universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

„Ant approaches for the QAP“

Verfasserin

Stephanie Richter

angestrebter akademischer Grad

Magistra der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec.)

Wien, im März 2011

Studienkennzahl lt. Studienblatt:	157
Studienrichtung lt. Studienblatt:	Internationale Betriebswirtschaft
Betreuer/Betreuerin:	o. Univ.-Prof. Dipl.-Ing. Dr. Richard F. Hartl

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit allein und nur unter Verwendung der angeführten Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

_____	_____
Ort und Datum	Stephanie Richter

Danksagung

Allen voran will ich mich bei meinen Eltern bedanken, welche mir mein bisheriges Leben lang immer mit Rat und Tat zur Seite standen und mir dieses Studium durch ihre finanzielle Unterstützung ermöglichten. Danke für alles – ich hätte oft nicht gewusst, was ich ohne euch tun sollte!!

Ein weiteres Dankeschön gilt meiner Familie – ihr habt mich immer tatkräftig unterstützt und mich dazu ermuntert nicht den Kopf hängen zu lassen in Zeiten, in denen ich selbst nicht mehr an meinen Erfolg geglaubt habe.
Besonders will ich hier meine Nichte Helene und meinen Neffen Alexander erwähnen, die mir beide sehr viel Freude bereiten.

Desweiteren möchte ich all meinen Freunden und Unikolleginnen für ihre Unterstützung und die vielen lustigen Stunden in unseren Lernkreisen danken – ohne euch hätte mir sicher oft die Motivation gefehlt.

Ganz besonders bedanken möchte ich mich bei meiner besten Freundin Ingrid Oberleitner – ich kann das alles hier gar nicht in Worte fassen – du weißt schon was ich meine!!!

Diese Diplomarbeit widme ich meinem verstorbenen Großvater, welcher mir mein Leben lang immer ein großes Vorbild war.

Table of Contents

1. Executive Summary 1

2. The Quadratic Assignment Problem (QAP) 2

   2.1 General Description 2

   2.2 Mathematical Model 3

   2.3 Example 4

   2.4 The QAPLIB 6

3. Ant Colony Optimization 7

   3.1 Definition of Metaheuristics 7

   3.2 Biological Principles 8

      3.2.1 Real Ants´ Behavior 8

      3.2.2 The Double Bridge Experiment 9

      3.2.3 From Real to Artificial Ants 11

   3.3 Main procedures of the ACO Metaheuristic 11

4. Ant Algorithms 13

   4.1 Historical Development 13

   4.2 Applications to several problems 13

   4.3 Ant System applied to the Traveling Salesman Problem 14

   4.4 The direct successors of Ant System 16

      4.4.1 Elitist Ant System 16

      4.4.2 Rank-Based Ant System 17

      4.4.3 Ant Colony System 17

   4.5 ACO applied to the QAP 19

      4.5.1 Ant System for the QAP 19

      4.5.2 HAS-QAP 21

      4.5.3 MAX-MIN Ant System (MMAS) 26

5. Local Search Methods 29
5.1 2-opt 30
5.1.1 A short example: 2-opt for the QAP 31
5.2 Iterated ants – a hybridization of ACO 32
5.3 Very Large Scale Neighborhood Search (VLSN) 34
5.3.1 Large Neighborhood Search (LNS) 36
5.3.2 Adaptive Large Neighborhood Search (ALNS) 37
6. Implementation 39
6.1 General Principles 39
6.1.1 Roulette Wheel Procedure 40
6.1.2 2-opt First Improvement 41
6.1.3 Updating the Pheromone Trails 41
6.2 The Algorithms 42
6.2.1 MMAS Basis Algorithm 43
6.2.2 MMAS Random Removal 43
6.2.3 MMAS Product Removal Highest 45
6.2.4 MMAS Product Removal Lowest 46
6.2.5 MMAS 3 Iterated 47
7. Computational Results 48
8. Conclusion 54
References 56
Abstract 60
Zusammenfassung 61
Curriculum Vitae 62

List of Figures

Figure 1: Branches of equal length 9
Figure 2: Branches have different length 10
Figure 3: Pseudo-code of an ACO metaheuristic....12
Figure 4: List of several applications of ant algorithms 14
Figure 5: A 40 node TSP (a) and a whole tour as a possible solution (b) 15
Figure 6: The HAS-QAP algorithm....23
Figure 7: Example for intensification....25
Figure 8: 2-opt procedure for the TSP 30
Figure 9: Pairwise interchange (a), adjacent pairwise interchange (b)....31
Figure 10: Outline of an IG algorithm....32
Figure 11: Gradual extension of the neighborhood in VDNS 35
Figure 12: Pseudo-code of an LNS algorithm....37
Figure 13: Pseudo-code of an ALNS algorithm....38
Figure 14: Roulette Wheel procedure 40
Figure 15: Pseudo-code for the MMAS Basis algorithm 43
Figure 16: Pseudo-code for the MMAS Random Removal 44
Figure 17: Pseudo-code for the MMAS Product Removal Highest 46
Figure 18: Destroy/reconstruct of MMAS Product Removal Lowest 47
Figure 19: Pseudo-code for the MMAS 3 Iterated....48

List of Tables

Table 1: Experimental results for tai20a....49
Table 2: Experimental results for tai25a....50
Table 3: Experimental results for tai30a....50
Table 4: Experimental results for tai35a....51
Table 5: Experimental results for tai40a....51
Table 6: Experimental results for tai50a....51
Table 7: Experimental results for scr12....52
Table 8: Experimental results for tho30 52
Table 9: Comparison of percentage deviations....53
Table 10: Comparison of total runtime....53
Table 11: Comparison of time needed to find best solution 54

1. Executive Summary

The objective of this diploma thesis is the enhancement of an MMAS algorithm by ingenious local search procedures and in the following to apply this new algorithm to the Quadratic Assignment Problem (QAP).

As described in chapter 2, the QAP is one of the hardest combinatorial optimization problems and was stated as NP-hard in 1976.[1] Due to the fact that exact algorithms work rather poor for the QAP, the scientific world started with the development of heuristic methods.[2]

In chapter 3 we describe one of these approaches, the so-called Ant Colony Optimization (ACO), in detail and show how the behavior of real ants – especially their foraging procedure based on pheromone trails – influenced the creation of this heuristic.

Chapter 4 is dedicated to the main ACO algorithms. After giving an insight in the historical development, we take a look at the very first ant algorithm, the so-called Ant System. Afterwards we discuss the successors of this algorithm and list the main algorithms for the Traveling Salesman Problem (TSP) as well as for the QAP.

The local search procedures, which are used in our implementation, orientate themselves towards Iterated Ants and Large Neighborhood Search (LNS). In chapter 5 we give a description of these original ideas.

Chapter 6 deals with the implementation part of this diploma thesis. We show all important principles, which form part of the proposed algorithms, and afterwards we take a closer look at the actual algorithms *MMAS Basis*, *MMAS Random Removal*, *MMAS Product Removal Highest*, *MMAS Product Removal Lowest* and *MMAS 3 Iterated*.

Last but not least we present the obtained results which refer to several classical QAP instances taken from the QAPLIB.[3]

[1] see Sahni/Gonzales, 1976
[2] see Ramkumar/Ponnambalam/Jawahar, 2009
[3] http://www.opt.math.tu-graz.ac.at/qaplib/

2. The Quadratic Assignment Problem (QAP)

2.1 General Description

The Quadratic Assignment Problem (QAP) is a typical combinatorial optimization problem which was first introduced by Koopmans and Beckmann in 1957[4].

The main aim of the QAP is the allocation of a set of n facilities (e.g. machines) to a set of n locations (e.g. working stations) in order to minimize the total sum of the products between distances and flows. The distances are measured as the way from one location to another one, the flows are measured as the material flow from one facility to another.

In 1976 Sahni and Gonzales[5] stated the QAP as NP-hard and it is still considered as one of the most difficult combinatorial optimization problems due to the complexity of computing a good solution[6].
Up to now it has just been possible to solve the QAP to optimality in the range of the smaller instance sizes (around n = 25) because the higher the instance size gets, the more intractable it becomes. Unfortunately exact algorithms work rather poor on average and/or need a very long period of time to calculate reasonable solutions. Therefore the development of various heuristics for the QAP took place and led to the possibility to receive relatively satisfying solution values within an acceptable time span. The most important heuristic approaches are ant algorithms, simulated annealing, tabu search, construction methods, genetic algorithms, etc…[7]
The QAP is often used to model real life applications like, for example, the layout planning of university grounds[8], typewriter keyboard design[9] or even hospital layout[10].

[4] see Koopmans/Beckmann, 1957, p 64 ff

[5] see Sahni/Gonzales, 1976

[6] see Ji/Wu/Liu, 2006, p 107

[7] see Ramkumar/Ponnambalam/Jawahar, 2009, p 621

[8] see Dickey/Hopkins, 1972

[9] see Burkard/Offermann, 1977

[10] see Elshafei, 1977

2.2 Mathematical Model[11]

Mathematically the definition of the Quadratic Assignment Problem consists of a set $N = \{1, 2, ..., n\}$ (n locations, n facilities) and two matrices of dimension n x n:

a. Distance matrix $D = \{d_{ij}\}$, where d_{ij} represents the distance between location i and location j
b. Flow matrix $F = \{f_{kl}\}$, where f_{kl} represents the material flow between facility k and facility l

The cost of transferring material, patients, data etc. from location i to location j can easily be calculated by the term:

$$d_{ij} * f_{\pi(i)\pi(j)} \tag{1}$$

After all the main aim of the QAP is to find a permutation out of $S(n)$ which minimizes the total sum of the products between distances and corresponding flows which leads us to the following objective function:

$$\min_{\pi \in S(n)} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} * f_{\pi(i)\pi(j)} \tag{2}$$

The term $\pi_{(i)}$ denotes the facility which is assigned to location i and conversely the term $\pi_{(j)}$ stands for the allocated facility on location j.

There are also two important constraints which need to be taken into consideration:

$$\sum_{k=1}^{n} x_{ik} = 1 \qquad for\ i = 1, \dots, n \tag{3}$$

...each location i can be occupied by exactly 1 facility

[11] see Maniezzo/Colorni/Dorigo, 1994, p 1

$$\sum_{i=1}^{n} x_{ik} = 1 \qquad for\ k = 1, \dots, n \tag{4}$$

...each facility *k* has to be assigned to exactly 1 location *i*

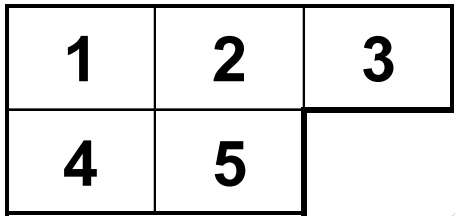
As a binary variable x_{ik} can either be of the value 1 (if facility *k* is assigned to location *j*) or of the value 0 (if facility *k* is not assigned to location *j*).

By taking this binary variable into account, the resulting objective function can be formulated as:

$$\min \sum_{k=1}^{n} \sum_{l=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} * f_{kl} * x_{ik} * x_{jl} \tag{5}$$

2.3 Example

For better understanding a short example of a symmetric QAP with problem size n = 5 is illustrated below.



.... map of locations 1-5

D = {1, 2, 3, 4, 5} F = {A, B, C, D, E}

	1	2	3	4	5
1	0	1	2	1	2
2	1	0	1	2	1
3	2	1	0	3	2
4	1	2	3	0	1
5	2	1	2	1	0

	A	B	C	D	E
A	0	3	4	2	6
B	3	0	5	3	7
C	4	5	0	10	5
D	2	3	10	0	1
E	6	7	5	1	0

Permutation 1:

A-1, B-3, C-5, D-2, E-4

1A	2D	3B
4E	5C	

Cost= 2*(3*2+4*2+2*1+6*1+5*2+3*1+7*3+10*1+5*1+1*2)= 146

Permutation 2:

A-2, B-5, C-1, D-3, E-4

1C	2A	3D
4E	5B	

Cost= 2*(3*1+4*1+2*1+6*2+5*2+3*2+7*1+10*2+5*1+1*3)= 144

Permutation 3:

A-2, B-1, C-3, D-4, E-5

1B	2A	3C
4D	5E	

Cost = 2*(3*1+4*1+2*2+6*1+5*2+3*1+7*2+10*3+5*2+1*1) = 170

Permutation 4:

A-3, B-4, C-2, D-5, E-1

1E	2C	3A
4B	5D	

Cost = 2*(3*3+4*1+2*2+6*2+5*2+3*1+7*1+10*1+5*1+1*2) = 132

Although these four permutations are only a fraction of all available solutions, it can obviously be observed that a better solution quality can be received if the two facilities with the largest material flow among themselves are being put on (the) two locations which have the smallest distance to each other. In case of permutation number three the total opposite (largest distance- highest material flow) leads to tremendously high total costs. Therefore the main concern should be to find a permutation which arranges the facilities in a way so that the highest material flows are being multiplied with the smallest distances.

2.4 The QAPLIB[12]

Since the first formulation of a Quadratic Assignment Problem model a whole lot of international scientists have conducted researches in this field in order to create algorithms which are capable of finding feasible solutions. Many algorithms have been established as well as a lot of different problem instances by several researchers.

In 1991 a group of Austrian scientists from the Graz University of Technology had the idea to put up the QAPLIB to provide all these information and solutions to the scientific community. At that time the QAPLIB, as an up-to-date source, contained all accessible QAP instances.

In 1994 Burkard, Rendl and Karisch performed a major update and enhanced the QAPLIB by several new problem instances and a list of the best known solutions and best lower bounds.

A real turning point marked the year 1996 when it became a homepage in the World Wide Web not only just because of to the steadily growing community which was interested in this particular area of research. Also new data and solutions as well as an overall view over recent dissertations concerning the QAPLIB were included.

During the years 2000 and 2002 some other updates took place: several new problem instances, a list of people being involved in the QAP research work and improved best solutions to some existing instances were included. Since 2002, the homepage has been updated by Peter Hahn at the University of Pennsylvania.

The descriptions and the solutions to all problem instances are clearly structured and give some indication of how good the current best solutions are. In case of an existing optimal solution, the QAPLIB gives information about the solution value, the applied heuristic and the permutation. In case of a non existing optimal solution, the best feasible solution is given accompanied by a lower bound and a value for the relative gap between the bound and the best feasible solution.

[12] http://www.opt.math.tu-graz.ac.at/qaplib/#intro, called up on 22.03.2011

Some examples of heuristics being used for calculating the solutions of the QAPLIB are ant systems, scatter search, simulated annealing, genetic hybrids and tabu search.

3. Ant Colony Optimization

The research field of Ant Colony Optimization (ACO) can be traced back to the observations of ant colonies and their behavior in real nature. Although a single ant is a quite simple living thing and not capable of solving difficult tasks, a whole social insect society is able to overcome this lack of capabilities due to a high grade of organization and communication among themselves. Because of that it is easier for ant colonies than for a single ant to find a solution for a certain problem.[13]

The observation of real ants showed up that they use some kind of indirect communication technique called stigmergy which is defined by changes of the immediate environment. Often these changes are a result of the use of chemicals known as pheromones which are deposited by the ants on the ground in order to create an incentive for the other ants to follow the same way. This natural behavior inspired the development of ant algorithms which make use of some kind of artificial stigmergy to influence a group of artificial ants.[14]

3.1 Definition of Metaheuristics

Originally, a metaheuristic can be defined as an algorithmic concept which combines a construction heuristic with a local search procedure with the aim to carry out a broad search in the space of possible solutions without getting stuck in local optima.[15] It finds application to various types of complex problems (in particular combinatorial optimization problems) and has the advantage that the adaption to a specific problem can be realized without performing any serious changes to the general framework. The rising utilization of metaheuristics has

[13] see Dorigo/Stützle, 2004, p 1
[14] see Dorigo/Stützle, 2004, p 1
[15] see Glover/Kochenberger, 2002, p xi

improved the possibility for generating better solutions especially for large instance sizes.[16]

Although the solution values calculated by metaheuristics cannot provide conclusive proof of optimality, existing exact algorithms often produce solutions without any chance of reaching the best values found by metaheuristics. This observation led to a stronger research work in the field of metaheuristics.[17]

3.2 Biological Principles

The basic principles which formed the basis for the creation of the research field dealing with Ant Colony Optimization is deeply embedded in another scientific discipline called swarm intelligence. All the knowledge and the observations that came from several biological studies of insect societies (also including ant colonies) as well as the finding that social insects are able to hide the simplicity of their individuals by forming a highly structured organization in order to cope with complex problems made it possible to establish swarm intelligence as an emerging research field.[18]

3.2.1 Real Ants´ Behavior

As already mentioned above, the scientific findings concerning the foraging behavior of ants in nature represent the most important basis for all ant algorithms.

When ants leave their nest to search for some food they continually leave some chemical known as pheromone on the ground which disposes the other ants to follow the same path. This chemical-driven kind of indirect communication among the single ants is also known as stigmergy, a term which was introduced by French entomologist Pierre-Paul Grassé in the late fifties of the twentieth century. Stigmergy differs from other forms of communication in two main aspects:[19]

a. Stigmergy is in contrast to human forms of communication neither visible nor audible. Ants mediate their information by modifying their direct environment.

[16] see Dorigo/Stützle, 2004, p 33
[17] see Glover/Kochenberger, 2002, p xi f
[18] see Garnier/Gautrais/Theraulaz, 2007, p 3
[19] see Dorigo/Birattari/Stützle, 2006, p 28

b. Stigmergic information has to deal with limitations in terms of space. Pheromones have a certain range which means that it can only be distinguished by the immediate neighborhood.

3.2.2 The Double Bridge Experiment

The idea to observe this foraging behavior of ants in order to prove the existence of stigmergy led to a lot of experiments by several scientists. Probably the best known of those experiments is the so-called "*double bridge experiment"* which was carried out by Denebourg, Goss and other colleagues in the late nineties of the twentieth century.[20]

In their experiment they observed the behavior of an Argentine ant species named Iridomyrmex humilis by connecting a nest with a food source through the implementation of a diamond shaped bridge where initially both branches were of equal length.

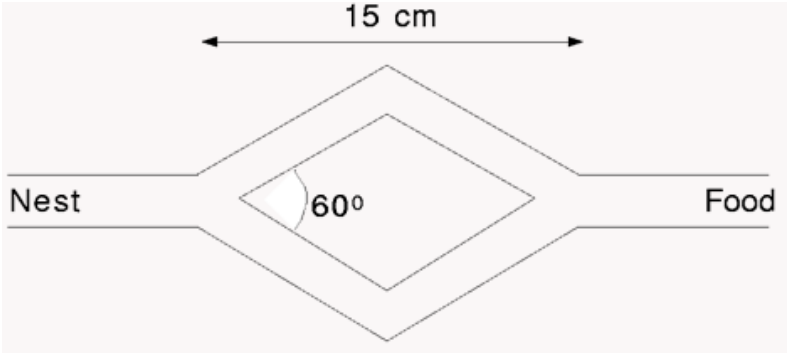


Figure 1: Branches of equal length[21]

At the beginning of the experiment the whole area between the nest and the food source is free from all pheromone but as soon as the ants start to explore the environment they continually leave pheromone on the ground.[22]

Due to the fact that there exist two branches which can be chosen to get to the food the ants randomly select their way in the starting phase. This leads to the initial observation that 50% of the ants choose the upper branch and the other 50% choose the lower branch. However, because of the fact that ants get

[20] see Denebourg et al., 1990, p 159 ff
[21] taken from http://www.scholarpedia.org/wiki/images/9/97/SameLengthDoubleBridge.png, called up on 27.01.2011
[22] see Denebourg et al., 1990, p 160 ff

stimulated and influenced in their decision making process by the pheromones left by their predecessors, a higher concentration of ants can be noticed on one branch after some time. So the amount of ants following one particular branch grows stronger over time until all insects exclusively go for the same way.[23]

In 1989 Goss and his colleagues made some amendments to this experiment in order to prove that ant colonies were able to find the shortest branch. The diamond shaped bridge is replaced with two new branches; a short branch and another branch which is twice as long as the shorter one.[24]

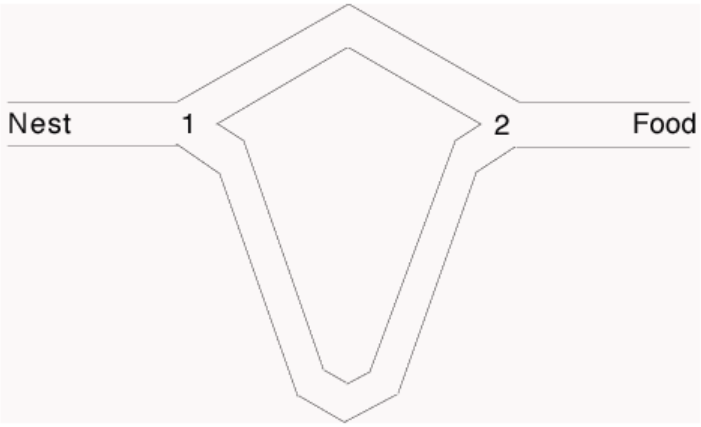


Figure 2: Branches have different length[25]

Again the ants head off towards the food source and randomly choose either the short or the longer branch. But the main difference to the other experiment is the time which an ant needs for the way from the nest to the food and back again; all the ants which choose the shorter branch need less time for their whole way and get back to the nest first. As a result these ants are able to leave more pheromone on the shorter branch than their colleagues on the longer branch can do in the same period of time which induces the pheromone level on the shorter branch to grow more rapidly. Now the other ants more likely follow the short path when leaving the nest and the ants leaving the food source also decide for this way due to the higher concentration of pheromone.[26] In most of

[23] see Denebourg et al., 1990, p 160 ff
[24] see Goss et al., 1989, p 579 ff
[25] taken from http://www.scholarpedia.org/article/File:DiffLengthDoubleBridge.png, called up 27.01.2011
[26] see Goss et al., 1989, p 579 ff

the experimental runs Goss et al. were able to observe the convergence towards the shorter branch.[27]

The process of this experiment can be compared to the generation of optimal solutions for the shortest-route problem.[28]

3.2.3 From Real to Artificial Ants

In order to generate solutions artificial ants are dependent on:[29]

a. Heuristic information which is problem specific
b. Artificial pheromone trails which reflect how desirable a certain solution component is

Besides that also some other assumptions have to be made:[30]

a. Artificial ants are not blind
b. The provided time for solution construction is discrete
c. Artificial ants have a memory in order to store the already added partial solutions (e.g. certain assignments, already walked ways, ...)

With this information artificial ants are able to continually build up their solutions by enhancing the already generated solution part through adding solution components in every step of the process.

3.3 Main procedures of the ACO Metaheuristic[31]

Since the first formulation of an ant algorithm, many successful adaptations have been developed and have found application to several combinatorial optimization problems (see chapter 4, section 4.2). Although many different variations of ant algorithms exist in the scientific world, they all have one thing in common: an ACO algorithm can always be described as the interaction of three different activities.

[27] see Dorigo/Stützle, 2004, p 4
[28] see Mullen et al., 2009, p 9609
[29] see Dorigo/Stützle, 2009, p 3 f
[30] see Dorigo/Maniezzo/Colorni, 1996
[31] see Dorigo/Stützle, 2004, p 37 f

```
procedure ACOMetaheuristic
      ScheduleActivities
            ConstructAntsSolutions
            UpdatePheromones
            DaemonActions % optional
      end-ScheduleActivities
end-procedure
```

Figure 3: Pseudo-code of an ACO metaheuristic[32]

ConstructAntsSolutions

In this step the artificial ants evaluate all possible solution components which can be added to the already existing partial solution. The new part of the solution is being selected by a probability based policy which includes the actual pheromones and sometimes heuristic information.

UpdatePheromones

This procedure deals with the modification of the pheromone trails. Depending on which assignments or connections an ant uses in its solution, new pheromone is being deposited on this assignment in order to make it more desirable for the following ants. Before doing so the natural process of evaporation has to be taken into consideration which means that the pheromone levels have to be decreases by a constant factor.

DaemonActions

Although this step is optional it can enhance the algorithm by helpful and optimizing procedures. A good example would be the implementation of a local search procedure in order to improve the solution constructed by a single ant. The daemon could also check all individual solutions and pick the best and/or the second best in order to deposit some additional pheromone.

[32] taken from Dorigo/Stützle, 2004, p 38

4. Ant Algorithms

4.1 Historical Development

The very first ant algorithm to be mentioned in scientific literature was Ant System (AS) which initially consisted of three different algorithms called ant-density, ant-quantity and ant-cycle. They differed from each other concerning the carrying out of the pheromone updates. While in ant-cycle the ants had to construct the whole solution before they were able to modify the pheromone values, in ant-density and ant-quantity the pheromone update was made after every solution construction step (e.g. after each assignment). Due to the fact that ant-cycle continually provided the best solution values, the research concentrated on the further development of this algorithm and called it Ant System.[33]

Its first application to the Traveling Salesman Problem (TSP) will be discussed later (see chapter 4, section 4.3).

In the following years numerous scientists got into the spirit of this new algorithm and tried to develop extensions or even to improve the basic idea. Several new algorithms were developed (see chapter 4, section 4.4 & 4.5) and the term ACO metaheuristic was found in order to define a new class of algorithms.[34]

4.2 Applications to several problems

During the last few years the interest in the research area of Ant Colony Optimization was continually growing and led to the modeling of several variants of older algorithms and also of new ant algorithms as well as to the application to a large number of problems.
In Figure 4 a short excerpt of applications including authors and year is given.

[33] see Dorigo/Stützle, 2003, p 260 f
[34] see Dorigo/Stützle, 2003, p 261

4.3 Ant System applied to the Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is an NP-hard combinatorial optimization problem which wants to minimize the total tour length of a salesman who has to visit a given list of customers (e.g. cities, private people, ...). The salesman has to start from his home and is allowed to include a customer exactly once in his tour before returning home.[35]

Problem Name	Authors	Year	Algorithm name
Traveling salesman	Dorigo, Maniezzo & Colorni	1991	AS
	Gambardella & Dorigo	1995	Ant-Q
	Dorigo & Gambardella	1996	ACS & ACS-3-opt
	Stützle & Hoos	1997	MMAS
	Bullnheimer, Hartl & Strauss	1997	ASrank
Quadratic Assignment	Maniezzo, Colorni & Dorigo	1994	AS-QAP
	Gambardella, Taillard & Dorigo	1997	HAS-QAP
	Stützle & Hoos	1998	MMAS-QAP
	Maniezzo & Colorni	1998	AS-QAP
	Maniezzo	1998	ANTS-QAP
	Wiesemann & Stützle	2006	Iterated Ants
Vehicle Routing	Bullnheimer, Hartl & Strauss	1996	AS-VRP
	Gambardella, Taillard & Agazzi	1999	HAS-VRP
Connection-oriented network routing	Schoonderwoerd, Holland, Bruten & Rothkrantz	1996	ABC
	White, Pagurek & Oppacher	1998	ASGA
	Di Caro & Dorigo	1998	AntNet-FS
	Bonabeau, Henaux, Guérin, Snyers, Kuntz & Théraulaz	1998	ABC-smart ants
Connection-less network routing	Di Caro & Dorigo	1997	AntNet & AntNet-FA
	Subramanian, Druschel & Chen	1997	Regular ants
	Heusse, Guérin, Snyers & Kuntz	1998	CAF
	van der Put & Rothkrantz	1998	ABC-backward
Sequential Ordering	Gambardella & Dorigo	1997	HAS-SOP
Graph Coloring	Costa & Hertz	1997	ANTCOL
Shortest common supersequence	Michel & Middendorf	1998	AS-SCS

Figure 4: List of several applications of ant algorithms[36]

[35] see Dorigo/Stützle, 2004, p 65 f
[36] modified from Dorigo/Di Caro, 1999

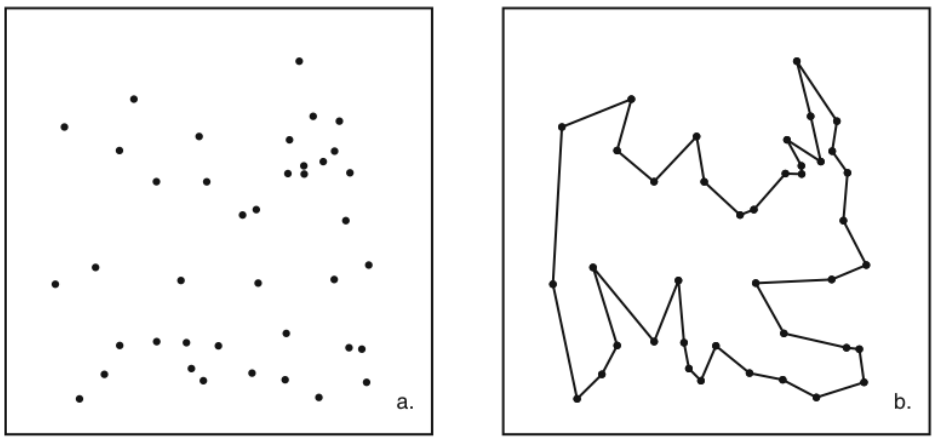


Figure 5: A 40 node TSP (a) and a whole tour as a possible solution (b)[37]

In all ant algorithms designed for the TSP the values of the pheromone matrix τ_{ij} represent the potential goodness of inserting city *j* directly after city *i* in the route.

Before starting the solution construction each ant *k* is assigned to a starting city (either randomly chosen or according to a certain criterion) and receives an internal memory which stores all completed construction moves of the ant. Then an ant performs the following steps:[38]

I. The next city to be visited is selected probabilistically by equation 6 which is based on some heuristic information $\eta_{ij} = 1/d_{ij}$ (where d_{ij} stands for the distance between the cities *i* and *j*) on the one hand and the pheromone trails on the other hand. The parameters α and β regulate the grade of influence on the result of the equation and N_i^k defines the set of all unvisited cities.

$$p_{ij}^k(t) = \frac{\left[\tau_{ij}(t)\right]^\alpha * \left[\eta_{ij}\right]^\beta}{\sum_{l \in N_i^k}[\tau_{il}]^\alpha * [\eta_{il}]^\beta} \quad if\ j \in N_i^k \tag{6}$$

II. Step number I. is repeated until all cities are included in the tour, then the ant returns to its starting point.

[37] modified from http://www.i-cherubini.it/mauro/blog/wp-content /uploads/ 2007/08/ images/ Dry_TSP_experiment.png, called up 06.01.2011

[38] see Dorigo/Stützle, 2003, p 261 f

III. After all ants have finished the construction of their tours, the pheromone trails are updated. First the pheromone values have to be lowered by a constant rate ρ (0 < ρ < 1) in order to fulfill the demand of the natural evaporation and to prevent the pheromone trails from unlimited growing. After that all ants (*m* is the number of ants) deposit their pheromones.

$$\forall(i,j)\ \tau_{ij}(t+1) = (1-\rho) * (\tau_{ij}(t)) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \tag{7}$$

The term $\Delta\tau_{ij}^{k}(t)$ denotes the amount of deposited pheromone on the edge between city *i* and *j*.

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{1}{L^{k}(t)} & if\ edge\ (i,j) is\ used\ by\ ant\ k \\ 0 & otherwise \end{cases} \tag{8}$$

$L^{k}(t)$ stands for the tour length of ant *k*.

4.4 The direct successors of Ant System

4.4.1 Elitist Ant System[39] [40]

One of the first improvements over the original ant system was the elitist ant system. This algorithm enables the current global best solution tour to deposit additional pheromone in order to help the edges of the best tour to get a stronger weight. The best tour is denoted with T^{gb}, where *gb* is the abbreviation for *global best*. The depositing of the additional pheromone happens during the normal pheromone update according to equation 9.

$$\tau_{ij}(t+1) = (1-\rho) * \left(\tau_{ij}(t)\right) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) + \Delta\tau_{ij}^{gb}(t) \tag{9}$$

[39] see Dorigo/Stützle, 2003, p 262
[40] see Dorigo/Stützle, 2004, p 73

The additional pheromone can be of the quantities

$$\Delta\tau_{ij}^{gb}(t) = \begin{cases} \frac{e}{L^{gb}(t)} & if\ edge\ (i,j) \in T^{gb} \\ 0 & otherwise \end{cases} \quad (10)$$

Where e is a positive integer and L^{gb} stands for the tour length of T^{gb}.

4.4.2 Rank-Based Ant System[41]

Another adaption of the original ant system which also follows and further develops the ideas of the elitist ant system is the rank-based ant system AS_{rank}. In this algorithm the amount of pheromone, which an ant is allowed to deposit, depends on the rank of the ant (the shorter the length of an ant´s tour, the more pheromone is provided for an ant).

Before starting the pheromone modification all ants are ranked according to the lengths of their tours (sorted in increasing order). In each pheromone updating only the first (w-1) ants as well as the best tour so far are allowed to modify the pheromone trails according to equation 11:[42]

$$\tau_{ij}(t+1) = (1-\rho) * \left(\tau_{ij}(t)\right) + \sum_{r=1}^{w-1} (w-r) * \Delta\tau_{ij}^{r}(t) + w * \Delta\tau_{ij}^{gb}(t) \quad (11)$$

4.4.3 Ant Colony System[43]

The ant colony system (ACS) algorithm distinguishes from the ant system algorithm in three main things:

a. Only the global best tour is brought in for the pheromone evaporation and the pheromone depositing
b. The ant colony system makes use of a different action choice rule in order to enhance the exploitation of the ants´ search experience
c. The single ants try to improve the exploration of alternative tours by constantly removing some pheromone when using a certain arc *(i, j)*.

[41] see Dorigo/Stützle, 2004, p 73 f
[42] see Dorigo/Stützle, 2004, p 73 f
[43] see Dorigo/Stützle, 2004, p 76 ff

Constructing the tour

In ACS the ants use a pseudorandom proportional rule in order to choose the cities to move to. This rule is given by

$$j = \begin{cases} argmax_{l \in N_i^k} \{\tau_{il}[\eta_{il}]^{\beta}\}, & if\ q\ \leq q_{0;} \\ J, & otherwise; \end{cases} \quad (12)$$

where variable *J* is randomly selected according to equation 6 (α = 1), q_0 is a parameter ($0 \leq q_0 \leq 1$) and q is a uniformly distributed variable [0, 1].

Updating the global pheromone trail

As mentioned before, only the ant with the global best tour is allowed to deposit pheromone in ACS which can be interpreted as a strong elitist strategy:

$$\tau_{ij}(t+1) = (1-\rho) * \left(\tau_{ij}(t)\right) + \rho * \Delta\tau_{ij}^{gb}(t) \quad (13)$$

During the first experiments with ACS the influence of the iteration best tour on the pheromone update was tested. Despite the relatively good findings for small TSP instances (≤ 100), in which the iteration best tour performed as good as the global best tour, the final result showed that the global best tour had the better overall performance (even for larger instance sizes).

Updating the local pheromone trail

In order to intensify the searching process and to circulate a stagnation behavior in ACS the ants use a special procedure to weaken the influence of the single pheromone values.

Each time an ant passes a certain arc *(i, j),* the corresponding pheromone value is updated by using the following equation:

$$\tau_{ij} = \left[(1-\xi) * \tau_{ij}\right] + (\xi * \tau_0) \quad (14)$$

The parameter τ_0 is given the initial value of the pheromone trails, and ξ is some kind of evaporation parameter (where $0 < \xi < 1$).

The main aim of this local pheromone update is to reduce the desirability of certain arcs to make a better exploration of different tours possible.

4.5 ACO applied to the QAP

The ant algorithms for the Traveling Salesman Problem can easily be adapted to the Quadratic Assignment Problem (see chapter 2 for a detailed problem description). While for the TSP the main aim is the construction of tours, a good solution for the QAP is characterized by an optimal and cost-effective assignment of facilities to the available locations. Despite the differing target settings, in both algorithms the use of pheromone trails shows a significant grade of influence.

The following sections describe some available ant algorithms for the QAP. The first one is the original ant system, which was adapted and first applied to the QAP in 1994.[44] The second algorithm to be presented here is the HAS-QAP, a hybrid ant-local search system.[45] The third one – the MAX-MIN ant system (MMAS) – is probably the most interesting one because it is the basis for the practical implementation part of this diploma thesis. MMAS was introduced by Stützle and Hoos and is an improvement over ant system.[46]

4.5.1 Ant System for the QAP[47]

Like all other ant algorithms this heuristic makes use of a set of *m* ants which assign a facility to a certain location in every construction step.

In order to guarantee that each ant doesn't include a location twice in its construction process, some kind of tabu list must be defined. This list stores all already occupied locations until a whole permutation is completed:

- $tabu_k$ is the tabu list for ant *k* (primarily a vector)
- $tabu_k(a)$ is the a-th element in the tabu list of ant *k*

In the ant system algorithm for the QAP the ants construct their solutions probabilistically by using the Roulette Wheel method (for further description see

[44] see Maniezzo/Colorni/Dorigo, 1994
[45] see Gambardella/Taillard/Dorigo, 1999
[46] see Stützle/Hoos, 1999
[47] see Maniezzo/Colorni/Dorigo, 1994, p 1 ff

chapter 6, section 6.1.2). The probability that ant k assigns facility *i* to location *j* can be calculated by:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \notin tabu_k} [\tau_{il}]^{\alpha} * [\eta_{il}]^{\beta}} & if\ j \notin tabu_k \\ 0 & otherwise \end{cases} \quad (15)$$

As in all other ant algorithms, the variable τ_{ij} stands for the pheromone trail (in this case the pheromone of the assignment location *j*- facility *i*), the variable η_{ij} marks the heuristic information (in this case the desirability or potential goodness of an assignment) and the parameters α and β determine the relative influence of these variables.

At the very beginning of the algorithm the potential goodness of a particular assignment has to be determined by calculating the two potential vectors *D* (distance potential vector) and *F* (flow potential vector) as well as the coupling matrix *A.* Given the distance matrix and the flow matrix the sums of each row form the two potential vectors.

D =

0	5	2	3	1
5	0	2	4	2
2	2	0	6	3
3	4	6	0	2
1	2	3	2	0

F =

0	2	3	1	2
2	0	4	2	1
3	4	0	3	1
1	2	3	0	3
2	1	1	3	0

The distance potentials d_i indicate the sum of all distances from one location to all other locations; the flow potentials f_j indicate the sum of all material flows from one particular facility to all the others.

$d_1 = 0+5+2+3+1 = 11$ $\quad f_1 = 0+2+3+1+2 = 8$

$d_2 = 5+0+2+4+2 = 13$ $\quad f_2 = 2+0+4+2+1 = 9$

$d_3 = 2+2+0+6+3 = 13$ $\quad f_3 = 3+4+0+3+1 = 11$

... ...

$$D = \begin{bmatrix} 11 \\ 13 \\ 13 \\ 15 \\ 8 \end{bmatrix} \qquad F = \begin{bmatrix} 8 \\ 9 \\ 11 \\ 9 \\ 7 \end{bmatrix}$$

The lower a flow potential, the less important this activity is for the whole network; the lower a distance potential of a certain node, the more barycentric it is considered in the system.

By taking these two potential vectors as a basis one may calculate the coupling matrix A with its elements a_{ij} by forming the products $d_i * f_j$. The potential goodness η_{ij} of a particular assignment can then be obtained by defining the inverse of the coupling matrix elements $\eta_{ij} = 1/ a_{ij}$.

$a_{11} = 11*8 = 88$ $a_{21} = 11*9 = 99$ $a_{31} = 11*11 = 121$

$a_{12} = 13*8 = 104$ $a_{22} = 13*9 = 117$ ...

$a_{13} = 13*8 = 104$ $a_{23} = 13*9 = 117$

$a_{14} = 15*8 = 120$ $a_{24} = 15*9 = 135$

$a_{15} = 8*8 = 64$ $a_{25} = 8*9 = 72$

$$A = \begin{bmatrix} 88 & 99 & 121 & 99 & 77 \\ 104 & 117 & 143 & 117 & 91 \\ 104 & 117 & 143 & 117 & 91 \\ 120 & 135 & 165 & 135 & 105 \\ 64 & 72 & 88 & 72 & 56 \end{bmatrix}$$

Now an ant is able to start the solution construction by starting with the facility which has the greatest flow potential and assigning it to the location obtained by equation (15). The pheromone trails are updated according to equation (7).

4.5.2 HAS-QAP[48]

Before giving a more detailed description of the HAS-QAP, it is necessary to give a short overview of the most important facts.

The greatest difference between the HAS-QAP – a hybrid ant colony system – and other ant algorithms is that ants use the pheromone trails in a non-standard

[48] see Gambardella/Taillard/Dorigo, 1999

way. Normally the pheromone trails are consulted for the construction of feasible solutions; in the HAS-QAP they are used only to modify existing solutions. After this modification based on the pheromone values, an additional local search is performed.

The updating process of the pheromone values happens by taking into account only the best solution so far. This global update considerably shortens the solution finding process; additionally, this effect is increased by the intensification mechanism which can also lead to an early convergence. The intensification mechanism helps the algorithm to solve the problem of choosing the starting solution for an ant (during each iteration). If at the end of the iteration the solution of an ant is worse than at the beginning, the ant will again choose the solution from the beginning of the iteration.

Because of this inconvenience mentioned before, there exists the possibility to activate a diversification mechanism in order to prevent the algorithm to converge too early. It consists of the erasing of all the pheromone trails and an additional re-initialization of the ants´ solutions.

Detailed description of HAS-QAP

This section explains the individual steps of the HAS-QAP algorithm (as shown in Figure 6) more precisely:

Initialization phase- solutions

The initial solution which is assigned to an ant is randomly generated and goes through a local search procedure (see section "*Manipulating the solutions by local search*") in order to optimize it.

Initialization phase- pheromone matrix

At the beginning of the algorithm all values of the pheromone matrix τ_{ij} are set to the same initial value τ_0.

```
*initialization*
Generate m random initial permutations π¹(1), ..., πᵐ(1), each one
      associated to an ant
Improve π¹(1), ..., πᵐ(1) with the local search procedure
Let π* be the best solution
Initialize the pheromone trail matrix T
Activate intensification
*main loop*
For i = 1 to I^max repeat
      *solution manipulation*
      For each permutation πᵏ(i) (1 ≤ k ≤ m) do
            Apply R pheromone trail swaps to πᵏ(i) to obtain π̂ᵏ(i)
            Apply the local search procedure to π̂ᵏ(i) to obtain π̃ᵏ(i)
      End For
      *intensification*
      For each ant k do
            If intensification is active
                  Then πᵏ(i+1) ← best permutation between πᵏ(i) and
                  π̃ᵏ(i)
            Else πᵏ(i+1) ← π̃ᵏ(i)
      End For
      If ∀k πᵏ(i+1)= πᵏ(i) then deactivate intensification
      If ∃k such that f(π̃ᵏ(i)) < f(π*)
            Then
            Update π*, the best solution found so far
            Activate Intensification
      *pheromone trail updating*
      Update the pheromone trail matrix
      *diversification*
      If S iterations have been performed without improving π* then
            Perform a diversification
End For
```

Figure 6: The HAS-QAP algorithm[49]

Manipulating the solutions using pheromones

The first part of the manipulation of solutions performs *R* swaps to the solution π^k to obtain the new permutation $\hat{\pi}^k$. The two elements to be swapped are chosen according to the following rule: first, an index *r* (between 1 and *n*) has to be selected. Second, depending on the value of *r* an index *s* ($s \neq r$) can be chosen by employing one of two different policies:

1. set *s* to a value so that $\tau^k_{r\pi_s} + \tau^k_{s\pi_r}$ is maximized; with probability q
2. choose *s* with probability $\frac{\tau^k_{r\pi_s} + \tau^k_{s\pi_r}}{\sum_{j \neq r} (\tau^k_{r\pi_j} + \tau^k_{j\pi_r})}$; with probability (1- q)

[49] modified from Gambardella/Taillard/Dorigo, 1999, p 169

After the selection of the two indices the elements π_s^k and π_r^k can be swapped.

Manipulating the solutions by local search

This neighborhood search is based on a first improvement strategy and examines all possible swaps of the elements π_i and π_j of π . The difference in the objective function can be determined by:

$$\begin{aligned}\Delta(\pi,i,j) = {} & (d_{ii}-d_{jj})\left(f_{\pi_j\pi_j}-f_{\pi_i\pi_i}\right)\\ & +(d_{ij}-d_{ji})\left(f_{\pi_j\pi_i}-f_{\pi_i\pi_j}\right)\\ & +\sum_{k\neq i,j}(d_{ki}-d_{kj})\left(f_{\pi_k\pi_j}-f_{\pi_k\pi_i}\right)\\ & +(d_{ik}-d_{jk})\left(f_{\pi_j\pi_k}-f_{\pi_i\pi_k}\right)\end{aligned} \quad (16)$$

If an improving swap of two elements is found, this swap is performed immediately.

Intensification

The intensification mechanism pursues the goal of exploring the neighborhood of the best solution so far more exactly. Intensification is active as long as at least one ant is capable of improving its solution. In Figure 7 a typical intensification mechanism is demonstrated: on the vertical axis the solution quality is measured (of ant k and the best known solution value), on the horizontal axis three of the main steps of an HAS-QAP algorithm are represented (the initial solution, manipulating the solutions using pheromones, manipulating the solutions using local search).

According to Figure 7, the intensification mechanism is not active at the beginning of the algorithm → so we have to set $\pi^k(i+1) \leftarrow \tilde{\pi}^k(i)$. After iteration i+1 a new best solution is found which requires the activation of intensification and $\pi^k(i+2) \leftarrow \tilde{\pi}^k(i+1)$. At the end of iteration i+2, due to the fact that intensification is active and the solution $\pi^k(i+2)$ is better than $\tilde{\pi}^k(i+2)$, $\pi^k(i+3)$ receives the solution value of $\pi^k(i+2)$.

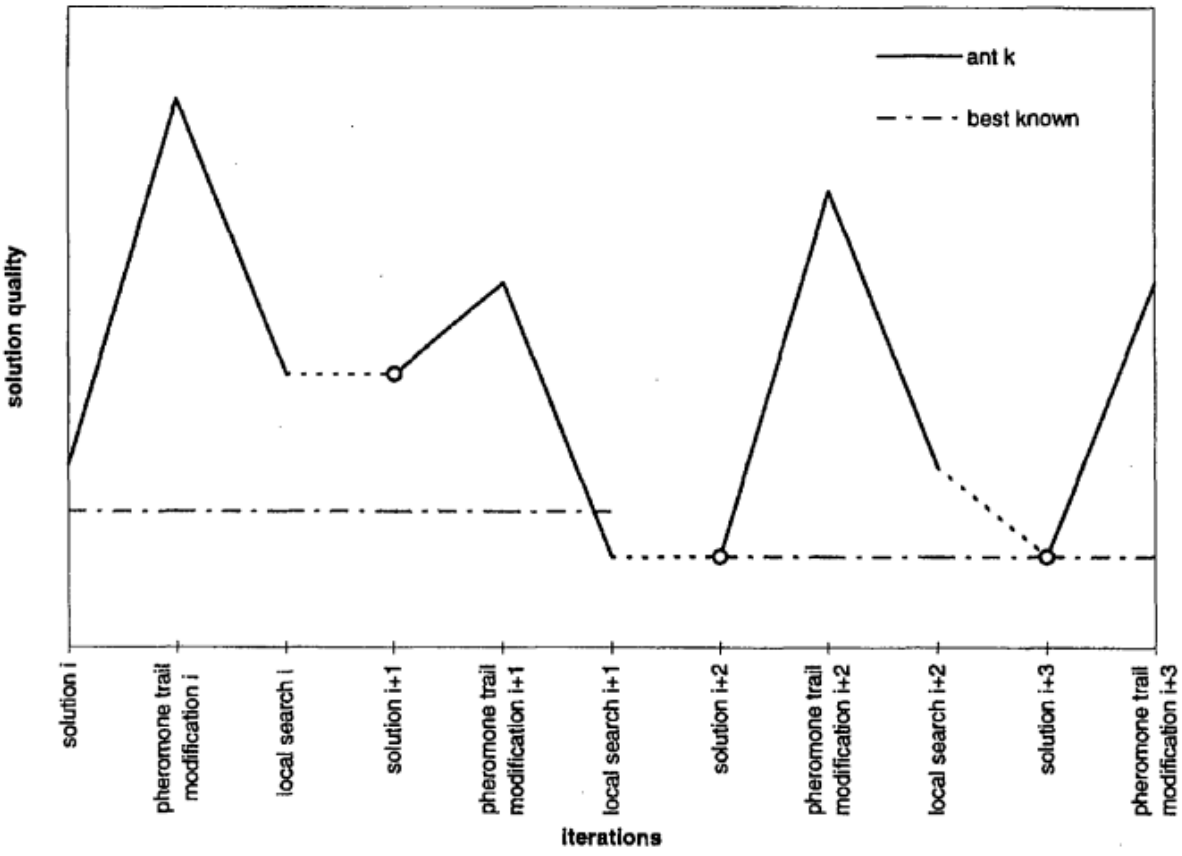


Figure 7: Example for intensification[50]

Updating the pheromone trails

As mentioned above, the pheromone update in the HAS-QAP is performed only by the global best solution π^*, which leads to a faster convergence of the algorithm. Before doing so, the usual pheromone evaporation has to be realized by using choosing parameter ρ and using the following equation:

$$\tau_{ij}(t+1) = (1-\rho) * \left(\tau_{ij}(t)\right) \tag{17}$$

Diversification

The diversification mechanism consists in generating new starting solutions for the ants (only one ant keeps the global best solution) and re-initializing the pheromone matrix. It is activated if no new best solution is generated during the last *S* iterations.

[50] taken from Gambardella/Taillard/Dorigo, 1999, p 170

4.5.3 MAX-MIN Ant System (MMAS)[51]

Since the first appearance of ant algorithms in the scientific literature, there has always been a strong interest to improve the performance of these algorithms in order to guarantee a better quality of solutions.

A lot of research projects came to the finding that a stronger utilization of the global best solution can have an enormous influence on the efficiency of the algorithm. Unfortunately a higher influence rate of the best solution can lead to early search stagnation. So the main aim was to create an algorithm which combines an effective use of the best solutions with a special mechanism for avoiding early stagnation.

The algorithm which is capable of meeting these requirements – the MAX-MIN Ant System – contains three special functions which distinguishes the MMAS from the normal ant system:

1. In the initialization phase the pheromone trails are set to a value $\tau_{\max}$ in order to allow a higher exploration
2. In MMAS only one single ant is allowed to update the pheromone trails after each iteration; this can be either the ant with the best known solution (global best solution) or the best ant of the current iteration (iteration best solution)
3. An infinite rise of the pheromone values and therefore stagnation can be avoided by introducing an interval for the pheromone trails $[\tau_{\min}, \tau_{\max}]$

ad 1- pheromone trail initialization

The initialization of the pheromone matrix has to be made with a very high value for τ_0 which can be chosen arbitrarily. By keeping to this rule, it can be guaranteed that all pheromone trails even out in the specified interval $[\tau_{\min}, \tau_{\max}]$ (normally exactly at $\tau_{\max}$) after the first iteration.

ad 2- pheromone trail updating

In MMAS the pheromone trail updating is realized according to

[51] vgl. Stützle/Hoos, 2000, p 898 ff

$$\tau_{ij}(t+1) = \rho * \tau_{ij}(t) + \Delta\tau_{ij}^{best} \quad (18)$$

where

$$\Delta\tau_{ij}^{best} = \frac{1}{f(s^{best})} \quad (19)$$

and $f(s^{best})$ refers to either the global best solution *(s^{gb})* or the iteration best solution *(s^{jb}).* This idea of using one single ant for the pheromone update has already been described in ACS; in ACS mainly the global best solution is used whereas in MMAS the main focus lies on the iteration best solutions.

For updating the pheromones it is also very common to use mixed strategies which consist of using both s^{gb} and s^{jb} in a constantly changing order (e.g. using s^{gb} every 10 iterations). The best strategy is probably the dynamic mixed one which mainly uses the iteration best solutions coupled with a growing influence factor of the global best solution throughout the algorithm. This kind of compromise prohibits the search from concentrating too fast around the value of the global best solution in case of only using the s^{gb}. By including the s^{jb}, which is normally significantly different in every iteration, in this mixed strategy, not only the pheromone trails belonging to the global best solution are updated but also less promising solutions get reinforced. In the practical part of this diploma thesis also a dynamic mixed strategy was implemented.

ad 3- pheromone trail limits

Every algorithm can have to face the problem of search stagnation which does not depend on the pheromone updating strategy. Search stagnation occurs if there exist significantly high pheromone trails for a certain permutation; these pheromone trails then have an essential influence on the solution construction of the ants (this situation is even worse in MMAS because in this algorithm the influence parameter β of the heuristic information is normally set to zero which means that the probability choice rule only depends on the values of the pheromone matrix) which can lead to an endless circle of constant reinforcement of the best solution.

In order to avoid this undesirable behavior the MMAS makes use of so called pheromone trail limits $[\tau_{\min}, \tau_{\max}]$which help to keep the pheromone trails τ_{ij} within a certain range. In every iteration it has to be verified that the constraint $\tau_{\min} \leq \tau_{ij}(t) \leq \tau_{\max}$ holds by checking the pheromone trails:

- if $\tau_{ij}(t) > \tau_{\max}$, set $\tau_{ij}(t) = \tau_{\max}$
- if $\tau_{ij}(t) < \tau_{\min}$, set $\tau_{ij}(t) = \tau_{\min}$

As mentioned above, the pheromone trails are initialized with a very high number which helps the pheromone values to even out in the interval $[\tau_{\min}, \tau_{\max}]$ after the first iteration. In MMAS this interval of trail limits is always updated if there is a new global best solution available.

$\tau_{\max}$ is updated as follows:

$$\tau_{ij}^{max}(t) = \sum_{i=1}^{t} \rho^{t-i} \frac{1}{f(s^{opt})} + \rho^{t} \tau_{ij}(0) \tag{20}$$

because of $\rho < 1$ the sum can be rewritten as

$$\frac{1}{1-\rho} * \frac{1}{f(s^{opt})} \tag{21}$$

$f(s^{opt})$ stands for the objective function value of the optimal solution. By substituting $f(s^{opt})$ for $f(s^{gb})$ (the solution value of the global best solution), the updating procedure of $\tau_{\max}$ is triggered with every new global best solution.

Going out from the value of $\tau_{\max}$ the updating of $\tau_{\min}$ can now be realized. For this procedure we need the two possibilities p_{best} – possibility of constructing the best solution after the convergence of MMAS – and p_{dec} – possibility of an ant choosing all permutations with pheromone trail $\tau_{\max}$ to construct its solution – in order to inset them into the following formulas. Assuming that $p_{best} > 0$ we can determine

$$p_{dec} = \sqrt[n]{p_{best}} \tag{22}$$

By setting *avg=n/2* to the value of p_{dec} can also be calculated by:

$$p_{dec} = \frac{\tau_{max}}{\tau_{max} + (avg - 1)\tau_{min}} \tag{23}$$

By transforming this equation for $\tau_{\min}$ we get:

$$\tau_{min} = \frac{\tau_{max}(1 - p_{dec})}{(avg - 1)p_{dec}} = \frac{\tau_{max}(1 - \sqrt[n]{p_{best}})}{(avg - 1)\sqrt[n]{p_{best}}} \tag{24}$$

5. Local Search Methods

The research on metaheuristics supplies us with the knowledge that the best solution values can be achieved by combining a well thought-out mechanism for generating the initial solution with an effective local search method. Probably the best working algorithms are the iterated local search algorithms, which iteratively try to improve the initial solution by using a certain local search method. The main aim of a local search procedure is to find the local optimum in the neighborhood of a starting solution constructed by an ant. The probability for good local search methods to improve the solution value is quite high because a neighborhood different to the one of the initial construction phase can be sifted through. A very popular local search procedure, especially for the Traveling Salesman Problem, is the k-exchange which provides different variants like for example 2-opt (see section 5.1), 2.5-opt and 3-opt.[52]

Although the combination of local search and a constructing mechanism is always a good choice for generating solutions, the two main aims to be optimized (efficiency and effectiveness) are mutually exclusive. Either you have an algorithm which generates high quality solutions within an above average time span or the algorithm works really fast and the quality of the solution has to

[52] see Dorigo/Stützle, 2004, p 92 f

suffer under it.[53] Because of that it has to be chosen between a best improvement strategy and a first improvement strategy:

- *Best Improvement* → the local search procedure sifts all possible solutions of the neighborhood carefully in order to find the best one
- *First Improvement* → as soon as the local search finds a better solution, the procedure is being stopped

5.1 2-opt

This simple local search method deals with the exchange of two different solution components. In case of the Traveling Salesman Problem, the algorithm chooses two edges from the solution and swaps them in the hope of a new best solution.[54]

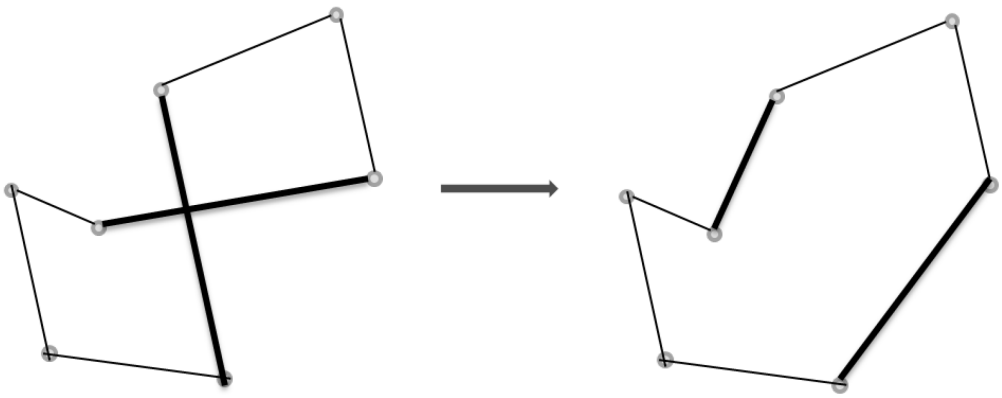


Figure 8: 2-opt procedure for the TSP

This local search method can find application to the Quadratic Assignment Problem in a straightforward way. Instead of switching two edges, 2-opt removes two already assigned facilities from their locations to switch them in order to improve the value of the objective function. In case of a first improvement strategy, if a better solution is found, the initial solution is replaced. This happens as long as there are no further improvements possible. The exact number of possible swaps in the neighborhood of a solution can always be calculated using the term $n*(n-1)/2$.[55]

[53] see Dorigo/Stützle, 2004, p 92 f
[54] http://en.wikipedia.org/wiki/2-opt, called up on 22.03.2011
[55] see Ramkumar/Ponnambalam/Jawahar, 2009, p 623

There exist two main groups of interchanges for the QAP:

1. *Pairwise interchange:* the two facilities don´t need to be adjacent
2. *Adjacent pairwise interchange:* the two facilities have to be adjacent (here the number of possible swaps is reduced to $(n-1)$)

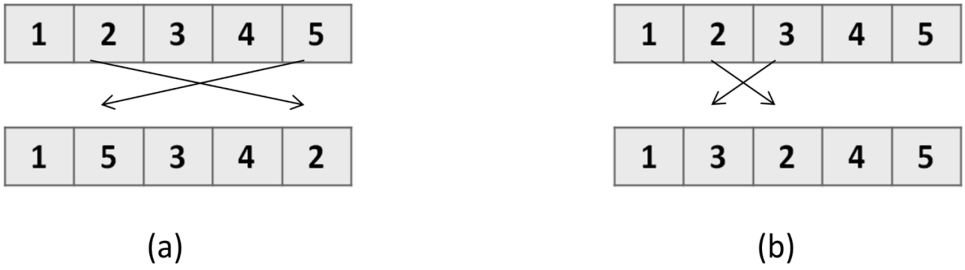


Figure 9: Pairwise interchange (a), adjacent pairwise interchange (b)

5.1.1 A short example: 2-opt for the QAP

1	**2**
3	**4**

	1	2	3	4
1	0	1	2	1
2	1	0	1	2
3	2	1	0	3
4	1	2	3	0

	A	B	C	D
A	0	3	4	2
B	3	0	5	3
C	4	5	0	10
D	2	3	10	0

Initial solution: 1-A, 2-B, 3-C, 4-D

Cost = 2*(1*3+2*4+1*2+1*5+2*3+3*10) = 108

Changing facilities B & C

Solution: 1-A, 2-C, 3-B, 4-D

Cost = 2*(1*4+ 2*3+1*2+1*5+2*10+3*3) = 92

Improvement of 16

Changing facilities A & C

Solution: 1-C, 2-B, 3-A, 4-D

Cost = 2*(1*5+2*4+1*10+1*3+2*3+3*2) = 76

Improvement of 32

Changing facilities A & D

Solution: 1-D, 2-B, 3-C, 4-A

Cost = 2*(1*3+2*10+1*2+1*5+2*3+3*4) = 96

Improvement of 12

5.2 Iterated ants – a hybridization of ACO

To hybridize an ant algorithm means that an effective local search mechanism is added to the algorithm in order to search for better solutions in the neighborhood of the initial solution (constructed by the ants). Many research projects dealt with this topic and tried to find the optimal metaheuristic to improve ants´ solutions – a considerable example would be the tabu search for the Quadratic Assignment Problem. Beside these findings a lot of different ways of hybridizing ant algorithms have been developed. One of them supports the idea of letting the single ants construct their solutions by starting from partial solutions. Normally the ants start their construction mechanism from scratch but starting from partial solutions – which are obtained by removing components from an ant´s initial solution – presents two very important advantages:[56]

1. The solution finding can be accelerated by far
2. The best parts of a solution are directly exploitable

Being one of the most important elements and ideas behind the implementation part of this diploma thesis, I would like to describe the Iterated Ants algorithm in detail which is faithful to the Iterated Greedy (IG) metaheuristic. Figure 10 shows a general outline of such an IG algorithm.

```
procedure Iterated_Greedy
    s_0 := GenerateInitialSolution;
    s := LocalSearch(s_0);          %optional
    repeat
        s_p := Destruction(s)
        s' := Construction(s_p)
        s' := LocalSearch(s')       %optional
        s := AcceptanceCriterion(s, s')
    until termination condition met
end
```

Figure 10: Outline of an IG algorithm[57]

[56] see Dorigo/Stützle, 2009, p 18
[57] taken from Ruiz/Stützle, 2008

A typical Iterated Greedy algorithm starts with generating an initial solution followed by a local search procedure. Once this starting solution *s* is available, the algorithm begins with the main loop which consists of four mechanisms:[58]

1. *Destruction:* this procedure is responsible for destroying a certain amount (fixed or variable) of solution components of *s* which results in the partial solution s_p; there exist a lot of different destroy algorithms which will be discussed later
2. *Construction:* in Iterated Ants the construction mechanism normally uses the same probability choice rule as in ACO; the partial solution s_p is reconstructed bit by bit until a whole permutation *s´* is obtained
3. *Local Search:* in the main loop of an Iterated Greedy algorithm there exists the possibility of running through a second local search procedure; this is optional and should be well thought-out in terms of a longer runtime
4. *Acceptance Criterion:* in this step we are free to choose how to accept a solution. For example if the solution value of *s´* is better than the value of *s*, we can take *s´* as the new *s* and start again with the main loop of the Iterated Greedy algorithm

In Iterated Ants algorithms it is assumed that each ant implements its own Iterated Greedy algorithm. This means that in all iterations every ant creates a complete candidate solution and then tries to improve it by using the Iterated Greedy algorithm. In 2006 W. Wiesemann and T. Stützle made an experimental study dealing with the idea of Iterated Ants and introduced 3 different destroy mechanisms:[59]

1. *rand*: the solution parts to be destroyed are chosen randomly
2. *prob*: the probability of removing a certain solution component depends on the belonging pheromone trail τ_{ij}; the higher this pheromone value, the higher is the possibility that this component is removed from the candidate solution. This means that the probability is proportional to the pheromone trail.

[58] see Dorigo/Stützle, 2009, p 18
[59] see Wiesemann/Stützle, 2006, p 182 f

3. *iprob*: this destroy mechanism is completely the opposite of the previous one and promotes a probability which is inversely proportional to the pheromone trails; the lower the pheromone value, the higher is the possibility that this component is removed from the candidate solution.

5.3 Very Large Scale Neighborhood Search (VLSN)

All VLSN algorithms are known for generating solutions of very high quality. They search a large neighborhood – this neighborhood is normally reduced to a subset of all possible solutions because otherwise the searching time would exceed all acceptable time limits – in order to find local optima. VLSN and LNS (which is going to be described in section 5.3.1) are two very similar terms which can easily be mixed up. So a very important fact to mention here is that the term VLSN stands for the class of algorithms dealing with very large neighborhood searches and LNS only denotes a certain metaheuristic belonging to this class. However, the main characteristic each algorithm needs to have to belong to the class of VLSN algorithms is an exponential growth of the available neighborhood depending on the instance size of the problem.[60] The class of VLNS algorithms can be divided into three categories:[61]

1. *Variable depth methods*

   The main idea of Variable Depth Neighborhood Search (VDNS) algorithms is not to start with the whole neighborhood but to gradually enhance its size. For example, by using the k-exchange neighborhood: At first the algorithm starts with the 1-exchange neighborhood N_1 but every time it gets trapped in a local minimum, the neighborhood is extended by the 2-exchange neighborhood N_2 (then by N_3, N_4,... N_k).

[60] see Pisinger/Ropke, 2010, p 399 f
[61] see Ahuja et al., 2002, p 79 ff

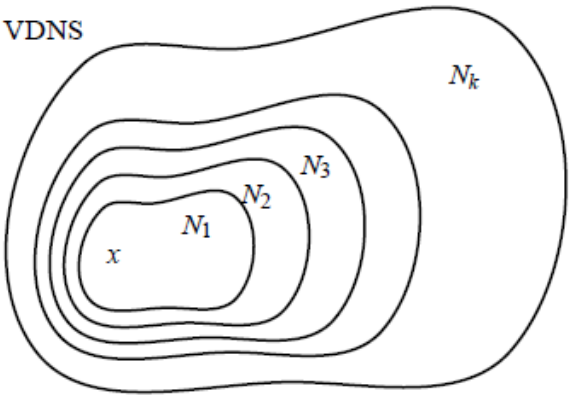


Figure 11: Gradual extension of the neighborhood in VDNS[62]

2. *Network flow based improvement methods*

   They can be divided into three different groups:

   - Cyclic exchange neighborhood: means that parts are moved from one subset to another; let *A* be the whole problem and S_1,... S_k are the subsets ($k = 4$) – a cyclic exchange would be if you shift one element of S_1 to S_2, one element from S_2 to S_3, one from S_3 to S_4 and last but not least one element from S_4 to S_1.
   - Path exchanges: is some kind of swap neighborhood and consists of deciding on a random number of independent swaps and realizing them together
   - Assignment neighborhood: this so-called exponential neighborhood structure can be obtained by making reasonable assignments in an improvement graph

3. *Methods based on constraining the original problem*

   Although one of the main characteristics of NP-hard problems is the fact that they can´t be solved in polynomial time, there exists the possibility to enhance the initial problem by additional constraints or even restrictions. The resulting neighborhood may be solved within an acceptable period of time.

[62] taken and modified from Pisinger/Ropke, 2010, p 403

5.3.1 Large Neighborhood Search (LNS)[63]

The LNS metaheuristic was introduced in 1998[64] and was originally designed for solving the Vehicle Routing Problem.

In this algorithm the neighborhood is determined by firstly destroying parts of the solution and secondly repairing them again. Therefore, two well thought-out methods are needed as well as an effective element of stochasticity which is included in the destroy method in order to guarantee that different solution parts are chosen for destruction in every retrieval. So the whole neighborhood *N(x)* of an initial solution *x* is the resulting set of solutions of the interplay between destroying and repairing.

In Figure 12 a pseudo-code of a typical LNS algorithm is shown. At the beginning we have an initial feasible solution *x* as the main input (see line 1). The variable x^b stands for the global best solution which is found during the whole algorithm – it takes on the value of x before starting the main loop (see line 2). In line number 4 the function *r(d(x))* destroys parts of the solution *x* and repairs this partial solution afterwards. The outcome of this function is the variable x^t.

For the accept function in line number 5 exist a lot of possibilities how to implement it – a very popular one is to accept only improved solutions which means new solutions x^t with a smaller objective function value than *x* (even in the paper of Shaw[65] only improving solutions are allowed). In this case *x* is set to the value of x^t. In line number 8 we can see the comparison of the objective function values of x^t and the global best solution x^b. If the equation $c(x^t) < c(x^b)$ holds, the best solution will be updated by $x^b = x^t$. In the line before last the stopping criterion is checked – this can be for example a certain number of iterations or any other criterion the implementer is keen of. At last the global best solution is returned (see line 12).

[63] see Pisinger/Ropke, 2010, p 405 ff
[64] see Shaw, 1998
[65] see Shaw, 1998

Algorithm 1 Large neighborhood search

```
1: input: a feasible solution x
2: x^b = x;
3: repeat
4:     x^t = r(d(x));
5:     if accept(x^t, x) then
6:         x = x^t;
7:     end if
8:     if c(x^t) < c(x^b) then
9:         x^b = x^t;
10:    end if
11: until stop criterion is met
12: return x^b
```

Figure 12: Pseudo-code of an LNS algorithm[66]

Probably the most important considerations before implementing an LNS algorithm are the choice and the extent of the destroy mechanism. It has to be determined how many solution parts to destroy which can have an immense influence on the behavior of the whole algorithm. If a very large part of the initial solution *x* is destroyed then it may happen that the repair mechanism is extremely time consuming or provides solutions of worse quality. If the number of destroyed solution components is very low then the effectiveness of a neighborhood search is lost due to a failure caused by exploring only a minimized solution space. There exist several different suggestions in scientific papers: Ropke and Pisinger[67]recommend a random determining of the degree of construction which depends on the instance size; Shaw[68] considers a gradual increase of the removed components to be effective. It also has to be guaranteed that different solution components are removed in every invocation of the remove operator so that every part of the solution can possibly be affected.

5.3.2 Adaptive Large Neighborhood Search (ALNS)[69]

The Adaptive Large Neighborhood Search differs from the Large Neighborhood search in the number of destroy and repair operators permitted in the algorithm. In the ALNS it is allowed to use several different operators which have to be given a certain weight in order to control how often the method is deployed

[66] taken from Pisinger/Ropke, 2010, p 407
[67] see Ropke/Pisinger, 2006
[68] see Shaw, 1998
[69] see Pisinger/Ropke, 2010, p 409 f

during the algorithm. In contrast to the LNS, in ALNS each destroy/repair method creates its own neighborhood which leads to multiple neighborhoods. In Figure 13 the pseudo-code of an ALNS algorithm is shown.

Algorithm 2 Adaptive large neighborhood search

```
1: input: a feasible solution x
2: x^b = x; ρ^- = (1,...,1); ρ^+ = (1,...,1);
3: repeat
4:     select destroy and repair methods d ∈ Ω^- and r ∈ Ω^+ using ρ^- and ρ^+;
5:     x^t = r(d(x));
6:     if accept(x^t, x) then
7:         x = x^t;
8:     end if
9:     if c(x^t) < c(x^b) then
10:        x^b = x^t;
11:    end if
12:    update ρ^- and ρ^+;
13: until stop criterion is met
14: return x^b
```

Figure 13: Pseudo-code of an ALNS algorithm[70]

The set of destroy methods is denoted Ω^- while the set of repair methods is denoted Ω^+; as it can be seen in line number 4 $d \in \Omega^-$ and $r \in \Omega^+$. In line number 2 the two new weight parameters ρ^- (where $\rho^- \in R^{|\Omega^-|}$) and ρ^+ (where $\rho^+ \in R^{|\Omega^+|}$) are introduced which help to select the methods following the roulette wheel algorithm. In case of the repair methods, the probability of choosing method number *z* is calculated as follows:

$$\phi_z^+ = \frac{\rho_z^+}{\sum_{k=1}^{|\Omega^+|} \rho_k^+} \tag{25}$$

The formula for the destroy methods works in the same way with using ϕ_z^- instead of ϕ_z^+, ρ_z^- instead of ρ_z^+ and ρ_k^- instead ρ_k^+; last but not least the set of methods changes from Ω^+ into Ω^-.

[70] see Pisinger/Ropke, 2010, p 409

6. Implementation

The practical part of this diploma thesis deals with the idea to take the MMAS algorithm as proposed by Stützle and Hoos[71] as a basis and to replenish it with new local search methods. In the best case these worked out methods should search the neighborhood of a solution on the one hand very fast, and on the other hand they should provide feasible solutions of good quality. The basic idea of these new local search methods is the so called Iterated Ants idea by Wiesemann and Stützle[72] (see chapter 5, section 5.2 for more details) which primarily tries to destroy a solution and then to reconstruct it in order to obtain a better one. Due to the fact that all algorithms which have been implemented in the course of this diploma thesis use the same mechanism for reconstruction, the main research focused on the development and implementation of effective destroy mechanisms to create a valuable neighborhood of solutions. Five different algorithms have been implemented by using C++ as the programming language. The algorithms differ mainly from each other in the functionality of the local search methods – the algorithmic basis is always the same MMAS Basis Algorithm as presented below (see chapter 6, section 6.2.1). The only exceptional case is the MMAS 3 Iterated algorithm (see chapter 6, section 6.2.5) which is the only algorithm that doesn´t orient itself by the Iterated Ants idea but by the Large Neighborhood Search idea. The main objective here was to implement a MMAS algorithm that generates an initial solution which is then used to run through a typical LNS procedure. This LNS procedure contains the three proposed destroy mechanisms called random removal, product removal highest and product removal lowest which will be described below.

6.1 General Principles

As mentioned above, the algorithms have a lot of different procedures like the pheromone update or the Roulette Wheel method in common. It is important to remember to implement the algorithms in a way that they use to be very similar to each other in order to make the resulting solutions comparable.

[71] see Stützle/Hoos, 2000, p 898 ff
[72] see Wiesemann/Stützle, 2006, p 179 ff

6.1.1 Roulette Wheel Procedure[73]

In all five MMAS algorithms which are proposed in this diploma thesis, ants construct their solutions by following a special procedure known as the Roulette Wheel method. In all runs of the algorithm (the number of runs can also be interpreted as the number of ant colonies) a certain amount of individual ants use this procedure in order to randomize their assignments.

At the beginning all facilities have to be sorted in decreasing order and are stored in a vector. Each ant starts its solution construction with the first element of this vector (facility with the highest sum of flows to all other facilities) and calculates the probability of assigning facility *i* to location *j* according to equation (15). Because in MMAS algorithms the influence factor of the heuristic information is set to zero, the obtained probabilities exclusively depend on the pheromone trails (factor α is set to 1):

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}}{\sum_{l \notin tabu_k}[\tau_{il}]^{\alpha}} & if\ j\ \notin tabu_k \\ 0 & otherwise \end{cases} \tag{27}$$

After the calculation of all probabilities they even have to be cumulated. Afterwards a random number in the range [0, 1] is generated to determine the location to be taken:

	p21	p22	p24	p25	p26	p27
probability	0,12	0,22	0,03	0,25	0,29	0,09
cumulated	0,12	0,34	0,37	0,62	0,91	1

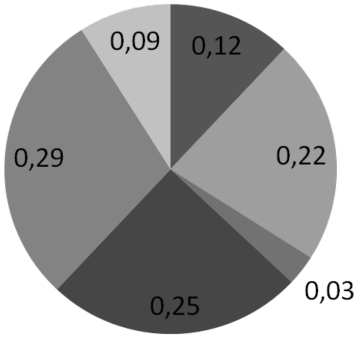


Figure 14: Roulette Wheel procedure

[73] http://en.wikipedia.org/wiki/Fitness_proportionate_selection, called up on 22.03.2011

Example: if the random number has the value 0,45 → facility 2 would be assigned to location 5 because 0,37 < **0,45** ≤0,62.

6.1.2 2-opt First Improvement

In all implemented algorithms the 2-opt local search procedure (for detailed description see chapter 5, section 5.1) is employed in two places: one the one hand 2-opt tries to optimize the initial solution generated by an ant and on the other hand it helps to improve the obtained assignment after the destroy and reconstruct mechanisms. This local search method is subject to a first improvement strategy which means that the loop of 2-opt starts over if a better solution is found. Due to the fact that no special termination criterion exists, it can be guaranteed that the 2-opt procedure keeps on passing through as long as an improved solution is available in the neighborhood.

For keeping the runtime of the algorithm low, the exchanges of the facilities are scored by a delta evaluation before they are actually switched:[74]

$$\Delta C_{xy}(b) = \sum_{i=1}^{n} (f_{ix} - f_{iy}) * \left[d(b_i, b_x) - d(b_i, b_y)\right] - 2f_{xy}d\left(b_x, b_y\right) \qquad (26)$$

The costs of the exchange $\Delta C_{xy}(b)$ depend on the original assignment *b* and the two facilities to be exchanged *x* and *y*. The higher the value of $\Delta C_{xy}(b)$ is, the more desirable the exchanging of *x* and *y* gets.

Initially the facilities in the first two locations are evaluated by setting *x=0* and *y=1* (the value refers to the index of a vector). The next evaluations would be *[x=0; y=2]*, *[x=0; y=3]*, *[x=0; y=4]* until *[x=0; y=n]* – then the algorithm continues with *[x=1; y=2]*, *[x=1; y=3]*, *[x=1; y=4]*, *[x=1; y=5]*,... and so on. The conditions $x \neq y$ and $x < y$ have to be fulfilled all the time in order to exclude redundant calculations.

6.1.3 Updating the Pheromone Trails

As already discussed in chapter 4-section 4.5.3, MMAS algorithms normally only make use of the iteration best solutions for updating the pheromone trails.

[74] see Askin/Standridge, 1993, p 219

In order to prevent an exclusive influence of the iteration best solutions we use a mixed strategy in our implementation:

- Iterations 1-9: use global best solution in every third iteration
- Iteration 10-24: use global best solution in every second iteration
- Iteration > 25: exclusive use of global best solution

This interplay between a constantly growing influence of the global best solution and a continually disappearing pheromone updating of the iteration best solution guarantees equable pheromone trails in the initial phase of the algorithm.

After all it is a natural cause of action that before updating the pheromone trails the evaporation of the pheromones (we choose the resistance factor ρ = 0,8) has to be realized. So the whole pheromone trail updating is carried out according to:

$$\tau_{ij}(t+1) = \rho * \tau_{ij}(t) + \frac{1}{f(s^{best})} \tag{28}$$

6.2 The Algorithms

For the practical part of this diploma thesis five algorithms have been implemented in C++. The *MMAS Basis* algorithm is a “normal” MMAS algorithm which only uses the 2-opt local search method to improve the initial solutions. The *MMAS Random Removal*, *MMAS Product Removal Highest* and *MMAS Product Removal Lowest* algorithms follow the idea of Iterated Greedy algorithms while the *MMAS 3 Iterated* is more an adaptation of a Large Neighborhood Search. All algorithms except the MMAS 3 iterated run through 1000 iterations – each including a colony of *k* = 5 ants. The resistance factor of the pheromone trails is set to ρ = 0,8 and the influence parameter of the pheromones is set to α = 1. According to equation (22) we set p_{best} = 0,005 for generating the pheromone trail limits. The pheromone trail initialization is done by setting $\tau_{\max}$ = 200. For all five different algorithms we will give a detailed description in the following sections.

6.2.1 MMAS Basis Algorithm

The *MMAS Basis Algorithm* was implemented as proposed by Stützle and Hoos.[75] This algorithm is the base of all other algorithms and is used to generate the initial solution before starting the destroy/reconstruct mechanisms. In Figure 15 the pseudo-code of the *MMAS Basis algorithm* is shown where *ib* = iteration best solution, *gb* = global best solution, *s* = best solution of an ant with costs *f(s)* and *s´* = best solution found during 2-opt with costs *f(s´)*.

```
For 1000 iterations
      For all 5 ants do
            Generate random number and assign facilities to locations
            following to the Roulette Wheel procedure in order to get s
            Calculate f(s)
            Do
                  2-opt local search to obtain s´ and f(s´)
                  If (f(s´) < f(s))
                        Set s = s´ and f(s) = f(s´)
                  EndIf
            While(improvement == true)
            If (s < ib)
                  Set ib = s
            EndIf
      EndFor
      If (ib < gb)
            Set gb = ib
            Update pheromone trail limits
      EndIf
      Update the pheromone trails
EndFor
```

Figure 15: Pseudo-code for the MMAS Basis algorithm

6.2.2 MMAS Random Removal

The *MMAS Random Removal* algorithm contains the easiest destroy mechanism of all implemented algorithms. By choosing the solution components to be removed with the help of a random number generator, the algorithm works much faster than *MMAS Product Removal Highest* and *MMAS Product Removal Lowest* because it doesn´t waste time on account of complicated calculations. Generated random numbers have to be in the range [0, (n-1)] and stand for the indices of the solution vector, e.g. the number 0 denotes location 1 in the vector.

[75] see Stützle/Hoos, 2000, p 898 ff

```
For 1000 iterations
      For all 5 ants do
            Generate initial solution s according to Roulette Wheel
            Calculate f(s)
            Do
                  2-opt local search to obtain s´ and f(s´)
                  If (f(s´) < f(s))
                        Set s = s´ and f(s) = f(s´)
                  EndIf
            While(improvement == true)
            For 6 runs
                  DESTRUCT
                        Calculate (n/6) components to be removed
                        Generate random numbers and remove solution
                        components from s → update f(s)
                  RECONSTRUCT
                        Reconstruct s according to Roulette Wheel
                  Do
                        2-opt local search to obtain s´ and f(s´)
                        If (f(s´) < f(s))
                              Set s = s´ and f(s) = f(s´)
                        EndIf
                  While(improvement == true)
            EndFor
            If (s < ib)
                  Set ib = s
            EndIf
      EndFor
      If (ib < gb)
            Set gb = ib
            Update pheromone trail limits
      EndIf
      Update the pheromone trails
EndFor
```

Figure 16: Pseudo-code for the MMAS Random Removal

The total number of components to be removed is set to (n/6) so it depends on the problem size how many components have to be chosen. It is important to mention here that the result of the division is always rounded down, e.g. in runs with the problem instance 35 there have to be removed 5 components because 35/6 results in 5. The reconstruct mechanism follows the same Roulette Wheel procedure as usual and the total number of destroy/reconstruct runs is set to 6. In Figure 16 the pseudo-code of the *MMAS Random Removal* is shown.

6.2.3 MMAS Product Removal Highest

As in *MMAS Random Removal*, in this algorithm the number of solution parts to be destroyed is set to (n/6). To determine these solution components, the *MMAS Product Removal Highest* algorithm applies a very complex procedure which includes the product of distances and flows and the total material flows among the individual facilities.

By starting from the initial solution of an ant the algorithm calculates the product of the distance between two locations and the flow between the corresponding facilities according to equation (29) and stores the five highest products in a vector.

$$d_{ij} * f_{\pi(i)\pi(j)} \tag{29}$$

Then exactly one product has to be chosen by using the Roulette Wheel method. For this each product is allotted a probability which is calculated by $product/(\sum products)$. By following the Roulette Wheel procedure the probabilities are cumulated, a random between [0, 1] is generated and the certain product is selected.

The two locations and the two facilities which are assigned to this product are the first components to be removed. Now the algorithm searches for the facility which has the highest material flow to the already destroyed facilities and removes it (and the corresponding location) as well. This step is repeated until the number of necessary removals is reached.

The reconstruction of the solution also follows the Roulette Wheel procedure. After this recreation the 2-opt local search method is applied for the second time in order to better the solution until no further improvements are possible. In Figure 17 the pseudo-code of the algorithm is shown.

```
For 1000 iterations
      For all 5 ants do
            Generate initial solution s according to Roulette Wheel
            Calculate f(s)
            Do
                  2-opt local search to obtain s´ and f(s´)
                  If (f(s´) < f(s))
                        Set s = s´ and f(s) = f(s´)
                  EndIf
            While(improvement == true)
            For 6 runs
                  DESTRUCT
                        Calculate (n/6) components to be removed
                        Generate 5 highest products and choose one
                        probabilistically → destroy the two
                        corresponding locations and facilities
                        Destroy facilities with highest flow to already
                        removed facilities until all necessary
                        components are destroyed
                  RECONSTRUCT
                        Reconstruct s according to Roulette Wheel
                  Do
                        2-opt local search to obtain s´ and f(s´)
                        If (f(s´) < f(s))
                              Set s = s´ and f(s) = f(s´)
                        EndIf
                  While(improvement == true)
            EndFor
            If (s < ib)
                  Set ib = s
            EndIf
      EndFor
      If (ib < gb)
            Set gb = ib
            Update pheromone trail limits
      EndIf
      Update the pheromone trails
EndFor
```

Figure 17: Pseudo-code for the MMAS Product Removal Highest

6.2.4 MMAS Product Removal Lowest

This algorithm is more or less the same as *MMAS Product Removal Highest* with only one exception: after the destruction of the first two locations with the corresponding facilities the algorithm removes the facilities with the lowest flow instead of the highest flow. In Figure 18 the pseudo-code of these destroy/reconstruct mechanisms is shown.

```
For 6 runs
      DESTRUCT
            Calculate (n/6) components to be removed
            Generate 5 highest products and choose one
            probabilistically → destroy the two corresponding
            locations and facilities
            Destroy facilities with lowest flow to already removed
            facilities until all necessary components are destroyed
      RECONSTRUCT
            Reconstruct s according to Roulette Wheel
      Do
            2-opt local search to obtain s´ and f(s´)
            If (f(s´) < f(s))
                  Set s = s´ and f(s) = f(s´)
            EndIf
      While(improvement == true)
EndFor
```

Figure 18: Destroy/reconstruct of MMAS Product Removal Lowest

6.2.5 MMAS 3 Iterated

In contrast to the previous algorithms the *MMAS 3 Iterated* makes use of a different local search procedure which is very similar to an Adaptive Large Neighborhood Search. The procedure contains all three destroy mechanisms of *MMAS Random Removal*, *MMAS Product Removal Highest* and *MMAS Product Removal Lowest*. In order to maintain the total number of 6 destroy/reconstruct runs, each destroy mechanism is applied exactly twice. Therefore, it is redundant to deploy certain weights to control the utilization rate of a certain operator as it normally happens in ALNS.

MMAS 3 Iterated consists of two main loops with 500 iterations each which guarantees a total sum of 1000 iterations to make the results comparable to the outcomes of the previous algorithms. The first loop is similar to the *MMAS Basis* algorithm and has the main task to generate an initial solution which is later used by the ALNS. This neighborhood search is implemented in the second loop which takes the initial solution and tries to improve it by destroying and reconstructing combined with a 2-opt local search as usual. In Figure 19 the pseudo-code of this algorithm is represented.

```
For 500 iterations
      For all 5 ants do
            Generate random number and assign facilities to locations
            following to the Roulette Wheel procedure in order to get s
            Calculate f(s)
            Do
                  2-opt local search to obtain s´ and f(s´)
                  If (f(s´) < f(s))
                        Set s = s´ and f(s) = f(s´)
                  EndIf
            While(improvement == true)
            If (s < ib)
                  Set ib = s
            EndIf
      EndFor
      If (ib < gb)
            Set gb = ib
            Update pheromone trail limits
      EndIf
      Update the pheromone trails
EndFor
For 500 iterations
      For 6 runs
            DESTRUCT
                  Remove (n/6) components from s
                  If (run==1||run==2) use MMAS Random Removal
                  If (run==3||run==4) use MMAS Product Highest
                  If (run==5||run==6) use MMAS Product Lowest

            RECONSTRUCT
                  Reconstruct s according to Roulette Wheel
            Do
                  2-opt local search to obtain s´ and f(s´)
                  If (f(s´) < f(s))
                        Set s = s´ and f(s) = f(s´)
                  EndIf
            While(improvement == true)
      EndFor
EndFor
```

Figure 19: Pseudo-code for the MMAS 3 Iterated

7. Computational Results

In this chapter the experimental results obtained by the test runs of *MMAS Basis*, *MMAS Random Removal*, *MMAS Product Removal Highest*, MMAS *Product Removal Lowest* and *MMAS 3 Iterated* are presented. The employed laptop was a Sony Vaio VGN-NS21M (Intel (R) Pentium (R) Dual CPU T3400

@ 2.16 GHz, 3 GB RAM) with Windows Vista Home Premium as operating system.

The processed data files were taken from the QAPLIB[76] and comprise various instance sizes (from n = 12 to n = 50). All parameters of the algorithms were fixed to the same values throughout the whole experiment:

- Total number of iterations = 1000
- Number of ants per iteration = 5
- Influence factor of pheromone trails α = 0
- Resistance of pheromone trails ρ = 0,8
- p_{best} = 0,005
- Number of destroy/reconstruct runs = 6

All results which are included and compared in the following tables stand for the mean of 5 independent runs of each algorithm. In Tables 1-8 we summarize the most important results for each problem instance which were obtained by the quoted algorithm. The term ***avg value*** denotes the average solution value, ***Best%*** shows the percentage deviation from the best known solution value, ***Runtime*** is the mean of all five attended runtimes and ***Runtime$_{best}$*** denotes the average point in time when the algorithm has already generated the global best solution.

Tai20a				
Best known solution: 703482				
Algorithm	avg value	Best%	Runtime	Runtime$_{best}$
MMAS Basis	713818,8	1,47%	1,1238	0,1892
MMAS Random Removal	707981,6	0,64%	9,6118	2,3910
MMAS Product Removal Highest	707255,6	0,54%	11,9530	0,7090
MMAS Product Removal Lowest	706830,0	0,48%	11,2584	1,4500
MMAS 3 Iterated	709905,6	0,91%	1,5984	0,6336

Table 1: Experimental results for tai20a

[76] http://www.opt.math.tu-graz.ac.at/qaplib/

For ***tai20a*** the best performing algorithm regarding its average solution value seems to be *MMAS Product Removal Lowest* (followed by *MMAS Product Removal Highest*) whereas *MMAS Basis* is the fastest in finding the global best solution.

Tai25a				
Best known solution: 1167256				
Algorithm	avg value	Best%	Runtime	$Runtime_{best}$
MMAS Basis	1186522,0	1,65%	2,2474	0,8226
MMAS Random Removal	1176860,0	0,82%	22,9786	3,9768
MMAS Product Removal Highest	1178170,0	0,94%	27,4612	3,3894
MMAS Product Removal Lowest	1181646,0	1,23%	25,8960	1,7016
MMAS 3 Iterated	1180998,0	1,18%	3,3726	1,3102

Table 2: Experimental results for tai25a

Tai30a				
Best known solution: 1818146				
Algorithm	avg value	Best%	Runtime	$Runtime_{best}$
MMAS Basis	1848306,0	1,66%	4,2778	1,7168
MMAS Random Removal	1837754,0	1,08%	42,1664	12,8358
MMAS Product Removal Highest	1838892,0	1,14%	47,5116	5,8688
MMAS Product Removal Lowest	1840376,0	1,22%	49,6724	4,6456
MMAS 3 Iterated	1848316,0	1,66%	7,1198	2,8382

Table 3: Experimental results for tai30a

For both ***tai25a*** and ***tai30a*** on average *MMAS Random Removal* generates the best solutions and again *MMAS Basis* is the fastest in doing so. *MMAS 3 Iterated* and *MMAS Basis* work very similar for ***tai30a*** with the same percentage deviation of 1,66% although *MMAS 3 Iterated* needs more time to pass through the given 1000 iterations. In addition it is mentionable that the average finding of the global best solution of *MMAS Random Removal* exceeds the results of the others by far.

Tai35a				
Best known solution: 2422002				
Algorithm	avg value	Best%	Runtime	$Runtime_{best}$
MMAS Basis	2459866,0	1,56%	6,7744	3,4618
MMAS Random Removal	2457496,0	1,47%	81,5020	28,5492
MMAS Product Removal Highest	2461998,0	1,65%	70,6380	7,1256
MMAS Product Removal Lowest	2452816,0	1,27%	75,1002	8,3894
MMAS 3 Iterated	2457294,0	1,46%	10,5102	3,6192

Table 4: Experimental results for tai35a

For ***tai35a*** again *MMAS Product Removal Lowest* performs best and in *MMAS Random Removal* the artificial ants need three times as much runtime to find the global best solution.

Tai40a				
Best known solution: 3139370				
Algorithm	avg value	Best%	Runtime	$Runtime_{best}$
MMAS Basis	3200836,0	1,96%	10,3558	4,9514
MMAS Random Removal	3171490,0	1,02%	157,4742	39,1266
MMAS Product Removal Highest	3183276,0	1,40%	131,1442	31,4480
MMAS Product Removal Lowest	3193490,0	1,72%	138,1652	16,9546
MMAS 3 Iterated	3192152,0	1,68%	20,2120	10,1150

Table 5: Experimental results for tai40a

The problem instance ***tai40a*** provides the worst results for *MMAS Product Removal Lowest* and is more or less the only outlier for this algorithm. Despite the fact that *MMAS Random Removal* still needs longest for the best solution, it performs absolutely best for ***tai40a***.

Tai50a				
Best known solution: 4941410				
Algorithm	avg value	Best%	Runtime	$Runtime_{best}$
MMAS Basis	5045378,0	2,10%	24,7618	14,0986
MMAS Random Removal	5019180,0	1,57%	481,2132	156,8340
MMAS Product Removal Highest	5018722,0	1,56%	352,6580	108,5374
MMAS Product Removal Lowest	4997166,0	1,13%	373,7104	74,6542
MMAS 3 Iterated	5025534,0	1,70%	48,4134	32,4442

Table 6: Experimental results for tai50a

Tai50a provides the worst results for *MMAS Basis*, *MMAS Random Removal* and *MMAS 3 Iterated*. In case of *MMAS Basis* this may happen due to a lack of an extensive local search procedure. Again *MMAS Product Removal Lowest* performs best and has an average runtime which lies slightly beneath the total mean. Very interesting is the fact that although *MMAS Product Removal Highest* and *MMAS Product Removal Lowest* generate solutions in a very similar way, *MMAS Product Removal Lowest* needs remarkable less time for the finding of the global best solution.

Scr12				
Best known solution: 31410				
Algorithm	avg value	Best%	Runtime	$Runtime_{best}$
MMAS Basis	31410,0	0,00%	0,3354	0,0184
MMAS Random Removal	31410,0	0,00%	3,6120	0,0210
MMAS Product Removal Highest	31410,0	0,00%	2,3244	0,0146
MMAS Product Removal Lowest	31410,0	0,00%	2,4188	0,0146
MMAS 3 Iterated	31410,0	0,00%	0,4144	0,0050

Table 7: Experimental results for scr12

Scr12 is the only problem instance for which the best known solution is found in every run of each algorithm whereas *MMAS Basis* is the fastest one.

Tho30				
Best known solution: 149936				
Algorithm	avg value	Best%	Runtime	$Runtime_{best}$
MMAS Basis	150372,8	0,29%	6,4364	3,7190
MMAS Random Removal	150181,6	0,16%	197,9978	28,3628
MMAS Product Removal Highest	150073,6	0,09%	94,8862	19,1026
MMAS Product Removal Lowest	150167,2	0,15%	111,5210	8,6936
MMAS 3 Iterated	150207,6	0,18%	15,2622	4,6800

Table 8: Experimental results for tho30

For ***Tho30*** the best known solution can be generated in almost every run (independent from the applied algorithm) and it is the only problem instance for which *MMAS Product Removal Highest* performs best.

In Table 9 we give an overview of all percentage deviations for all problem instances – best results are indicated in italic face. *MMAS Basis* and *MMAS 3 Iterated* are the two worst performing algorithms for all instances with ***scr12*** being the only exception. After all *MMAS Random Removal* seems to be the best algorithm followed by *MMAS Product Removal Highest* and *MMAS Product Removal Lowest* which differ by 0,01%. Again the algorithm to come in last is *MMAS Basis* which is never able to generate the best solution – this is the best proof that the main idea to extend the MMAS Basis by a more precise local search procedure leads to better performing algorithms.

Problem instance	MMAS Basis	MMAS Random	MMAS Product Rem. Highest	MMAS Product Rem. Lowest	MMAS 3 Iterated
tai 20a	1,47%	0,64%	0,54%	***0,48%***	0,91%
tai 25a	1,65%	***0,82%***	0,94%	1,23%	1,18%
tai 30a	1,66%	***1,08%***	1,14%	1,22%	1,66%
tai 35a	1,56%	1,47%	1,65%	***1,27%***	1,46%
tai 40a	1,96%	***1,02%***	1,40%	1,72%	1,68%
tai 50a	2,10%	1,57%	1,56%	***1,13%***	1,70%
scr12	***0,00%***	***0,00%***	***0,00%***	***0,00%***	***0,00%***
tho30	0,29%	0,16%	***0,09%***	0,15%	0,18%
Mean:	**1,34%**	**0,85%**	**0,91%**	**0,90%**	**1,10%**

Table 9: Comparison of percentage deviations

In Table 10 the total runtimes of *MMAS Random Removal* and *MMAS Product Removal Lowest* are checked against each other. As mentioned before, these two algorithms perform absolutely best and although *MMAS Random* provides slightly better solution values, *MMAS Product Removal Lowest* needs less time to fulfill the 1000 iterations (on average 26 seconds faster).

Problem instance	MMAS Random	MMAS Product Rem. Lowest
tai 20a	***9,6118***	11,2584
tai 25a	***22,9786***	25,896
tai 30a	***42,1664***	49,6724
tai 35a	81,502	***75,1002***
tai 40a	157,4742	***138,1652***
tai 50a	481,2132	***373,7104***
scr12	3,612	***2,4188***
tho30	197,9978	***111,521***
Mean:	**124,5695**	**98,4678**

Table 10: Comparison of total runtime

By looking at Table 11 it can be observed that *MMAS Product Removal Lowest* is for nearly every problem instance the fastest algorithm regarding the point in time when the best solution is first available throughout the total runtime. The comparison of these algorithms, which all implement the Iterated Ants idea, shows that the artificial ants in *MMAS Random Removal* need longest to generate the best solution. After all, this algorithm still finds the best solution values on average which makes the discussion concerning the runtime more or less dispensable.

Problem instance	MMAS Random	MMAS Product Rem. Highest	MMAS Product Rem. Lowest
tai 20a	2,3910	***0,709***	1,45
tai 25a	3,9768	3,3894	***1,7016***
tai 30a	12,8358	5,8688	***4,6456***
tai 35a	28,5492	***7,1256***	8,3894
tai 40a	39,1266	31,448	***16,9546***
tai 50a	156,8340	108,5374	***74,6542***
scr12	0,0210	***0,0146***	***0,0146***
tho30	28,3628	19,1026	***8,6936***
Mean:	**34,0122**	**22,024425**	**14,56295**

Table 11: Comparison of time needed to find best solution

8. Conclusion

In this diploma thesis five MMAS algorithms, which differ from each other by the implemented local search procedures, have been proposed. Unfortunately, they weren´t able to improve the best known solutions in the tested instances, but by looking at the results some very interesting findings can be observed. First of all, we can say that it has absolutely been proofed that by enhancing the *MMAS Basis* algorithm by an efficient local search we can definitely improve the solution quality. In doing so, we can say that in our case the random removal of solution components worked slightly better than the destroy methods of *MMAS Product Removal Highest* and *MMAS Product Removal Lowest*, although these algorithms have the focus on incorporating important information like the distance and flow matrices. The worst results were obtained by the *MMAS 3 Iterated* which is more or less a modification of an Adaptive Large Neighborhood Search.

After all there´s one conclusion which can be drawn from these experimental results: the research field of Ant Colony Optimization for the QAP seems to be exploited very good because no improved results were obtained. Nevertheless, I think that there still exist lots of basic approaches which can still be pursued.

References

Ahuja, R.K., Ergun, Ö., Orlin, J.B., Punnen, A.P.: *A survey of very large-scale neighborhood search techniques*, Discrete Applied Mathematics 123, 75-102, 2002

Askin, R.G., Standridge, C.R.: *Modeling & Analysis Of Manufacturing Systems*, John Wiley & Sons, 1993

Burkard, R.E., Offermann, J.: *Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme*, Zeitschrift für Operations Research 21, B121-B132, 1977

Denebourg, J.L., Aron, S., Goss, S., Pasteels, J.M.: *The self-organizing exploratory pattern of the Argentine ant*, Journal of Insect Behavior 3 (2), 159-168, 1990

Dickey, J.W., Hopkins, J.W.: *Campus building arrangement using TOPAZ*, Transportation Science 6, 59-68, 1972

Dorigo, M., Stützle, T.: *Ant Colony Optimization: Overview and Recent Advances*, Technical Report TR/IRIDIA/2009-013, IRIDIA, Université Libre de Bruxelles, http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2009-013r001.pdf, 2009

Dorigo, M., Stützle, T.: *Ant Colony Optimization*, MIT Press, Cambridge, Massachusetts, London, England, 2004

Dorigo, M., Stützle, T.: *The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*, in: F. Glover, G.A. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, Dordrecht, London, 251-285, 2003

Dorigo, M., Di Caro, G.: *The Ant Colony Optimization Meta-Heuristic*, in: D. Corne et al. (Eds.), New Ideas in Optimization, McGraw-Hill, 11-32, 1999

Dorigo, M., Birattari, M., Stützle, T.: *Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique*, IEEE Computational Intelligence Magazine 1 (4), 28-39, 2006

Dorigo, M., Maniezzo, V., Colorni, A.: *The Ant System: Optimization by a colony of cooperating agents*, IEEE Transactions on Systems, Man, and Cybernetics-Part B 26 (1), 29-41, 1996

Elshafei, A.N.: *Hospital layout as a quadratic assignment problem*, Operations research Quarterly 28, 167-179, 1977

Gambardella, L.M., Taillard, E.D., Dorigo, M.: *Ant colonies for the quadratic assignment problem*, Journal of the Operational Research Society 50 (2), 167-176, 1999

Garnier, S., Gautrais, J., Theraulaz, G.: The *biological principles of swarm intelligence*, in Swarm Intelligence 1 (1), 3-31, 2007

Glover, F., Kochenberger, G.A.: *Handbook of Metaheuristics*, Kluwer Academic Publishers, Boston, Dordrecht, London, 2003

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

Ji, P., Wu, Y., Liu, H.: *A Solution Method for the Quadratic Assignment problem (QAP)*, The Sixth International Symposium on Operations Research and Its Applications (ISORA'06), Xinjiang, China, August 8-12, 106–117, 2006

Koopmans, T.C., Beckmann, M.J.: *Assignment Problems and the Location of Economic Activities*, Econometrica 25, 53-76, 1957

Maniezzo, V., Colorni, A., Dorigo, M.: *The Ant System applied to the Quadratic Assignment Problem*, Technical Report IRIDIA/94/28, Université Libre de Bruxelles, Belgium, 1994

Mullen, R.J., Monekosso, D., Barman, S., Remagnino, P.: *A review of ant algorithms*, Expert Systems with Applications 36 (6), 9608-9617, 2009

Pisinger, D., Ropke, S.: *Large neighborhood search*, in: J. Potvin, M. Gendreau (Eds.), Handbook of Metaheuristics, Springer-Verlag, 399-419, 2010

Ramkumar, A.S., Ponnambalam, S.G., Jawahar, N.: *A new iterated last local search heuristic for solving QAP formulation in facility layout design*, Robotics and Computer-Integrated Manufacturing 25 (3), 620-629, 2009

Ropke, S., Pisinger, D.: *An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows*, Transportation Science 40 (4), 455-472, 2006

Ruiz, R., Stützle, T.: *An Iterated Greedy Heuristic for the Sequence Dependent Setup Times Flowshop Problem with Makespan and Weighted Tardiness Objectives*, European Journal of Operational Research 187 (3), 1143-1159, 2008

Sahni, S., Gonzales, T.: *P-complete approximization problems*, Journal of the Association for Computing Machinery 23 (3), 555-565, 1976

Shaw, P.: *Using constraint programming and local search methods to solve vehicle routing problems*, in: CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming), Volume 1520 of LNCS, 417-431, 1998

Stützle, T., Hoos, H.H.: *MAX-MIN Ant System*, Future Generation Computer Systems 16 (8), 889-914, 2000

Wiesemann, W., Stützle, T.: *Iterated Ants: An Experimental Study for the Quadratic Assignment Problem*, in: M. Dorigo et al. (Eds.), Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006, Proceedings, Volume 4150 of LNCS, Springer-Verlag, Berlin, 179-190, 2006

Internet-resources

The QAPLIB: http://www.opt.math.tu-graz.ac.at/qaplib/
Called up 13.01.2011

The QAPLIB introduction: http://www.opt.math.tu-graz.ac.at/qaplib/#intro
Called up 22.03.2011

Roulette Wheel: http://en.wikipedia.org/wiki/Fitness_proportionate_selection
Called up on 22.03.2011

2-opt local search method: http://en.wikipedia.org/wiki/2-opt
Called up on 22.03.2011

Figure 1:
http://www.scholarpedia.org/wiki/images/9/97/ SameLengthDoubleBridge.png
Called up 27.01.2011

Figure 2:
http://www.scholarpedia.org/article/File:DiffLengthDoubleBridge.png
Called up 27.01.2011

Figure 5:
http://www.i-cherubini.it/mauro/blog/wp-content/uploads/2007/08/images/Dry_TSP_experiment.png
Called up 06.01.2011

Abstract

This diploma thesis deals with the scientific research area of Ant Colony Optimization algorithms and applies them to the Quadratic Assignment Problem.

The central aim of the Quadratic Assignment Problems is to find an optimal allocation of a certain number of facilities to the equal number of possible locations in order to minimize the overall costs. This theoretical formulation can be passed on real life problems in a straightforward way. One of the best examples is the challenge each company has to deal with when opening a new production site. The production cost can be kept to a minimum as long as the used machines are cleverly arranged to their locations so that the overall sum of the products between material flows and distances comes to an economical appropriate value.

In the theoretical part of this thesis some of the most important ant algorithms like MMAS and HAS-QAP are discussed and it is shown how the additional implementation of an effective local search method can improve the solution quality.

The practical part of this thesis tries to enhance a basic MMAS algorithm by implementing additional local search methods based on the ideas of Iterated Ants and Adaptive Large Neighborhood Search. Therefore five different algorithms have been implemented in C++: *MMAS Basis*, *MMAS Random Removal*, *MMAS Product Removal Highest*, *MMAS Product Removal Lowest and MMAS 3 Iterated*. At the end the generated results are discussed and the solution qualities of the individual algorithms are compared among themselves.

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit dem wissenschaftlichen Forschungsgebiet der Ameisenoptimierung und deren Algorithmen und wendet diese auf das Quadratische Zuordnungsproblem an.

Das Hauptziel des Quadratischen Zuordnungsproblems besteht darin eine geeignete Zuordnung von einer gewissen Anzahl an Funktionen zu der gleichen Anzahl an möglichen Plätzen zu finden, um die Gesamtkosten zu minimieren. Dieser theoretische Ansatz kann auf direktem Weg auf Probleme der Wirklichkeit übertragen werden. Eines der besten Beispiele hierfür ist die Herausforderung, welcher ein Unternehmen sich stellen muss wenn es eine neue Produktionsstätte eröffnet. Die Produktionskosten können solang auf einem minimalen Level gehalten werden wie die verwendeten Maschinen intelligent auf ihren Standorten angeordnet werden, sodass die Gesamtsumme der Produkte zwischen Materialflüsse und Distanzen einen ökonomisch angemessenen Wert ergibt.

Im theoretischen Teil dieser Arbeit werden einige der wichtigsten Ameisenalgorithmen wie MMAS und HAS-QAP diskutiert und es wird gezeigt wie die zusätzliche Implementierung von effektiven Local Search Methoden die Lösungsqualität verbessern kann.

Der praktische Teil dieser Arbeit versucht einen grundlegenden MMAS Algorithmus um zusätzliche Local Search Methoden, welche auf den Ideen von Iterated Ants und Adaptive Large Neighborhood Search basieren, zu erweitern und so zu verbessern. Hierfür wurden fünf verschiedene Algorithmen in C++ implementiert: *MMAS Basis*, *MMAS Random Removal*, *MMAS Product Removal Highest*, *MMAS Product Removal Lowest and MMAS 3 Iterated*. Zuletzt werden die generierten Ergebnisse diskutiert und die Lösungsqualitäten der einzelnen Algorithmen untereinander verglichen.

Curriculum Vitae

Persönliche Daten

Name:	Stephanie Richter
Geburtsdatum:	17.Juni.1986
Geburtsort:	Linz, Oberösterreich
Staatsbürgerschaft:	Österreich
Familienstand:	ledig

Ausbildung

10/2004 – 03/2011 **Universität Wien**
Betriebswirtschaftliches Zentrum Wien (BWZ)
Studium der Internationalen Betriebswirtschaftslehre
Spezialisierungen:

- Produktionsmanagement
- Internationale Unternehmensführung

09/1996 – 06/2004 **BRG Fadingerstrasse**, Linz

09/1992 – 06/1996 **VS 43 Stadlerschule**, Linz