concerning MMAS and other ACO algorithms. Since this research focuses on a comparison of two algorithms, our implementation of MMAS should also have this possibility. The run time is not used often in literature. However, as will be explained in the introduction of chapter 6, during our experiments more emphasis will be on performance with respect to time. The last

termination condition enables to set a threshold value. If the found cost is smaller or equal to this threshold value, the run is terminated. With this option, the run time is unknown a priori. This termination condition is useful when the optimal solution of a TSP instance is known.

5.7. CONCEPTUAL MODEL

In section 4.1 a generic overview of the MMAS algorithm is given. In this section, that algorithm will be translated to a conceptual model. For this conceptual model a few assumptions hold:

- Symmetric (so $d_{ij} = d_{ji}$) TSP instances
- Data part in 2D coordinates
- Euclidean edge weight type (denoted as EUC_2D)

In the conceptual model all classes and associated processes from the implemented MMAS algorithm will be described. Firstly all classes and their attributes are denoted. Secondly, all the processes are described. In order to create a clear overview of the algorithm, the processes are reduced to the highest level of simplicity. Finally, the procedure of loading a TSP map is described to support the comprehension of the algorithm. If possible, references are made to the equations introduced in chapter 4. Please note that the ant class does not have a process and the node and edge class do have a process. This is a design choice.

- Node Class
  - ID
  - x
  - y
  - Occupance queue
  - Candidate nodes list
  - Candidate edges list
  - Remaining (= connected − candidate) edges list
  - Connected may be visited edges list
  - *Process*
- AllNodes Class
  - Node queue
- Edge Class
  - ID
  - Length
  - Tau
  - Connected nodes list
  - Occupance queue
  - Position in "visited edges list"
  - *Process*
- AllEdges Class
  - Edge queue
- Ant Class
  - Home node
  - Last visited node
  - Cost
  - Visited edges list
  - May not be visit edges list
  - Number of constructive steps
- UpdateManager Class
  - *Process*

- Node class process
  repeat
  begin
    for all ants on node determine next edge
      if not last edge
        if not all nearest neighbours visited, choose among nearest neighbours:
        1. determine stochastically which edges of connected edges may be visited and the denominator of Equation 4.1
        2. determine which edge to be selected (Equation 4.1)
        else choose among remaining edges:
        1. determine deterministically which edge to be selected (Equation 4.10)
      if last edge, find edge to home node
        if not all nearest neighbours visited, search among nearest neighbours
        else search among remaining edges
        NewIteration = true
    move ant to selected edge
    proceed 1 time unit
  end

- Edge class process
  repeat
  begin
    for all ants on edge
      update data of ant: cost and visited edges
      move ant to next node
    proceed 1 time unit
  end

- Update manager class process
  repeat
  begin
    if NewIteration = true then
    begin
      find iteration best path among ants
      apply 2-opt local search to iteration best ant
      check for new global best
      update pheromone limits (Equation 4.4 and Equation 4.5)
      destroy ant
      evaporate pheromone (Equation 4.2)
      deposit pheromone on path: 3 possible strategies (Equation 4.2)
      constrict pheromone (Equation 4.4 and Equation 4.5)
      identify stagnation (Equation 4.9)
      if stagnation is identified then apply pheromone trail smoothing (Equation 4.6)
      create new ants
      NewIteration = false
    end
    proceed 1 time unit
  end

- Load TSP map procedure
  begin
    create nodes
    determine k nearest neighbours for search width w for every node
    determine k − l nearest neighbours
    create edges
      attach candidate edges to node based on candidate nodes
      attach remaining (=connected − candidate) edges to nodes
  end

5.8. TSP LIBRARY

The TSP library was introduced in section 2.2. As indicated, in this research the TSP LIB will be used. The TSP LIB rounds the calculated distances to the nearest integer, since most of the written applications require this. For further and detailed information on the TSP LIB, one could read the complete description of the library (named DOC.PS).

All the TSP instances are provided in a file with extension .tsp. Delphi 2010 is not directly able reading files with this extension. Therefore, in this case the file must be converted. One way of doing this is as follows. The file can be opened with Wordpad (text editor for Microsoft Windows). Notepad could also be used, but does not show the incorporated layout of the file. Then the file must be saved as a text file (.txt). The application is able to read coordinates in scientific notation. All the other information must be deleted. Above that, negative coordinates are not allowed. Also take care that no white lines are inserted, especially at the end of the file. A folder with all adjusted and suitable TSP instances is incorporated with the program. Own TSP instances can be used, as long as they have the same format as the incorporated TSP instances. This means obeying the rules prescribed in this section and for every node a node ID, x-coordinate and y-coordinate on one line.The Euclidean distance between a pair of nodes is calculated as follows:

$$d_{ij} = \text{round}\left\{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\right\} \tag{5.1}$$

An overview of the symmetric travelling salesman problems with their solution bounds (a single number indicates that the optimal solution is known) is given in Appendix A. All the smaller TSP instances ($n < 50$) are given in an other format than EUC_2D. For research purposes it is also desired to investigate these smaller instances. To make these instances readable for the written application, it is chosen to convert the instances with explicit edge weight matrices (denoted as MATRIX) to 2D coordinates. This conversion is done by means of an approximation algorithm called multidimensional scaling (MDS, also known as principal coordinates analysis). An introduction to this technique is given in Appendix C.

5.9. INTERFACE

In this section a short introduction will be given on the interface of the computer program. In Figure 5.2 the control panel is shown. A description of the controls is given below the figure. The computer program generates three output files: a text file and two figures. The text file contains all the information of the performed run, the format of this text file is shown in Figure 5.3. One figure captures the graph indicating history of minimal found cost (as it is part of Figure 5.2). The other figure captures the route of the best route found and is shown in Figure 5.4.

A few additional remarks:

- Concerning radio buttons number 4 (see Figure 5.2), this functionality is implemented because it is useful during model development. With this radio button turned on, the random number generators are pseudo random. Every time the same sequence of random numbers is generated and thus is predictable. During normal usage of the algorithm, it should be turned off.
- It is not possible to turn off the nearest neighbours functionality. This is because it was too entwined with the rest of the code.

- The number of nearest neighbours also influences the local search procedure. The more nearest neighbours are set, the local search procedure becomes computationally more expensive.
- There is no possibility for animating the movement of the ants. As described in the introduction of section 5.7, the ant class does not have a process. A disadvantage of this choice is that animation of the movement of the ants is more complicated.

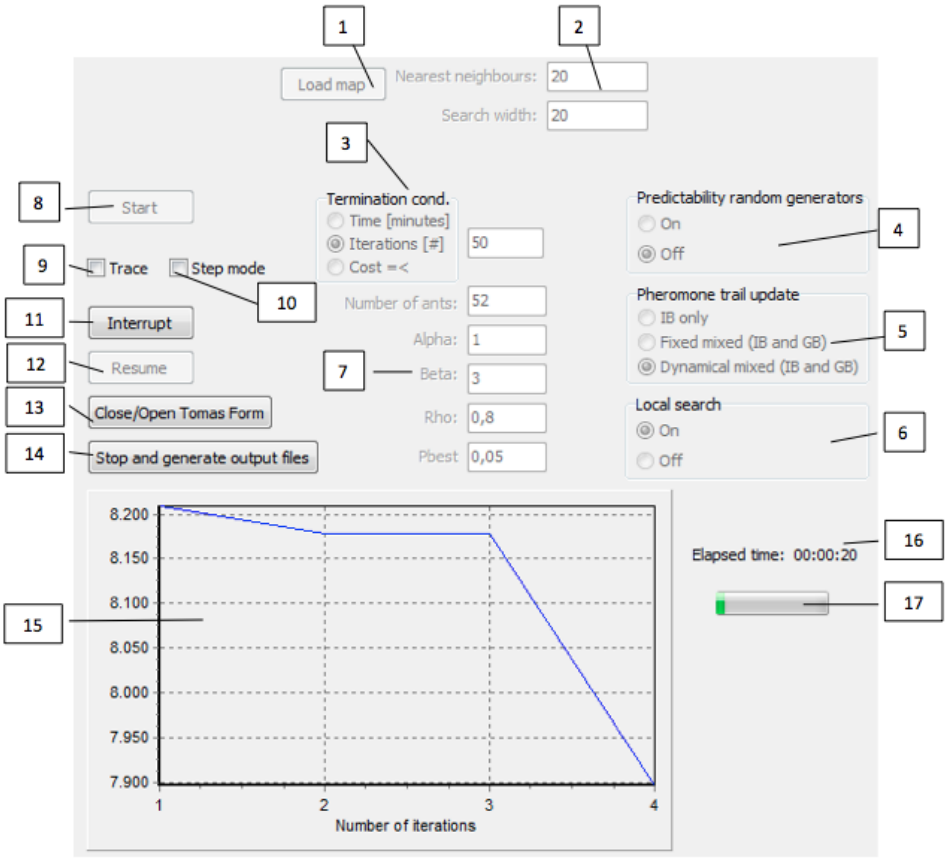

Figure 5.2: Control panel

1. Button for loading TSP map.
2. Two edit boxes for nearest neighbour settings (see section 4.7 and section 5.5).
3. Radio buttons and edit box for setting termination condition.
4. Radio buttons for enabling/disabling predictability of the random number generators.
5. Radio buttons for selecting pheromone trail update strategy (see section 4.3 and section 5.2).
6. Radio buttons for enabling/disabling local search (see section 4.7 and section 5.5).
7. Five edit boxes for setting parameters (see subsection 4.8.2).
8. Button to start run.
9. Check box for enabling/disabling trace function of TOMAS.
10. Check box for enabling/disabling step mode of TOMAS.
11. Button to interrupt run.
12. Button to resume run.
13. Button to close/open TOMAS form.
14. Button to stop run and generate output files (see Figure 5.3 and Figure 5.4).
15. Graph indicating history of minimal found cost.
16. Timer indicating elapsed time.
17. Progress bar indicating progress with respect to set termination condition.

```
///// DURATION /////

Started:
Stopped:
Elapsed [hh:mm:ss]:
Termination condition (cost) =<

///// TSP MAP /////

File:
Number of nodes:
Number of edges:

///// SETTINGS /////

Number of ants:
Alpha:
Beta:
Rho:
Pbest:
Predictability random generator:
Pheromone trail update:
Nearest neighbours:
Search width:
Local search:

///// RESULTS /////

Minimal found cost:
Found in iteration:
Found after [hh:mm:ss]:
Number of iterations performed:
Number of times stagnation detected:

Path (edge IDs):
```

Figure 5.3: Format text file

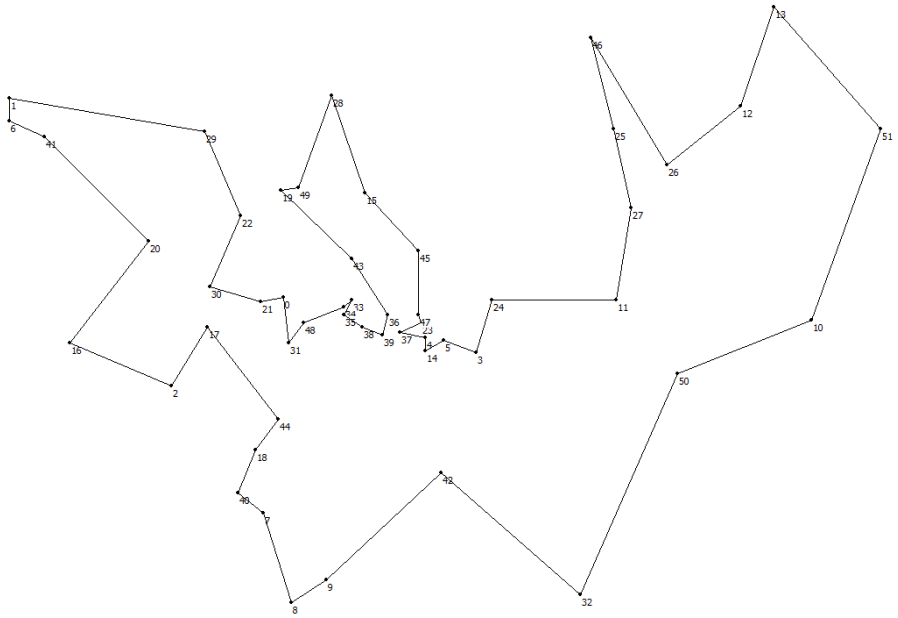

Figure 5.4: Best found route

5.10. TOMAS

As indicated in section 1.3 discrete simulation package TOMAS [Veeke and Ottjes, 2010] will be used in combination with Delphi 2010. However, the runs performed with the implemented MMAS algorithm are not discrete event simulations. It does not matter if all the ants that belong to one iteration move around simultaneously or one after one. Also, there is no time interaction. An ant does not have to wait because of the ant in front of him. In other words, there is no process interaction. In essence is a run of an ACO algorithm (as presented in chapter 4) a (static) Monte Carlo experiment. It relies on random sampling and runs are repeated many times in order to obtain numerical results. The reason why TOMAS still is used is that it provides many useful tools to process elements like queues. Further clarification may be found in the conceptual model of section 5.7.

6
EXPERIMENTS WITH MMAS

In this section various experiments will be performed with the implemented MMAS algorithm. The main purpose of these experiments is to evaluate its performance and compare it with B&B. For verification of the implemented MMAS algorithm, some experiments performed in literature with MMAS are repeated. In this research verification and validation are intermingled, because for most instances of the TSP library the optimal solution is known. This optimal solution should correspond to a virtual reality and thus also enables validation. The optimal solution corresponds to a virtual reality because the formulation of a TSP instance is a model of reality and a model will never exactly correspond to reality. In the available literature concerning MMAS, extensive experiments are performed on parameter settings, model setups and settings. However, the objective of this research is not to repeat these experiments. Certain settings and setups emerged to let MMAS perform better than others, as indicated throughout chapter 4 and chapter 5. This information is used for the experiments in this chapter. It was also noted that there was not much emphasis on performance with respect to time. In the view of the objectives of this research introduced in section 1.2, it was chosen to complement research on MMAS with respect to the aspect time performance and make use of the findings of previous research.

6.1. EXPERIMENTAL PLAN

Various experiments are performed in this chapter. Globally they can be divided in three categories:

- Verification
- Parameter setting
- Performance

The first experiment (section 6.2) focuses on verification of the implemented MMAS algorithm. In other words, is the algorithm implemented correctly? Experiments for two TSP instances performed in literature are repeated and compared. As indicated in subsection 4.8.2, the number of ants m is one of the most important parameter settings. In the second experiment (section 6.3), the influence of the number of ants on solution quality and speed is investigated. The remaining experiments focus on performance. On one hand the performance of MMAS is compared with B&B. On the other hand, insight in the performance of MMAS with respect to time is deepened by limiting the number of iterations.

As MMAS is a heuristic algorithm, the solution quality is not constant. Therefore, for each experiment several runs are performed. Resulting in a best solution and an average solution with standard deviation (σ). For most of the experiments an average is taken over 10 runs. For the sake of clarity, an experiment consists of one or multiple series of a number of runs. Each run consists of a number of iterations. During experiments a moderate computer with a Intel Core 2 DUO 2.66GHz processor, a Crucial MX100 256GB SSD and 4GB DDR3 memory is used.

For each experiment, various parameters need to be set. As indicated in the introduction of this chapter, in literature extensive experiments are performed on parameter settings. Some of these parameters let MMAS perform better than others. In fact, the parameters need to be optimized for every new TSP instance. However, the purpose of this research is not to repeat extensive experiments with parameter settings. Therefore, the guidelines provided by literature will be used. A brief summary of the recommendations concerning parameter settings is given in subsection 4.8.2. This forms the basis of the parameters used in the experiments. For every experiment, a list with the parameter settings used is provided.

6.2. VERIFICATION

Verification of the implemented MMAS algorithm consists of two parts. The first part was performed during implementation of the algorithm. As described trough chapter 4 and chapter 5, MMAS consists of various components with their own mathematical relations and principles. The MMAS algorithm was built step by step. The function of every added component is checked for correct (corresponds to function as intended in literature) implementation by comparing results of the algorithm with hand calculations. As the optimal solutions are known for the TSP instances from TSB LIB, it provides another form of verification. Above that, a second part of verification is performed by means of a comparative experiment with the complete algorithm.

For further verification an experiment performed in literature (see Stützle and Hoos [1997]) is repeated. The same settings are used and are listed in Table 6.1. Only the number of iterations and the number of runs performed is reduced. For eil51 the number of iterations is reduced from 10 000 to 51 and the number of runs is reduced from 25 to 10. For kroA100 the number of iterations is reduced from 10 000 to 10 and the number of runs is reduced from 25 to 10. The reason why the number of iterations and runs is reduced, is because of the run time. If 10 000 iterations would be performed with our implementation, it would result in extreme long run times. The same reasoning holds for the size of the TSP instance, since comparison could be made for larger TSP instances (over 1 000 nodes). For every run an other random seed is chosen by turning off the predictability of the random generators. Above that, pheromone trail update is set to dynamical and local search is turned on.

The results of the experiment are shown in Table 6.2. The percentage between the parenthesis indicates the deviation from optimal tour length given by TSP LIB. If the results from our implementation are compared with the results from literature, one can see that the results from literature are better in terms of solution quality. Which is not surprising, since our implementation performed a fraction of the number of iterations. However, our implementation approaches the solution quality from literature quite well. The necessity to decrease the number of iterations and runs indicates that the code is better optimized in literature than for our implementation.

Table 6.1: Settings experiment

Number of ants	$n/2$
α	1
β	2
ρ	0.96
p_{best}	0.05
Nearest neighbours	20
Search width	20

Table 6.2: Results from literature (left) and implemented MMAS algorithm (right), average over 5 runs

Instance	Best solution literature	Avg. solution literature	σ lit.	Best solution MMAS	Avg. solution MMAS	σ MMAS	Avg. run time [hh:mm:ss]
eil51	426 (0.00%)	427 (0.16%)	0.70	430 (0.94%)	434 (1.81%)	2.10	0:02:16
kroA100	212×10^2 (0.00%)	213×10^2 (0.01%)	16.4	216×10^2 (1.29%)	221×10^2 (3.93%)	260	0:07:53

6.3. TOUR CONSTRUCTIONS

One tour construction equals one ant that has completed a route. So the number of tour constructions performed during a run is the product of the number of ants and the number of iterations. Given a certain number of tour constructions, how should these tour constructions be distributed over the number of ants

and the number of iterations? The objective of this experiment is to investigate the influence of this distribution. If more information is known about the influence of this distribution, this can be kept in mind while parameters for a run of the algorithm are set in general. During experiments the solution quality and speed is monitored. A fixed number of 750 tour constructions is used for two TSP instances: eil51 and pr76. For every run an other random seed is chosen by turning off the predictability of the random generators. Above that, pheromone trail update is set to dynamical and local search is turned on. All the other settings are listed in Table 6.3.

Table 6.3: Settings experiment

α	1
β	3
ρ	0.85
p_{best}	0.05
Nearest neighbours	20
Search width	20

The results of the experiment are shown in Table 6.4 and Table 6.5. The percentage between the parenthesis indicates the deviation from optimal tour length given by TSP LIB. The results indicate that the influence of the distribution of the number of tour constructions on the run time to complete 750 tours constructions is significant (reduces with 77% and 82% for eil51 and pr76 respectively), while influence on the solution quality is relative limited. Therefore, the choice for the combination of number of ants and the number of iterations should be made with care. This makes the number of ants a relative important setting. As stated in subsection 4.8.2 it is recommended to use a number of ants m equal to the number of nodes n. This recommendation is supported by these measurements. The results from Table 6.4 and Table 6.5 are graphically represented in Figure 6.1 and Figure 6.2 respectively.

Table 6.4: Results for TSP instance eil51, average over 5 runs

Ants	Iterations	Best sol.	Avg. solution	σ	Avg. run time [hh:mm:ss]
10	75	427 (0.23%)	429 (0.70%)	1.20	0:02:36
15	50	428 (0.47%)	430 (0.85%)	1.50	0:01:51
25	30	428 (0.47%)	431 (1.22%)	2.60	0:01:14
30	25	428 (0.47%)	430 (0.99%)	1.50	0:01:04
50	15	433 (1.64%)	435 (2.21%)	1.70	0:00:46
75	10	429 (0.70%)	436 (2.35%)	5.40	0:00:36

Table 6.5: Results for TSP instance pr76, average over 5 runs

Ants	Iterations	Best sol.	Avg. solution	σ	Avg. run time [hh:mm:ss]
10	75	109×10^3 (0.59%)	110×10^3 (1.34%)	730	0:18:51
15	50	109×10^3 (1.06%)	110×10^3 (1.36%)	310	0:12:09
25	30	109×10^3 (0.88%)	110×10^3 (1.58%)	905	0:08:06
30	25	109×10^3 (0.86%)	110×10^3 (1.49%)	682	0:06:23
50	15	109×10^3 (1.24%)	111×10^3 (2.20%)	631	0:04:31
75	10	109×10^3 (0.95%)	110×10^3 (1.92%)	809	0:03:24

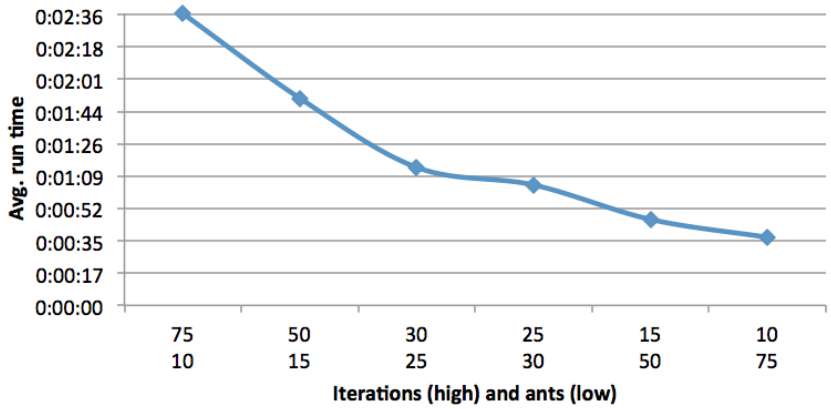

Figure 6.1: Graph presenting average run time for TSP instance eil51 for different distributions of 750 tour constructions

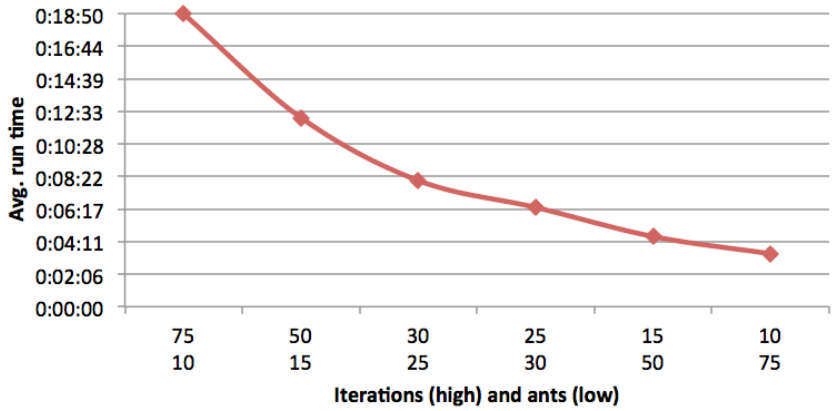

Figure 6.2: Graph presenting average run time for TSP instance pr76 for different distributions of 750 tour constructions

6.4. MMAS VERSUS B&B

In this section various TSP instances will be solved by both the MMAS as the B&B algorithm. The B&B algorithm was not able to solve TSP instances with sizes of $n > 50$ within reasonable time limits (hours). Therefore, the comparison is limited to instances with sizes $n < 50$. The MMAS algorithm is terminated when a solution quality within 1% of the optimal solution provided by TSP LIB is found. The objective of this experiment is to give an indication of the performance of the implemented MMAS algorithm with respect to B&B in general, so for the TSP instances which are not included in TSP LIB and for which the optimal solution is often unknown. For every run an other random seed is chosen by turning off the predictability of the random generators. Above that, pheromone trail update is set to dynamical and local search is turned on. All the other settings are listed in Table 6.6. Please note that only 5 nearest neighbours are used, since relative small TSP instances are considered.

During experiments it appeared that the exact B&B algorithm and the MMAS algorithm end up with the same optimal solution as provided by TSP LIB. This provides another form of verification. The results of the experiment are shown in Table 6.7. One could see that for the smaller instances, both algorithms are competitive regarding solving speed. However, as problem size increases, the strength of the heuristic algorithm becomes visible. In relative small amount of time a near optimal solution is found. The advantage for even larger TSP instances will become clear in section 6.5.

Table 6.6: Settings experiment

Number of ants	n (17 to 48)
α	1
β	3
ρ	0.8
p_{best}	0.05
Nearest neighbours	5
Search width	10

Table 6.7: Performance B&B algorithm (left) compared with MMAS algorithm (right), average over 10 runs

TSP instance	Solution B&B	Run time B&B [hh:mm:ss]	Solution threshold MMAS	Avg. run time MMAS [hh:mm:ss]
gr17	190×10^1	< 00:00:01	$\leq 190\times10^1+1\%$	< 00:00:02
gr24	140×10^1	< 00:00:02	$\leq 140\times10^1+1\%$	< 00:00:02
fri26	936	< 00:00:01	$\leq$ 936+1%	< 00:00:02
swiss42	130×10^1	0:02:14	$\leq 130\times10^1+1\%$	00:00:08
gr48	458×10^1	0:10:42	$\leq 458\times10^1+1\%$	00:00:31

6.5. ITERATION CONSTRAINTS

6.5.1. VARIABLE CONSTRAINT

In this subsection one TSP instance (berlin52) will be solved with various restrictions on the number of iterations. The objective of this experiment is to give an indication of the development in accuracy (in terms of solution quality) for the implemented MMAS algorithm for different limitations on the number of iterations. It is assumed that TSP instance berlin52 is representative for a general conclusion in view of the objective of this experiment. For every run an other random seed is chosen by turning off the predictability of the random generators. Above that, pheromone trail update is set to dynamical and local search is turned on. All the other settings are listed in Table 6.8.

The results of the experiment are shown in Table 6.9. The percentage between the parenthesis indicates the deviation from optimal tour length given by TSP LIB. One could see that for an increasing number of iterations, the accuracy increases. Above that, within five iterations or less, the implemented MMAS algorithm was not able to find the optimal solution within 10 runs. In addition to Table 6.9, the accuracy for increasing number of iterations is shown in Figure 6.3. The figure shows that the improvement in accuracy decreases for an increasing number of iterations. The red line indicates the optimal solution.

Table 6.8: Settings experiment

Number of ants	52
α	1
β	3
ρ	0.8
p_{best}	0.05
Nearest neighbours	20
Search width	20

Table 6.9: Results TSP instnace berlin52, average over 10 runs

Iterations	Best sol.	Avg. solution	σ	Avg. run time [hh:mm:ss]
1	782×10^1 (3.65%)	797×10^1 (5.65%)	128	00:00:06
5	767×10^1 (1.70%)	788×10^1 (4.55%)	107	00:00:29
10	754×10^1 (0.00%)	775×10^1 (2.71%)	109	00:00:58
15	754×10^1 (0.00%)	767×10^1 (1.69%)	73.7	00:01:26
20	754×10^1 (0.00%)	765×10^1 (1.47%)	83.1	00:01:49

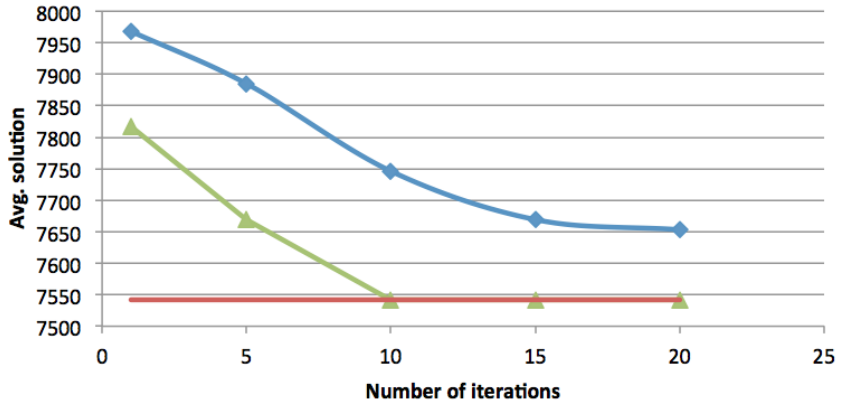

Figure 6.3: Graph presenting average solutions (blue) and best solutions (green) from Table 6.9 and the optimal solution (red)

6.5.2. FIXED CONSTRAINT

In this subsection various larger ($n > 50$) TSP instances will be solved with a restriction on the number of iterations. The algorithm is allowed to perform only 2 iterations. The objective of this experiment is to give an indication of the performance (both in solution quality as in required run time) of the implemented MMAS algorithm during a limited number of iterations. It is assumed that the TSP instances used are representative for a general conclusion in view of the objective of this experiment. For every run an other random seed is chosen by turning off the predictability of the random generators. Above that, pheromone trail update is set to dynamical and local search is turned on. All the other settings are listed in Table 6.10.

The results of the experiment are shown in Table 6.11. The percentage between the parenthesis indicates the deviation from optimal tour length given by TSP LIB. One could see that within this restricted run time, the implemented MMAS algorithm provides a reasonable (within 9% of the optimal solution) average solution quality. An other interesting thing to notice is that the average solution quality for example the TSP instance pr76 is better than for berlin52. Since the number of iterations performed is the same, one could think of a difference in solution quality due to the structure of the map. In addition to Table 6.11, the average run time for increasing TSP instance size is shown in Figure 6.4. This figure gives an indication of the development of the run time for increasing TSP instance size of the implemented MMAS algorithm.

Table 6.10: Settings experiment

Number of ants	n (52 to 150)
α	1
β	3
ρ	0.8
p_{best}	0.05
Nearest neighbours	20
Search width	20

Table 6.11: Results under time constraint, average over 10 runs

Instance	Iterations	Best sol.	Avg. solution	σ	Avg. run time [hh:mm:ss]
berlin52	2	780×10^1 (3.43%)	802×10^1 (6.33%)	152	00:00:10
pr76	2	110×10^3 (1.86%)	112×10^3 (3.13%)	102×10^1	00:00:42
kroA100	2	218×10^2 (2.48%)	227×10^2 (6.62%)	539	00:01:40
bier127	2	122×10^3 (3.17%)	124×10^3 (5.05%)	198×10^1	00:05:13
ch150	2	696×10^2 (6.56%)	706×10^2 (8.16%)	86.2	00:11:46

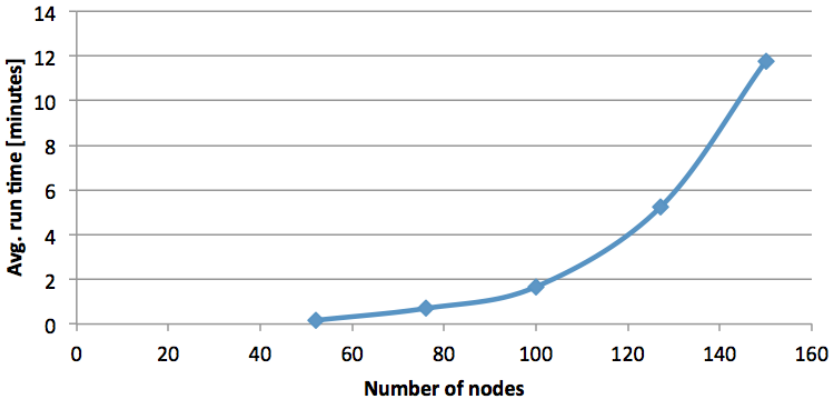

Figure 6.4: Graph presenting average run time from Table 6.11

7

CONCLUSIONS AND RECOMMENDATIONS

7.1. CONCLUSIONS

The performance of MMAS (ACO algorithm) and B&B (exact solver) is compared by implementing these algorithms in Delphi 2010. Both models are verified and validated. With both models exact the same TSP instances are solved. TSP instances are used for which the optimal solution is known. The B&B algorithm stopped when the optimal solution was found. The MMAS algorithm stopped when a solution within 1% of the optimal solution was found. The run time required for completing these operations is measured.

With MMAS qualitative good solutions can be obtained in solving TSP instances. The advantage of the heuristic MMAS approach with respect to the exact solver B&B starts to pay off for TSP instances of around 40 nodes and larger. A good solution (within 1% of the optimal solution) can be found with MMAS within a small amount of time compared to the run time required by B&B. The run time required by MMAS increases exponentially with the number of nodes of the TSP instance (see Figure 6.4). The worst-case time complexity of the B&B algorithm could be as bad as $\mathcal{O}(n!)$. It is shown that the computational time required by B&B increases so fast with increasing TSP instance size, that for larger TSP instances this exact algorithm is not suitable anymore. MMAS thus provides a necessary and good alternative.

The accuracy (in terms of solution quality) of the implemented MMAS algorithm increases with the number of iterations allowed. In the region of with a low number of iterations allowed, the accuracy increases fast. The increment in accuracy decreases as the number of iterations allowed increases (see Figure 6.3). If the MMAS algorithm is performed under an iteration constraint and is only allowed to perform two iterations, an average solution quality of within 9% of optimality is found for TSP instances up to 150 nodes. The number of iterations performed and TSP instances attacked in literature concerning MMAS give a strong indication that the code is better optimized in literature than for our implementation of MMAS.

The performance of MMAS algorithm relies on conventional algorithms like nearest neighbour and local search. The implementation of local search significantly improves the performance of MMAS. Without these algorithms, MMAS cannot compete with other specialized algorithms for solving TSP instances. If MMAS is compared with the most efficient heuristics to solve TPS instances, there is still a considerable gap to close [Stützle and Hoos, 1996].

The Travelling Salesman Problem is a classical problem in combinatorial optimization problems. The TSP has many applications within logistic systems. However, the simplicity of formulation of the problem is deceitful. Globally, algorithms to solve the TSP can be divided in exact algorithms and approximate algorithms. Little's B&B is such an exact algorithm. The idea of this algorithm is to break up all feasible routes into increasing smaller subsets by a procedure called branching. For each subset, a lower bound on the cost (length) of the included routes is calculated. These bounds guide the branching of the subsets. Eventually, a subset is found that contains a single route whose length is less than or equal to the lower bound of all other subsets. That route is optimal.

ACO is a heuristic (approximate) algorithm. Many different ACO algorithms exist, but the underlying thought is the foraging behaviour of ants. During foraging, the ants communicate to each other via depositing pheromone. When an ant locates a food source, it carries the food to the nest and deposits a pheromone trail. The path of the other foraging ants is influenced by this pheromone trail. Stronger pheromone trails attract more ants and shorter paths accumulate faster in pheromone intensity. This positive feedback mechanism biases the ants to shorter paths. Eventually, all the ants follow the shortest path to the food source. For ACO features are modified or added to create artificial ants. Above that, further improvements are realized by incorporating other heuristics, like local search and nearest neighbour procedures.

7.2. RECOMMENDATIONS

As stated in section 1.2, the main objective of this research is obtaining knowledge and experience in the field of Ant Colony Optimization. For further extension of this knowledge and experience recommendations are formulated. Regarding the implemented MMAS algorithm:

- In the implemented MMAS algorithm, the parameters are set once. In literature concerning MMAS the dynamic adjustment of the most important parameters (like ρ and m) is suggested. By adjusting these parameters, the speed of convergence and the obtainable solution quality can be influenced [Stützle and Hoos, 1996]. This extension provides an interesting way to continue research on MMAS.
- As stated in section 5.1, during implementation the route of an ant is forced to a Hamiltonian cycle, as prescribed by literature concerning MMAS. Therefore, if the ant returns to the home node, no loops are included in the route. However, one could also think of not forcing the route to a Hamiltonian cycle. The loops then have to be eliminated after the ant returned to the home node. It is recommended to investigate the advantages and disadvantages of both approaches.
- During literature review one TSP solver was encountered many times, called Concorde [Waterloo, 2011]. It is praised for it's performance. The library of Concorde contains over 700 functions to solve TSP instances. It may pay off to take a closer look at this computer program. For example the incorporated Lin-Kernighan heuristic to improve the local search phase of MMAS.
- In section 3.4 a deliberately choice for MMAS algorithm is given. However, this algorithm originates from the first developments in ACO algorithms (mid 90s). Since then a tremendous number of different ACO algorithms are developed. To further extend the knowledge and experience in the field of ACO algorithms, a more recent algorithm may be considered.

Regarding the implemented research model:

- At the beginning of the research, there is searched for a "graph package" to extend to possibilities of Delphi 2010. This graph package should give additional possibilities to set up and work with a graph. However, such a package was not found. During implementation of the model, it is encountered that setting up a graph by using lists of node and edge IDs is sufficient for a basic setup of an ACO algorithm. However, as need for increasing complex operations increases, the lack of such a graph package becomes visible. At first sight simple operations do not only become very devious and hard to program, they also become computationally expensive. For example, the implementation of local search (see section 5.5). After two edges are deleted, there is only one way to create a new route. To reconnect the two parts, the two correct pairs of nodes/edges must be identified. To do this, the orientation of an edge in the route is required. Subsequently, if an improvement is detected, the list of visited edges must be updated. The right edge must be replaced with the right edge and the visited edges in between these two edges are now visited in opposite direction and thus must be mirrored. As can be seen, this forms a cumbersome structure. Therefore it is recommended to pay extra attention to setting up a graph.
- In this research only symmetric TSP instance are considered. However, in reality this is of course not always the case. Therefore, asymmetric TSP instances may be considered. Solving asymmetric TSP instances entails more difficulties. For example in adding local search, a 2-opt exchange is not directly applicable since the direction in which the nodes are visited does matter now.
- During experiments more emphasis was on performance with respect to time than the experiments performed in literature. The presented run times are real wall-clock times. It is recommended to use CPU time instead to benchmark an algorithm. CPU time measures only the time while the processor is actively performing calculations for the program. Wall-clock time measures the total time to complete the process and is for example dependent on the time for resources to become available.
- As stated in section 5.9, there is no possibility for animating the movement of the ants. For illustrative purposes one could implement this animation. One could also think of adjusting the thickness of the drawn edges dependent on their pheromone trail intensity.
- In section 2.4 an introduction to how OpenStreetMap could be used to represent TSP instances is given. As many applications of the TSP are physical transport problems with cities and roads involved, it's recommended to further investigate the possibilities to make use of this promising source of information.

BIBLIOGRAPHY

Chang, X., X. Jun, and C. Huiyou (2009, August). Ant Colony Optimization Based on Estimation of Distribution for the Traveling Salesman Problem. In *Fifth International Conference on Natural Computation*, pp. 19–23. IEEE.

Deneubourg, J. L., S. Aron, S. Goss, and J. M. Pasteels (1990). The self-organizing exploratory pattern of the Argentine ant. *Journal of insect behavior 3*(2), 159–168.

Dorigo, M. (2006, November). Ant colony optimization. *Computational Intelligence Magazine 1*(4), 28–39.

Dorigo, M. and L. M. Gambardella (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 53–66.

Dorigo, M., V. Maniezzo, and A. Colorni (1991, June). Positive feedback as a search strategy. Technical report, Politecnico di Milano, Italy. Report number: 91-016.

Dorigo, M., V. Maniezzo, and A. Colorni (1996, February). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics 26*(1), 29–41.

Engelbrecht, A. P. (2005). *Fundamentals of computational Swarm Intelligence*. Wiley.

Gambardella, L. M. and M. Dorigo (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. In *ICML*, pp. 252–260.

Gambardella, L. M. and M. Dorigo (2000). An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing 12*(3), 237–255.

Goss, S., S. Aron, J. L. Deneubourg, and J. M. Pasteels (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften 76*(12), 579–581.

Guo, J. (2006, October). An Ant Colony Optimization Algorithm with Evolutionary Operator for Traveling Salesman Problem. In *Sixth International Conference on Intelligent Systems Design and Applications*, Volume 1, pp. 385–389. IEEE.

Hong, Z. and F. Bian (2008, December). Novel Ant Colony Optimization for Solving Traveling Salesman Problem in Congested Transportation System. In *Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, pp. 122–125. IEEE.

Laporte, G. (1992). The Traveling Salesman Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research 59*(2), 231–247.

Lin, S. and B. W. Kernighan (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research 21*(2), 498–516.

Little, J. D. C., K. G. Murty, D. W. Sweeney, and C. Karel (1963). An algorithm for the traveling salesman problem. *Operations research 11*(6), 972–989.

OpenStreetMap (2014a). OpenStreetMap Wiki. `http://www.osmfoundation.org/wiki/Main_Page`. View date: 31/01/15.

OpenStreetMap (2014b). OpenStreetMap Wiki. `http://wiki.openstreetmap.org/wiki/Elements`. View date: 31/01/15.

OpenStreetMap (2014c). OpenStreetMap Wiki. `http://wiki.openstreetmap.org/wiki/Downloading_data`. View date: 31/01/15.

QGIS (2014). Qgis. http://planet.qgis.org/planet/tag/routing/. View date: 31/01/15.

Reinelt, G. (1994). *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verslag.

Reinelt, G. (1995). T S P Library. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. View date: 15/11/14.

Sanchez, G. (2013, January). 7 Functions to do Metric Multidimensional Scaling in R. http://gastonsanchez.com/blog/how-to/2013/01/23/MDS-in-R.html. View date: 26/02/15.

Shi, X., L. Wang, Y. Zhou, and Y. Liang (2008, October). An Ant Colony Optimization Method for Prize-collecting Traveling Salesman Problem with Time Windows. In *Fourth International Conference on Natural Computation*, Volume 7, pp. 480–484.

Shokouhifar, M. and S. Sabet (2012, July). PMACO: A pheromone-mutation based ant colony optimization for traveling salesman problem. In *International Symposium on Innovations in Intelligent Systems and Applications*, pp. 1–5.

Socha, K., M. Sampels, and M. Manfrin. *Ant algorithms for the university course timetabling problem with regard to the state-of-the-art*, pp. 334–345. Springer.

Solnon, C. (2010). *Ant Colony Optimization and Constraint Programming*. John Wiley & Sons.

Stony-Brook-University (-). Branch and bound Algorithm for Travelling Salesman Problem. http://www3.cs.stonybrook.edu/~{}algorith/implement/syslo/distrib/processed/babtsp.p.

Stützle, T. and H. Hoos (1996). Improving the Ant System: A detailed report on the Max-Min Ant System.

Stützle, T. and H. Hoos (1997). Max-Min Ant System and Local Search for the Traveling Salesman Problem. In *International Conference on Evolutionary Computation*, pp. 309–314. IEEE.

Stützle, T. and H. Hoos (2000). Max-Min Ant System. *Future generation computer systems 16*(8), 889–914.

Veeke, H. P. M. and J. A. Ottjes (2010, September). Tomas. http://www.tomasweb.com. View date: 15/11/14.

Waterloo, U. o. (2011, December). Concorde. http://www.math.uwaterloo.ca/tsp/concorde/index.html. View date: 31/01/15.

Wei, L. and Z. Yuren (2010, May). An Effective Hybrid Ant Colony Algorithm for Solving the Traveling Salesman Problem. In *International Conference on Intelligent Computation Technology and Automation*, pp. 497–500.

Zeng, D., Q. He, B. Leng, W. Zheng, H. Xu, Y. Wang, and G. Guan (2012, December). An improved ant colony optimization algorithm based on dynamically adjusting ant number. In *International Conference on Robotics and Biomimetics*, pp. 2039–2043. IEEE.

Zhang, Y. and L. Li (2008, October). Back to Results MST Ant Colony Optimization with Lin-Kerninghan Local Search for the Traveling Salesman Problem. In *International Symposium on Computational Intelligence and Design*, pp. 344–347. IEEE.

Zhang, Y., Z. Pei, J. Yang, and Y. Liang (2007, October). An Improved Ant Colony Optimization Algorithm Based on Route Optimization and Its Applications in Travelling Salesman Problem. In *7th International Conference on Bioinformatics and Bioengineering*, pp. 693–698. IEEE.

Zhang, Z. and Z. Feng (2009, November). A novel Max-Min ant system algorithm for traveling salesman problem. In *International Conference on Intelligent Computing and Intelligent Systems*. IEEE.

Zhao, F., J. Dong, S. Li, and J. Sun (2008, June). An improved ant colony optimization algorithm with embedded genetic algorithm for the traveling salesman problem. In *7th World Congress on Intelligent Control and Automation*, pp. 7902–7906.

Zhou, X., L. Zhao, Z. Xia, Z. Chen, and R. Wang (2012, May). An ant system with two colonies and its application to Traveling Salesman Problem. In *Eighth International Conference on Natural Computation*, pp. 744–748.

Zhou, X., L. Zhao, Z. Xia, and R. Wang (2012, October). A Max-Min Ant System with two colonies and its application to Traveling Salesman Problem. In *Fifth International Conference on Advanced Computational Intelligence*, pp. 319–323. IEEE.

APPENDIX A

Name	#cities	Type	Bounds
a280	280	EUC_2D	2579
ali535	535	GEO	202310
att48	48	ATT	10628
att532	532	ATT	27686
bayg29	29	GEO	1610
bays29	29	GEO	2020
berlin52	52	EUC_2D	7542
bier127	127	EUC_2D	118282
brazil58	58	MATRIX	25395
brd14051	14051	EUC_2D	[468942,469445]
brg180	180	MATRIX	1950
burma14	14	GEO	3323
ch130	130	EUC_2D	6110
ch150	150	EUC_2D	6528
d198	198	EUC_2D	15780
d493	493	EUC_2D	35002
d657	657	EUC_2D	48912
d1291	1291	EUC_2D	50801
d1655	1655	EUC_2D	62128
d2103	2103	EUC_2D	[79952,80450]
d15112	15112	EUC_2D	[1564590,1573152]
d18512	18512	EUC_2D	[644650,645488]
dantzig42	42	MATRIX	699
dsj1000	1000	CEIL_2D	18659688
eil51	51	EUC_2D	426
eil76	76	EUC_2D	538
eil101	101	EUC_2D	629

Figure 1: Symmetric travelling salesman problems (part 1) [Reinelt, 1995]

Name	#cities	Type	Bounds
fl417	417	EUC_2D	**11861**
fl1400	1400	EUC_2D	**20127**
fl1577	1577	EUC_2D	[22204,22249]
fl3795	3795	EUC_2D	[28723,28772]
fnl4461	4461	EUC_2D	**182566**
fri26	26	MATRIX	**937**
gil262	262	EUC_2D	**2378**
gr17	17	MATRIX	**2085**
gr21	21	MATRIX	**2707**
gr24	24	MATRIX	**1272**
gr48	48	MATRIX	**5046**
gr96	96	GEO	**55209**
gr120	120	MATRIX	**6942**
gr137	137	GEO	**69853**
gr202	202	GEO	**40160**
gr229	229	GEO	**134602**
gr431	431	GEO	**171414**
gr666	666	GEO	**294358**
hk48	48	MATRIX	**11461**
kroA100	100	EUC_2D	**21282**
kroB100	100	EUC_2D	**22141**
kroC100	100	EUC_2D	**20749**
kroD100	100	EUC_2D	**21294**
kroE100	100	EUC_2D	**22068**
kroA150	150	EUC_2D	**26524**
kroB150	150	EUC_2D	**26130**
kroA200	200	EUC_2D	**29368**
kroB200	200	EUC_2D	**29437**
lin105	105	EUC_2D	**14379**
lin318	318	EUC_2D	**42029**
linhp318	318	EUC_2D	**41345**
nrw1379	1379	EUC_2D	**56638**
p654	654	EUC_2D	**34643**
pa561	561	MATRIX	**2763**
pcb442	442	EUC_2D	**50778**
pcb1173	1173	EUC_2D	**56892**
pcb3038	3038	EUC_2D	**137694**
pla7397	7397	CEIL_2D	**23260728**
pla33810	33810	CEIL_2D	[65913275,66116530]
pla85900	85900	CEIL_2D	[141904862,142487006]

Figure 2: Symmetric travelling salesman problems (part 2) [Reinelt, 1995]

Name	#cities	Type	Bounds
pr76	76	EUC_2D	108159
pr107	107	EUC_2D	44303
pr124	124	EUC_2D	59030
pr136	136	EUC_2D	96772
pr144	144	EUC_2D	58537
pr152	152	EUC_2D	73682
pr226	226	EUC_2D	80369
pr264	264	EUC_2D	49135
pr299	299	EUC_2D	48191
pr439	439	EUC_2D	107217
pr1002	1002	EUC_2D	259045
pr2392	2392	EUC_2D	378032
rat99	99	EUC_2D	1211
rat195	195	EUC_2D	2323
rat575	575	EUC_2D	6773
rat783	783	EUC_2D	8806
rd100	100	EUC_2D	7910
rd400	400	EUC_2D	15281
rl1304	1304	EUC_2D	252948
rl1323	1323	EUC_2D	270199
rl1889	1889	EUC_2D	316536
rl5915	5915	EUC_2D	[565040,565530]
rl5934	5934	EUC_2D	[554070,556045]
rl11849	11849	EUC_2D	[920847,923368]
si175	175	MATRIX	21407
si535	535	MATRIX	48450
si1032	1032	MATRIX	92650
st70	70	EUC_2D	675
swiss42	42	MATRIX	1273
ts225	225	EUC_2D	126643
tsp225	225	EUC_2D	3919
u159	159	EUC_2D	42080
u574	574	EUC_2D	36905
u724	724	EUC_2D	41910
u1060	1060	EUC_2D	224094
u1432	1432	EUC_2D	152970
u1817	1817	EUC_2D	57201
u2152	2152	EUC_2D	64253
u2319	2319	EUC_2D	234256
ulysses16	16	GEO	6859
ulysses22	22	GEO	7013
usa13509	13509	EUC_2D	[19947008,19982889]
vm1084	1084	EUC_2D	239297
vm1748	1748	EUC_2D	336556

Figure 3: Symmetric travelling salesman problems (part 3) [Reinelt, 1995]

APPENDIX B

```
NAME: gr17
TYPE: TSP
COMMENT: 17-city problem (Groetschel)
DIMENSION: 17
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: LOWER_DIAG_ROW
EDGE_WEIGHT_SECTION
0
633 0
257 390 0
91 661 228 0
412 227 169 383 0
150 488 112 120 267 0
80 572 196 77 351 63 0
134 530 154 105 309 34 29 0
259 555 372 175 338 264 232 249 0
505 289 262 476 196 360 444 402 495 0
353 282 110 324 61 208 292 250 352 154 0
324 638 437 240 421 329 297 314 95 578 435 0
70 567 191 27 346 83 47 68 189 439 287 254 0
211 466 74 182 243 105 150 108 326 336 184 391 145 0
268 420 53 239 199 123 207 165 383 240 140 448 202 57 0
246 745 472 237 528 364 332 349 202 685 542 157 289 426 483 0
121 518 142 84 297 35 29 36 236 390 238 301 55 96 153 336 0
EOF
```

Figure 4: Example TSP file with explicit cost matrix

```
NAME: berlin52
TYPE: TSP
COMMENT: 52 locations in Berlin (Groetschel)
DIMENSION: 52
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 565.0 575.0
2 25.0 185.0
3 345.0 750.0
4 945.0 685.0
5 845.0 655.0
6 880.0 660.0
7 25.0 230.0
8 525.0 1000.0
9 580.0 1175.0
10 650.0 1130.0
11 1605.0 620.0
12 1220.0 580.0
13 1465.0 200.0
14 1530.0 5.0
15 845.0 680.0
16 725.0 370.0
17 145.0 665.0
18 415.0 635.0
19 510.0 875.0
20 560.0 365.0
21 300.0 465.0
22 520.0 585.0
23 480.0 415.0
24 835.0 625.0
25 975.0 580.0
26 1215.0 245.0
27 1320.0 315.0
28 1250.0 400.0
29 660.0 180.0
30 410.0 250.0
31 420.0 555.0
32 575.0 665.0
33 1150.0 1160.0
34 700.0 580.0
35 685.0 595.0
36 685.0 610.0
37 770.0 610.0
38 795.0 645.0
39 720.0 635.0
40 760.0 650.0
41 475.0 960.0
42 95.0 260.0
43 875.0 920.0
44 700.0 500.0
45 555.0 815.0
46 830.0 485.0
47 1170.0 65.0
48 830.0 610.0
49 605.0 625.0
50 595.0 360.0
51 1340.0 725.0
52 1740.0 245.0
EOF
```

Figure 5: Example TSP file with implicit cost matrix

APPENDIX C

As indicated in section 5.8, it is chosen to convert the smaller TSP instances with explicit edge weight matrices to a 2D coordinates list. This conversion is done by means of an approximation algorithm called multidimensional scaling (MDS). In this appendix an introduction to MDS will be given.

MDS is a collective noun for multi variable data analysis methods that are used to analyse similarities or dissimilarities in a data set. MDS visualizes the similarities or dissimilarities in a low-dimensional space and thus enables inspection of underlying meaningful dimensions. MDS can be divided in two main approaches: metric and non-metric. If the analysed matrix contains metric distances, the approach is metric MDS. Otherwise, the approach is non-metric MDS. In view of the usage of MDS in this research, this appendix focuses on metric MDS. The general approach of MDS consists of two steps:

- Calculate a (dis)similarity matrix among pairs of objects (i.e. observations, samples, elements etc.)
- Apply one of the MDS models to obtain a low-dimensional representation.

Dissimilarity is defined as the distance between two elements under a certain criterion. In other words: how different are these elements? So for example, the Euclidean distance between two nodes is a measure of their dissimilarity. A (dis)similarity matrix is a matrix that contains the (dis)similarity between every pair of elements. The matrix is square and symmetric. The diagonal entries are defined as zero, meaning that zero is the measure of dissimilarity between an element and itself.

Remember that an explicit edge weight matrix is given and that a 2D coordinates list is desired. The next step is that metric MDS (also known as principal coordinate analysis) transforms the distance matrix into a set of coordinates such that the (Euclidean) distances derived from these coordinates approximate as well as possible the original distances from the distance matrix [Sanchez, 2013]. This conversion makes use of the (dis)similarity matrix. It is important to note that the orientation of the coordinates in the output file is arbitrary. This is because the created map with nodes can be rotated, while the distances between the nodes are maintained. Thus the orientation of the coordinates in the output file is arbitrary. Above that, it is emphasized that MDS is an approximation technique and thus does not provide exact results. Therefore, the optimal solutions found by the B&B and MMAS algorithm for the converted TSP instances deviate from the optimal solutions provided by TSP LIB. However, this deviation is limited (order of a few percent for the converted TSP instances). Since both algorithms face the same TSP instance, there is no mutual difference.

MDS is implemented as a standard function in many statistical computer programs. For this research use is made of the computer program R and the standard function *cmdscale*. For further information on this function the reader is referred to the manual of R. The result of the conversion for TSP instance swiss42 is shown in Figure 6.

Figure 6: Result of MDS for TSP instance swiss42