Aalto University
School of Science
Degree programme in Engineering Physics and Mathematics

Implementation of Selected Metaheuristics to the Travelling Salesman Problem

Bachelor's thesis
1.11.2014

Jari Hast

Aalto University

AALTO UNIVERSITY SCHOOL OF SCIENCE PO Box 11000, FI-00076 AALTO http://www.aalto.fi	ABSTRACT OF THE BACHELOR'S THESIS
Author: Jari Hast	
Title: Implementation of Selected Metaheuristics to the Travelling Salesman Problem	
Degree programme: Degree programme in Engineering Physics and Mathematics	
Major subject: Systems science	Major subject code: F3010
Supervisor: Prof. Harri Ehtamo Instructor: Prof. Harri Ehtamo	

Abstract:

Metaheuristics are approximate optimization algorithm frameworks used to tackle hard optimization problems. In recent years they have attracted popularity due to their efficiency and flexibility. A metaheuristics describes the problem solving process on an abstract level and thus doesn't dictate the specific implementation. The metaheuristics are often derived from natural processes.

This thesis gives an introduction to metaheuristics by discussing the relevant concepts surrounding the topic and by giving examples of implementations to the travelling salesman problem. The metaheuristics selected for this task are Greedy algorithm, Ant colony optimization, Artificial bee colony and a genetic algorithm.

All implementations are documented and programmed with MATLAB. Pseudo codes of the algorithms are also presented. The algorithms' performance is then tested on two test instances of the travelling salesman problem. Lastly the results are compared and discussed.

Date: 1.11.2014	Language: English	Number of pages: 25+4

Keywords: metaheuristics, travelling salesman problem, combinatorial optimization, ant colony optimization, artificial bee colony, genetic algorithm

Abbreviations

ABC: Artificial bee colony

ACO: Ant colony optimization

GA: Genetic algorithm

TSP: Travelling salesman problem

Contents

1 Introduction

Metaheuristics are approximate optimization algorithm frameworks that are often inspired by natural processes. A metaheuristic can be described as a high-level strategy or as a general algorithmic framework that can be applied to a variety of problems with only a few modifications. They are commonly used when faced by a complex problem that cannot be solved by exact algorithms in reasonable time.

The term metaheuristic was first introduced in [Glover, 1986]. It derives from the composition of two Greek words. Heuristic derives from the verb heuriskein which means "to find", while the suffix meta means "beyond, in an upper level" [Blum and Roli, 2003].

A great interest has been devoted to metaheuristics in the last thirty years. They are widely used to solve complex problems in industry and services, in areas ranging from finance to production management and engineering [Boussaïd et al., 2013].

The travelling salesman problem on the other hand is probably among the most intensively studied problems in optimization. Its applications include e.g. computer wiring, vehicle routing, data array clustering and job-shop scheduling as shown in [Lenstra and Kan, 1975]. Since the TSP is classified as an NP-hard problem, it serves as a good example for application of metaheuristics even though for solving it there are powerfull exact algorithms available, such as the Concorde TSP solver [Applegate et al., 2006].

The objective of this thesis is to apply four metaheuristics to the TSP. Namely these are ant colony optimization algorithm, artificial bee colony algorithm, a greedy algorithm and a genetic algorithm. All programming is done with MATLAB. Problem instances are acquired from TSPLIB [Reinelt, 1991], where problem instances of various size can be obtained. A comparison on the quality of the results and the differences in implementation will be presented. Also, the classification and different characteristics of the metaheuristics as well as important concepts regarding the design of the algorithms are discussed. Eventhough metaheuristics can also be used to solve continuous problems, this thesis focuses on combinatorial optimization aspect of optimization.

2 Theoretical background

2.1 On the definition of metaheuristics

There are several definitions of metaheuristics present in the literature. [Osman and Laporte, 1996] claims that:

> "A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."

An almost identical definition is made in [Talbi, 2009]:

> "Metaheuristics search methods can be defined as upper lever general general methodologies (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems"

[Voss et al., 1999] defines metaheuristics as:

> "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method."

To summarize, metaheuristics can be seen as higher level, possibly abstract, strategies that guide the problem solving process, which can be either simple, complicated or anything in between. Metaheuristics aim to efficiently exploit and explore the search space for near-optimal solutions.

2.2 Taxonomy of metaheuristics

Optimization algorithms can be divided into exact methods and approximate methods. Exact methods are guaranteed to obtain optimal solutions, but for complex problems they are usually non-polynomial-time algorithms. Approximate methods on the other hand generate high-quality solutions in reasonable running time, but don't guarantee finding a globally optimal solution.

The class of approximate methods can be further divided into approximation algorithms and heuristic algorithms. With approximation algorithms, provable solution quality and provable run-time bounds are obtained. Heuristics on the other hand generally find "good" solutions but cannot guarantee similar bounds.

In the class of heuristics, two families of methods may be distinguished: specific heuristics and metaheuristics. Specific heuristics are designed to solve a single spesific problem or instance whereas metaheuristics can be practically tailored to solve any kind of problem. They can be seen as higher level strategies that can be used to generate methods for solving a specific optimization problem. [Talbi, 2009]

2.3 Classification for metaheuristics

Metaheuristics can be classified according to several criteria. Firstly, metaheuristics can be numbered among *nature inspired* or *non-nature inspired*. From the metaheuristics discussed in this thesis, the greedy algorithm is the only one belonging to the non-nature inspired as for example the ant colony optimization algorithm mimics the food foraging behavior of ants. This classification, however, can be problematic. There are, for instance, recently proposed hybrid algorithms that may fit to both of these classes or neither depending on one's perspective.

Another way of classification is *population-based search vs. single-solution based search*. Single-solution based algorithms, such as the greedy algorithm discussed in this thesis, work on single solutions whereas population-based methods manipulate a given number of solutions at a time. Single-solution based metaheuristics tend to excel in exploitation of the local search space while population-based metaheuristics allow better exploration of the search space.

Perhaps, the most straight forward way of classification is *deterministic vs. stochastic*. In stochastic methods one starting point may lead to different solutions since random numbers are used in the process. In deterministic metaheuristics, on the contrary, a specific initial solution always leads to the same final solution. In this thesis, the only deterministic algorithm discussed is again the greedy algorithm.

One of most important ways of classification is *memory usage vs. memoryless methods*. For instance, some local search metaheuristics perform Markov-processes as they base their next moves solely on the information regarding their current state of the search process. These are called memoryless methods. Memory usage in metaheuristics can usually be divided into short-term and long-term memory. Short-term memory utilizes recently gathered information of the search process. Long-term memory on the other hand takes advantage of the information accumulated during the whole search. [Talbi, 2009] [Blum and Roli, 2003]

2.4 Important concepts for metaheuristics

Each metaheuristic implementation process starts with the definiton of *representation* and *objective function*. That is, how the solution is encoded in the algorithm and how we measure the goodness of a solution [Talbi, 2009]. For example, the TSP (defined in chapter 2.5) can be encoded in several ways. One way is to simply give indices to each town and store the permutation of these indices, which then represents a route. Another way would be to store every city with the two cities that are connected to it. In this thesis, the selection of the objective function is unambigous but that is not always the case.

Every implementation of metaheuristics must balance between *diversification* and *intensification*. Diversification is the process of exploring more of the search space or generating solution candidates that differ from the ones already obtained in the course of the algorithm. Intensification on the other hand means improving the best candidates or exploring their neighborhoods in the hope of finding even better solutions. Typically diversification focused algorithms explore too large section of the search space without being able to find satisfactory solutions where as intensification focused heuristics tend to get stuck in the local optima. [Talbi, 2009] [Blum and Roli, 2003]

The definition of *neighborhood* often becomes relevant during the implementation of a metaheuristic especially if it contains any local search methods. The concept of neighborhood defines solutions that are, by some measure, close to each other. Thus it is closely connected to the definition of distance in the search space. For example, in the Travelling salesman problem a neighboring solution to our current one could be a route where two cities have switched places (see the definition of TSP in 2.5). In this case, a local search is called a city swap. Even though it might seem the most intuitive way to describe a neighborhood, city swap is not an effective method compared to other local search techniques such as 2-opt which is discussed in chapter 3.3.3. [Talbi, 2009]

Oftentimes the performance of a metaheuristic can be altered by changing how the *initial solutions are generated.* If a heuristic is given good enough initial solutions, it might converge into a solution in less time. However, certain initial input can also harm the performance of the algorithm. For example, too little diversified initial solutions can lead to inefficient exploration of the search space. Initial solutions can be generated randomly or even algorithmically.

2.5 Travelling salesman problem

The travelling salesman problem can be defined as follows. The salesman is given a set of cities and distances from any city to another. The task is then to visit every city exactly once and return to the starting city while minimizing the total distance travelled. In this thesis, the problems instances are symmetric, wherefore the distace from city i to city j is equal to the distance from j to i.

Formally this can be denoted as a graph problem. Let $G = (C, A)$ be a graph where $C = \{c_1, c_2, ..., c_n\}$ denotes a set of nodes or cities and $A = \{(i, j)|\forall i, j \in C, i \neq j\}$ arcs between nodes. Also let $d_{i,j}$ denote the weights associated with each arc. All weights or distances can then be expressed as a $n \times n$ matrix D. The objective is now to construct a Hamiltonian cycle $\pi = \{\pi(1), \pi(2), ..., \pi(n)\}$ with minimal total weight or

$$\min \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}. \tag{1}$$

The search space of a TSP problem is of size $(n-1)!$, when the starting city is fixed. This quickly renders brute-force search (going through all the possible paths one by one) as an infeasible approach when the size of the problem instances grow. However, until modern day, numerous other ways of solving the TSP have been devised, including but not limited to dynamic programming [Bellman, 1962], integer programming with cutting planes and branch-and-X algorithms [Padberg and Rinaldi, 1991] and specific heuristics [Helsgaun, 2000].

3 Overview and implementation of the selected metaheuristics

3.1 Greedy algorithm

Perhaps the simplest metaheuristics discussed in this thesis is the greedy algorithm. The main idea of this metaheuristic is to start with an empty solution and incrementally add values to decision variables one variable at a time. This is repeated until all variables have a value, that is to say, until a complete solution has been achieved. The variables and the values assigned to them are decided by a local heuristics of choice, which tries to minize (or maximize) the objective function at every step. In this case, the nearest neighbor approach is selected. In TSP this translates into the following: the next city to be visited is the one with smallest euclidean distance to the current city.

Let $C = c_1, c_2, ..., c_n$ represent the set of cities of a TSP instance. If the distances between the cities are known, the pseudo code for the greedy algorithm can be formulated as in listing 1.

Listing 1: Pseudo code of the implemented greedy algorithm

```
Initialize C to empty list
Select a starting city and add it to C
for i=2...n
  Select neighbor nearest to last visited city and add it to C
end for
Return C
```

As seen from the pseudo code, the implementation of the greedy algorithm is relatively simple. The starting city can be selected randomly. However, since the algorithm is not expensive in terms of computing, every city can easily be tried in problem instances of less than one thousand cities.

3.2 Ant colony optimization

3.2.1 Overview

Ant colony optimization metaheuristics mimics the food foraging behavior of ants. It was introduced in [Dorigo and Di Caro, 1999]. The underlying logic of the metaheuristic was first implemented on the Traveling salesman problem a few of years earlier by the same authors.

Ant colony optimization is a population based metaheuristic where simple agents called ants communicate indirectly with each other via pheromone trails. Ants deposit pheromones of different concentration depending on the quality of route they have traversed. They also use the concentration of pheromones to determine their paths in the search space. The trick is that because pheromones evaporate over time, the shortest routes are covered most and thus intensified.

The ACO framework consists of three phases. In the first phase the ants try to find a feasible path or a solution from the search space that minimizes the cost function. This phase is called the management of ants' activity. The other two phases are pheromone evaporation and daemon action, latter being optional. In the daemon action phase, global actions, which single ants are unable to do, can be executed. For example, the best route this far can be intensified or new traversal rules can be added. These phases can be carried out in any order and there are many possible variations in each of them. For example, the pheromones can be chosen to be updated after every step of the ant or in the end when every ant has completed their routes. [Dorigo and Stützle, 2003]

3.2.2 Implementation

Ants are first initialized into random starting cities. Until each ant has visited every city, ant k chooses to travel from city i to an unvisited city j with a probability of

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} \qquad \text{if } j \in N_i^k, \tag{2}$$

where $\eta_{ij} = 1/d_{ij}$ is in the most general sense the a priori information of the move and here, as in most implementations, it is simply the inverse of distance. N_i^k is the set of unvisited cities of ant k when it has traversed to city i. Function $\tau_{ij}(t)$ gives the amount of pheromone along the route from i to j at a time t. Coefficients α and β define the relative weight of the a priori information and the pheromone trails, respectively. For example, if α equals zero then only the a priori information affects the probability.

When every ant has visited every city and thus achieved a complete solution the algorithm moves into the pheromone evaporation phase. The pheromones are updated according to the formula

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \qquad \forall\,(i,j)\,. \tag{3}$$

Here ρ is the parameter which determines how quickly the pheromones evaporate. $\Delta\tau_{ij}^{k}(t)$ on the other hand is the added pheromone from ant k. There are many possible variations on how to calculate this, but the most basic version, which is implemented in this thesis as well, is of the form

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} 1/L^{k}(t) & \text{if ant has traveled from city } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}\,, \tag{4}$$

where $L^{k}(t)$ is the total length of the complete path of kth ant. So the shorter the length of the tour is, the more pheromone gets deposited and that path is more probably traversed in the next iterations. [Dorigo and Stützle, 2003]

The whole algorithm in this thesis is implemented as in listing 2.

Listing 2: Pseudo code of the implemented Ant colony optimization

```
Initialize ants with empty routes
while (StoppingCriteria)
  Management of ants' activity i.e. build complete path for each ant
  Pheromone evaporation and update
end while
Return the best ant
```

The stopping criteria used in this thesis is simply a maximum limit for the iterations.

3.3 Artificial bee colony optimization

3.3.1 Overview

Artificial bee colony optimization is also a nature inspired population based metaheuristic. As the name suggests, the metaheuristic is loosely coupled with bees' behaviour in searching for a nest site, marriage in the colony or food foraging. It features concepts borrowed from nature such as waggle dance, nectar exploration and division of labour. Algorithms based on bee colony's have been been used in a variety of problems of both continuous and combinatorial nature. [Talbi, 2009]

Artificial bee colony optimization is based on the bees' food foraging behaviour where the insects try to find the shortest way to their food source. In this thesis' implementation, the bee colony is divided into three different categories of bees. There are scout bees, who are responsible for diversification i.e. they explore new areas in the search space. Then there are foragers who can be either active (employed) or inactive (unemployed). Active bees are in charge of intensification i.e. they perform local search around promising neighborhoods for a given amount of time until they retire to inactive bees. Inactive bees on the other hand wait in the nest for actives bees' information about prospective food sources.

The information about about the food sources is transmitted via "waggle dance". This term is present in almost every paper that discusses bee colony based metaheuristics although its implementation varies. In summary, the bees in the nature are able to communicate the direction, quality and the distance of a food source with a series of movements dubbed "waggle dance". In this thesis, the implementation of the dance is rather latent as described below.

3.3.2 Implementation

In practice, a bee holds information of a route and its length. In some cases an implementation can be said to contain a hive object. It contains global information of the algorithm such as the shortest route and the number of each bee type. In this thesis, the hive is not a concrete object in the code. However, the same information present in separate variables.

At the beginning of the algorithm, the bees are initialized to probabilistic greedy trails. That is, bee k will choose to travel from city i to city j at a time t with the probability of

$$p_{ij}^{k}(t) = \frac{[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\eta_{il}]^{\beta}} \qquad \text{if } j \in N_i^k, \tag{5}$$

where the terms are the same as in equation (2) with the exceptions that this equation concerns bees. Notice that the only difference between this equation and (2) is that the pheromone term is missing.

After the initialization of the bees, the main loop of the implementation is split into two parts. First, the active and inactive foragers are handled. Active bees have two relevant limits concerning their local search: the number of searches they perform during one iteration of the algorithm and the number of searches before they retire. If the local search yields the globally best route it is saved in the global memory and naturally to the ant's memory also.

Once an employed forager has retired, an unemployed forager is activated. This means that the population and the proportions of the different bee types remain constant through the whole algorithm. An unemployed bee can see the routes of every scout bee and chooses its local search area according to a probabilistic greedy rule much like the one shown in equation (5). That is the shorter the route is the more probably it is selected. This can be expressed as follows. The probability $p_k(t)$ of selecting the route of bee k at a time t is

$$p_k(t) = \frac{[\rho_k(t)]^{-\beta}}{\sum_{i=1}^{N}[\rho_i(t)]^{-\beta}}, \tag{6}$$

where $\rho_k(t)$ is the length of the route of bee k at a time t. As before, the coefficient β determines the significance of the differences in route lengths. The bigger β is the more probably a shorter route is selected. In a sense, every active bee performs the waggle dance after each iteration of heuristic. Consequently, the dance is not explicitly programmed or implemented.

The scout bees make one move per every iteration of the algorithm. It simply explores a new probabilistic greedy route seen in equation (5) and saves it if it's better than the best it has found this far. The pseudo code of the whole heuristic is presented in listing 3.

Listing 3: Pseudo code of the implemented Artificial bee colony optimization

```
Initialize bees with probabilistic greedy routes
while (StoppingCriteria)
  Active foragers perform local search
  if (active forager iteration limit exceeded)
    Set active forager to inactive
    Set inactive forager to active and select a route to it
  end if
  Scouts perform global search
end while
Return the best bee
```

3.3.3 2-opt local search

In this thesis, the implemented local search algorithm is called 2-opt. If the TSP problem is modeled as graph described in chapter 2.5, the 2-opt algorithm is simply a process of removing and replacing two edges. 2-opt is a special case of k-opt operator which has been widely used and proved to be efficient in the travelling salesman problem. [Talbi, 2009]

3.4 Genetic algorithm

3.4.1 Overview

Genetic algorithms belong under the class of evolutionary algorithms. Evolutionary algorithms are inspired by natural evolution and genetics.

GAs have been applied to a variety of optimization problems (see e.g. [Soares et al., 2013] and [Batenburg et al., 1995]). They are population based metaheuristics in which the solutions are produced by different operators such as mutation or crossovers. Genetic algorithms are based on the hypothesis that combining good solutions can result in even better ones.

Typical to metaheuristics, concepts such as genotype, fitness, environment etc. are used in place of typical terms such as solutions or objective function. Very often the concepts of reproduction and natural selection are present.

Since different operators and ways to simulate selection are easy to come by, numerous different algorithms have been introduced. The algorithm devised in this thesis is an example of what a genetic algorithm can look like.

3.4.2 Implementation

First step in the implementation is to initialize the population of individuals (or solutions/routes) to probabilistic greedy trails, which were presented in (5). Then 1st order crossover operator is used to create offspring from which the best 50% are selected and carried over to the next generation. The remaining half are terminated and replaced by newly initialized individuals. This process is repeated for a desired number of times. The pseudo code for the algorithm is presented in listing 4.

The 1st order crossover operator is executed as follows. First, two crossover points are selected randomly. The offspring is created so that the cities inside the points are inherited from one parent. Then, starting from the latter crossover point of the other parent, one starts adding cities that are not yet included to the offspring. In this thesis, two offspring are created by switching the parents' roles to preserve the population count.

Listing 4: Pseudo code of the implemented genetic algorithm

```
Initialize individuals to random paths
while (StoppingCriteria)
  Create offspring with 1st order crossover operator
  Reinitialize the worst 50% of individuals
end while
Return best individual
```

4 Results

4.1 Overview

The metaheuristics were tested in practice with a MATLAB implementation. Two test cases were selected for the task from TSPLIB [Reinelt, 1991]. These are a280, shown in figure 1, and lin105, in figure 2, which contain 280 and 105 cities, respectively. Performance was tested on a computer with Intel Core i7-4500U 1.8GHZ processor and 8 GB RAM.

All the heuristics, except the greedy algorithm, were iterated until their return values converged converged. An algorithm is considered converged when the best solution given by it has not changed during the last 25 iterations. Unlike the rest of the heuristics, the implemented greedy algorithm is deterministic with regard to the starting city. Despite running the algorithm with every city as a starting point, it proved to run well within reasonable time limits.

This thesis does not pursue to give a full fledged benchmark of the algorithms. Rather, a typical run of each algorithm on each problem is presented. The reason for this is explained in section 5. The results of the runs do not drastically vary between runs given enough iterations before convergence. Even with maximally biased result selection, the heuristics still hold on to their distinguishable run times, iteration counts and quality of solutions.

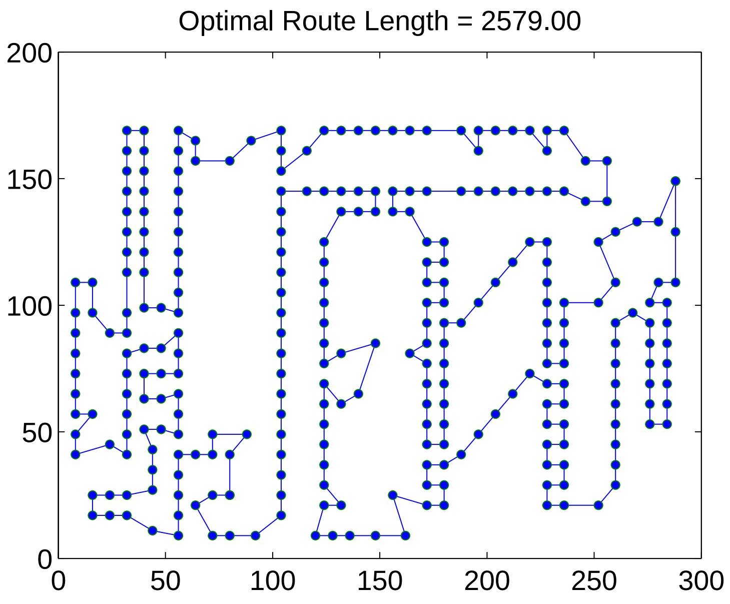

Figure 1: Optimal route for the problem a208 from TSPLIB

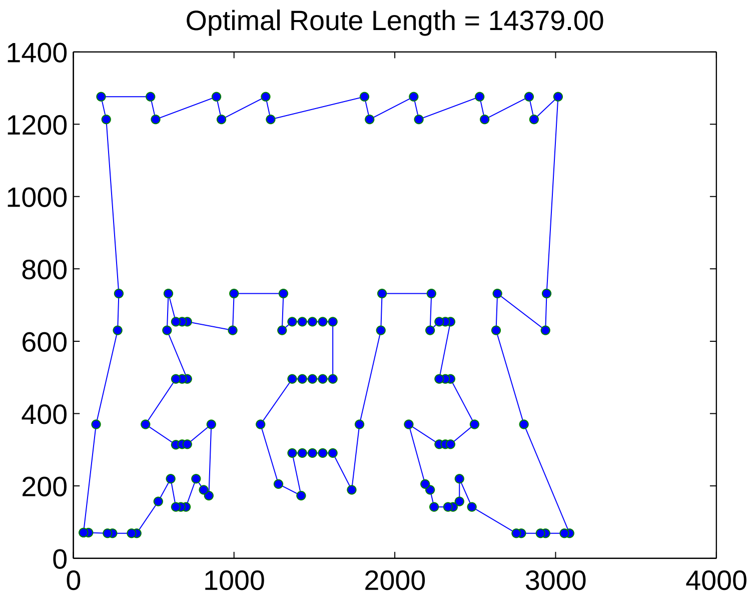

Figure 2: Optimal route for the problem lin105 from TSPLIB

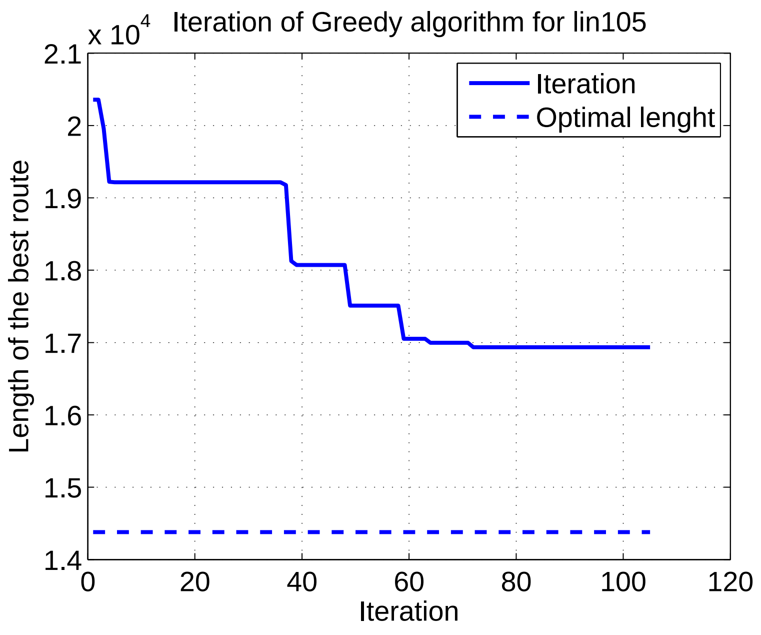

Figure 3: Iteration of Greedy algorithm for problem lin105

4.2 Greedy algorithm

The Greedy heuristic was one of two best algorithms tested in this thesis. It was also clearly the fastest one in terms of time. The results for both problems are shown in figures 3 and 4.

Both of the problems run under 0.5 seconds and yield a route that is less than 20% longer than the optimum.

In this problem, one iteration corresponds to running the algorithm with a new starting city.

4.3 Ant colony optimization

Along with Greedy heuristic, ACO achieved the best results among the implemented heuristics. The two algorithms were mostly on par with each other for the length of the outputted routes. However, the ACO implementation took the most time to converge. The iteration of the algorithm for both problems is shown in figures 5 and 6.

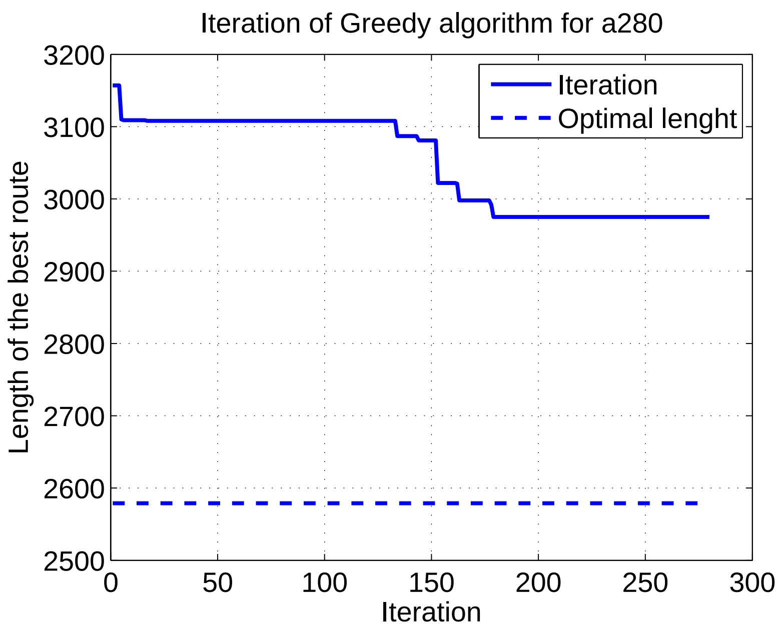

Figure 4: Iteration of Greedy algorithm for problem a280

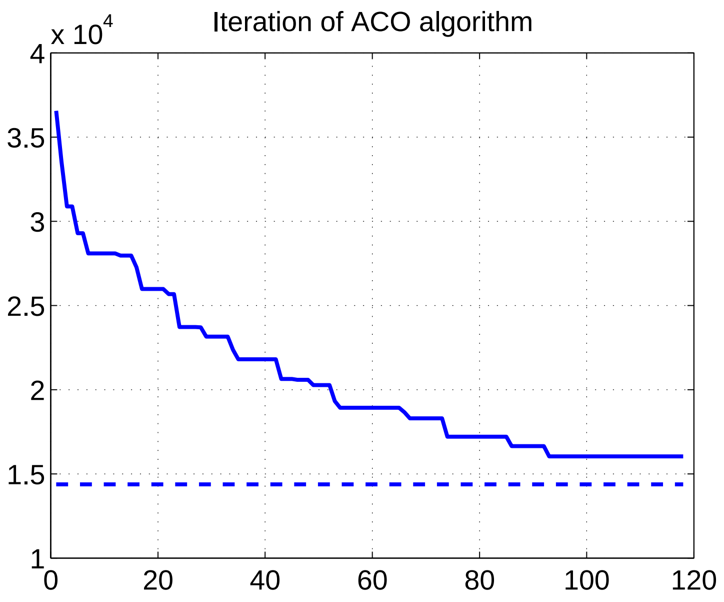

Figure 5: Iteration of ACO algorithm for problem lin105

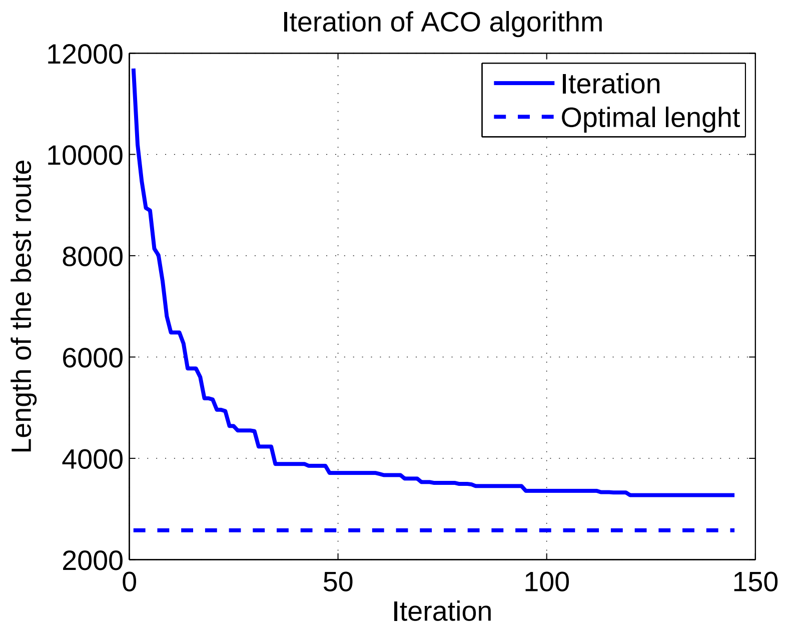

Figure 6: Iteration of ACO algorithm for problem a280

4.4 Artificial bee colony optimization

ABC algorithm was found to produce low quality results for the two problem instances. It's running time was mediocre compared to other three algorithms. It came in the third place int terms of the length of the resultant route, winning only the genetic algorithm. The results for both problems are shown in figures 7 and 8.

4.5 Genetic algorithm

The 1st order crossover operator accompanied with probabilistic greedy route initialization did not fare very well compared to other heuristics. The algorithm converged faster compared to Ant colony optimization and Artificial bee colony but produced inferior results.

In problem lin105, GA converged to a route with nearly double the length of the optimum. With problem a280 the result was almost fourfold. The results are presented in figures 9 and 10.

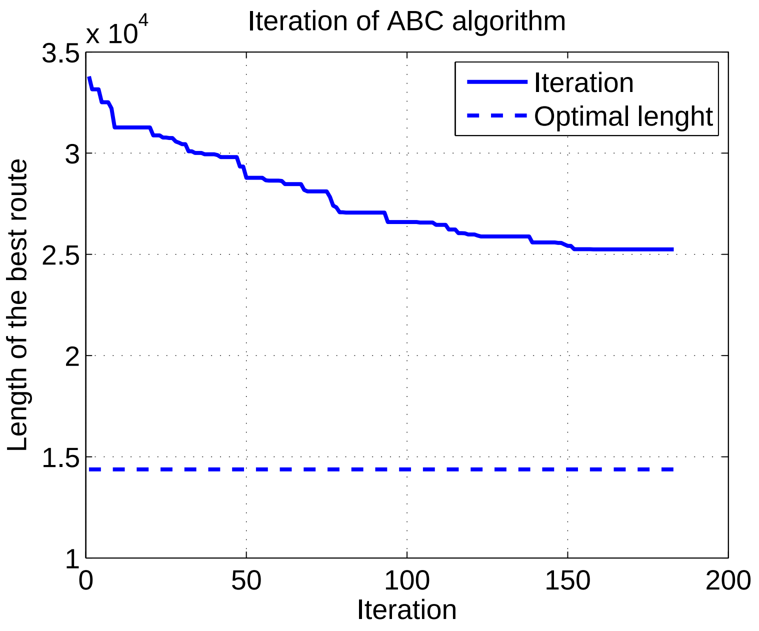

Figure 7: Iteration of ABC algorithm for problem lin105

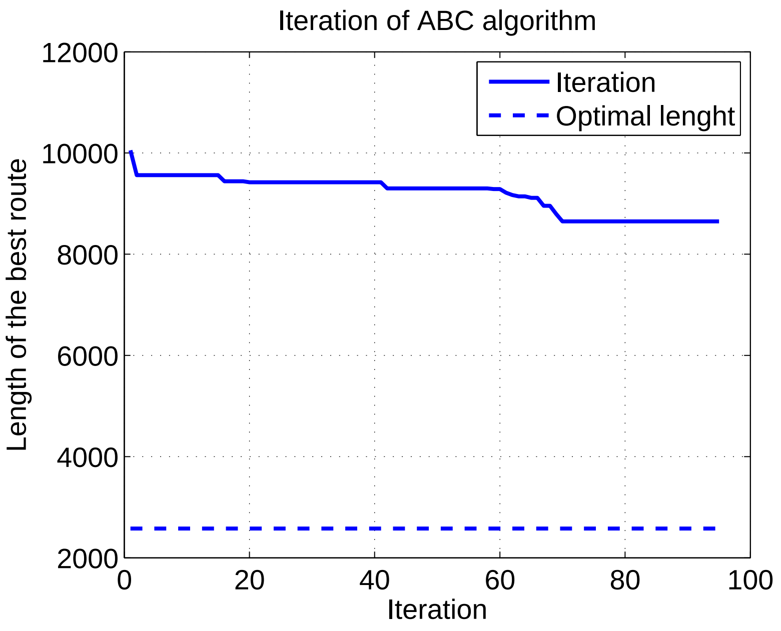

Figure 8: Iteration of ABC algorithm for problem a280

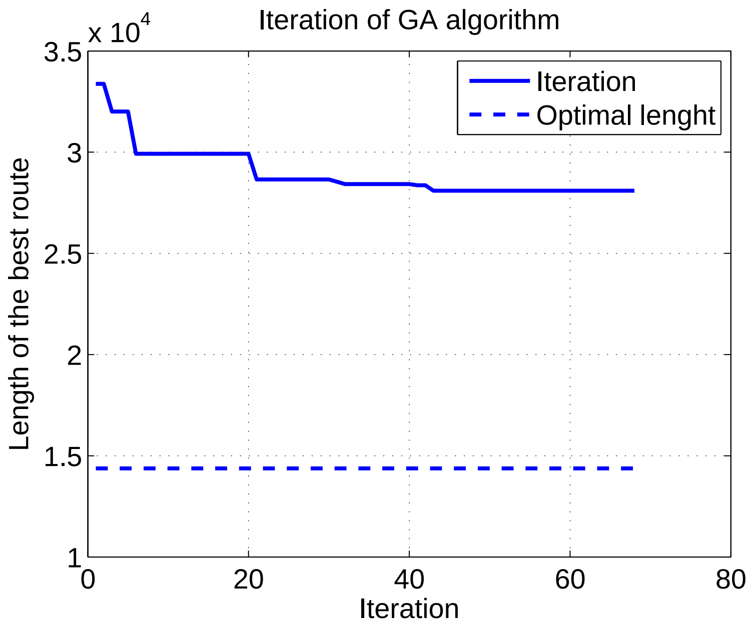

Figure 9: Iteration of Genetic algorithm for problem lin105

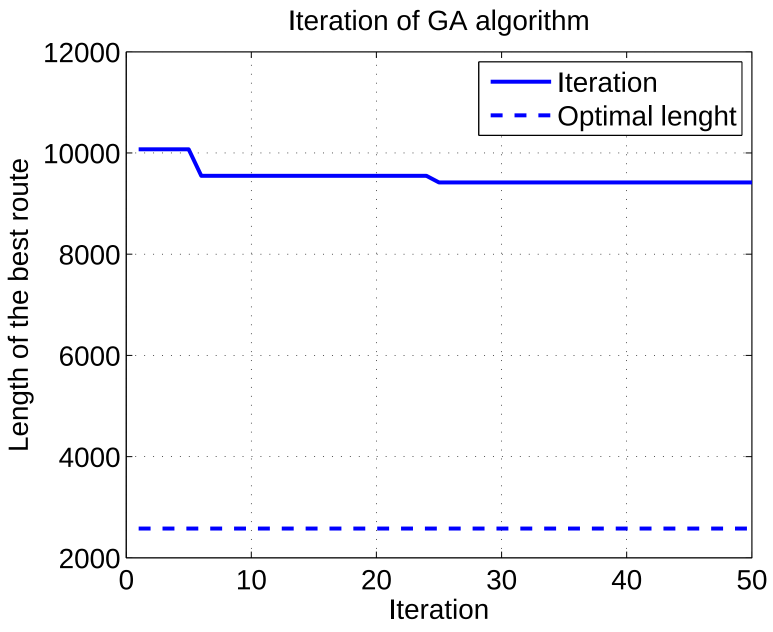

Figure 10: Iteration of Genetic algorithm for problem a280

Table 1: The results for problem lin105

	Route length	Time (s)	Iterations
Greedy	16935	0.0620	105
GA	28097	16.0940	68
ACO	16038	42.4428	118
ABC	25246	29.7336	183
Optimum	14379	N/A	N/A
Random mean	123476	N/A	N/A

Table 2: The results for problem a280

	Route length	Time (s)	Iterations
Greedy	2975	0.4885	280
GA	9486	54.8676	50
ACO	3272	229.6782	145
ABC	8649	59.5542	95
Optimum	2579	N/A	N/A
Random mean	34111	N/A	N/A

4.6 Summary of results

The results of this thesis are summarized in tables 1 and 2. To give a sense of the quality of the results, the optimum route lengths are also given. In addition, the mean length of 10000 randomized routes for both problems are presented in rows "Random mean".

From the results it is seen that the Greedy algorithm and ACO were clearly better than GA and ABC in terms of length of the resulting routes . However, every algorithm was one order of magnitude ahead of the random mean.

5 Conclusions and discussion

Among the tested implementations of metaheuristics, perhaps the Greedy algorithm's adaptiveness was the most significant finding. In addition to being conceptually the simplest algorithm, it was also remarkably faster than the other implementations. It also boasted the best result in problem a280.

All three of the non-deterministic algorithms were based on the probabilistic greedy initialization. This is the case also for Ant colony optimization, since in the first iteration all the pheromones are equal. This leads to notion that what was actually measured between these algorithms, was how well they could leverage or build upon the initial solutions. It seems that the reason why ABC and GA ended up with approximately the result was because neither of them was able to improve the sate where they started.

It is to be noted that, in general, the results of this thesis have no implications on how well certain metaheuristics are applicable to the travelling salesman problem or combinatorial optimization. This is because the algorithms devised in this paper represent only one possible way of implementing the metaheuristics in question. As mentioned in section 2, a metaheuristic is no more than a very high abstraction of the problem solving process.

The effectiveness of the non-deterministic algorithms can be tuned by changing their parameters. For instance, in ABC one can change the proportions of different bee types, how many times a local search is performed on a specific region and so on. This makes the benchmarking of different implementations tedious since the algorithms themselves should be optimized first. Sometimes even the optimization of parameters is done with a metaheuristic, which is then called a hyper-heuristic [Talbi, 2009]. In addition, there are a number of different factors that complicate the benchmarking such as the programming language, the quality of code, compilers and so on.

As shown in this thesis, metaheuristics are able to produce near optimal solutions in difficult problems in very reasonable time. The potential of metaheuristics has not gone unnoticed. For instance, by the time of writing there are numerous publications available from Google Scholar written this year. Hopefully the field continues to thrive and and produce new interesting methods and insights to the hard computational problems across all areas of research.

References

David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver, 2006.

F.H.D. Van Batenburg, A.P. Gultyaev, and C.W.A. Pleij. An apl-programmed genetic algorithm for the prediction of {RNA} secondary structure. *Journal of Theoretical Biology*, 174(3):269 – 280, 1995. ISSN 0022-5193. doi: http://dx.doi.org/10.1006/jtbi.1995.0098. URL http://www.sciencedirect.com/science/article/pii/S0022519385700987.

Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, January 1962. ISSN 0004-5411. doi: 10.1145/321105.321111. URL http://doi.acm.org/10.1145/321105.321111.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3): 268–308, September 2003. ISSN 0360-0300. doi: 10.1145/937503.937505. URL http://doi.acm.org/10.1145/937503.937505.

Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237(0):82 – 117, 2013. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2013.02.041. URL http://www.sciencedirect.com/science/article/pii/S0020025513001588. Prediction, Control and Diagnosis using Advanced Neural Computations.

M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages –1477 Vol. 2, 1999. doi: 10.1109/CEC.1999.782657.

Marco Dorigo and Thomas Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In Fred Glover and GaryA. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 250–285. Springer US, 2003. ISBN 978-1-4020-7263-5. doi: 10.1007/0-306-48056-5_9. URL http://dx.doi.org/10.1007/0-306-48056-5_9.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533 – 549, 1986. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/0305-0548(86)

90048-1. URL http://www.sciencedirect.com/science/article/pii/0305054886900481. Applications of Integer Programming.

Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130, 2000. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/S0377-2217(99)00284-2. URL http://www.sciencedirect.com/science/article/pii/S0377221799002842.

J. K. Lenstra and A. H. G. Rinnooy Kan. Some simple applications of the travelling salesman problem. *Operational Research Quarterly (1970-1977)*, 26(4):pp. 717–733, 1975. ISSN 00303623. URL http://www.jstor.org/stable/3008306.

IbrahimH. Osman and Gilbert Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996. ISSN 0254-5330. doi: 10.1007/BF02125421. URL http://dx.doi.org/10.1007/BF02125421.

M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33 (1):60–100, 1991. doi: 10.1137/1033004. URL http://epubs.siam.org/doi/abs/10.1137/1033004.

Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

Ana Soares, Állvaro Gomes, Carlos Henggeler Antunes, and Hugo Cardoso. Domestic load scheduling using genetic algorithms. In AnnaI. Esparcia-Alcázar, editor, *Applications of Evolutionary Computation*, volume 7835 of *Lecture Notes in Computer Science*, pages 142–151. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-37191-2. doi: 10.1007/978-3-642-37192-9_15. URL http://dx.doi.org/10.1007/978-3-642-37192-9_15.

E.G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley, 2009. ISBN 9780470496909. URL http://books.google.fi/books?id=SIsa6zi5XV8C.

Stefan Voss, Ibrahim H. Osman, and Catherine Roucairol, editors. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Norwell, MA, USA, 1999. ISBN 0792383699.

A Summary in Finnish

Tämän työn tavoite on esitellä metaheuristiikoita ja niihin liittyviä keskeisiä käsitteitä. Esityksen tueksi sovelletaan neljää eri metaheuristiikkaa kauppamatkustajan ongelmaan.

Metaheuristiikat ovat approksimatiivisia optimointimenetelmiä, joita tavallisesti käytetään hankalien ongelmien ratkaisuun. Tarkemmin kuvattuna metaheuristiikka on korkean tason abstraktio ongelmanratkaisuprosessista. Metaheuristiikka ikäänkuin antaa siis raamit itse optimointialgoritmin toteutukselle, jota kutsutaan tässä kontekstissa spesifiksi heuristiikaksi. Metaheuristiikoiden ideologia tai malli on usein johdettu luonnosta.

Metaheuristiikat ovat kiinnittäneet paljon mielenkiintoa 30 vuoden olemassa olonsa aikana. Todennäköisiä syitä positiiviselle huomiolle ovat niiden mukautuvaisuus ja tehokkuus. Käytännön ongelmanratkaisutilanteissa on yleensä riittäävä saada ”tarpeeksi” hyvä ratkaisu lyhyessä ajassa, mikä on metaheuristiikoille ominaista. Niitä onkin sovellettu laajasti eri teollisuudenaloilla.

Kauppamatkustajan ongelma taas on ehkäpä yksi tutkituimpia optimointiongelmia. Sillä on sovelluksia muun muassa piirien sähkösuunnittelussa (computer wiring) sekä koneiden aikatauluttamisessa (job-shop scheduling). Probleeman lähtökohtana on joukko kaupunkeja, joiden keskinäiset etäisyydet on annettu. Tehtävänä on löytää lyhin reitti, joka kulkee jokaisen kaupungin läpi täsmälleen kerran.

Metaheuristiikoille on sekä monia ominaisia että muiden optimointimenetelmien kanssa yhteisiä piirteitä. Kuten lähes kaikkien optimointimenetelmien kohdalla, myös metaheuristiikkojen soveltamisessa ongelman koodaus sekä kohdefunktion muodostaminen ovat tärkeässä asemassa. Koodauksessa viitataan tässä siihen, miten ratkaisu kuvataan matemaattisesti. Esimerkiksi kauppamatkustajan ongelman yhteydessä reitti voi olla luontevaa kuvata permutaationa kaupunkien indekseistä. Toinen tapa olisi esimerkiksi muodostaa lista, jonka jokaisessa alkiossa olisi listattu jokainen kuvattavan reitin kaupunki sekä sitä edeltävä että seuraava kaupunki. Kohdefunktio taas määrittää sen, mitä pidämme hyvänä ratkaisuna.

Ongelman koodaukseen liittyy suurelta osin se, miten ratkaisuavaruudessa määritellään naapurusto. Naapuruston määrittely vastaa kysymykseen siitä, mitkä ratkaisut katsomme olevan jonkin etäisyyden määritelmän mukaan lähellä toisiaan. Esimerkiksi kauppamatkustajan ongelmassa voisimme määrittää lähinaapuriratkaisuksi sellaiset reitit, jotka eroavat vain kahden kaupungin järjestyksessä toisistaan.

Usein metaheuristiikkoja soveltaessa joudutaan tasapainottelemaan erilaistamisen (diversification) ja vahvistuksen (intensification) välillä. Erilaistaminen tarkoittaa ratkaisuavaruuden uusien osioiden läpikäyntiä, kun taas vahvistaminen viittaa lupaavien alueiden tarkempaan tutkimiseen. Mikäli spesifi heuristiikka keskittyy vain vahvistukseen, päädymme nopeasti läheiseen lokaaliin optimiin, joka ei todennäköisesti ole kovin laadukas ratkaisu. Liika erilaistaminen taas johtaa ison alueen läpikäyntiin löytämättä tyydyttäviä ratkaisuja.

Metaheuristiikkoja on mahdollista luokitella monin eri kriteerein. Metaheuristiikan malli voi olla peräisin luonnosta tai ei. Esimerkiksi tässä työssä käsitellään muurahaisalgoritmia, joka perustuu muurahaisten ruoanetsintään luonnossa. Populaatiopohjaisissa (population-based) metaheuristiikoissa algoritmi käsittelee useaa ratkaisua samanaikaisesti, kun taas yksikköpohjaisissa (single-solution-based) rakennetaan tai parannellaan vain yhtä. Eräs luokittelukriteeri on deterministisyys eli saavuttaako algoritmi aina saman lopputuloksen, kun lähtökohdat pysyvät muuttamattomina. Metaheuristiikkoja voi luokitella vielä muistin käytön perusteella. Jotkin metaheuristiikat eivät käytä muistia ollenkaan, toisten tallentama ja käyttämä muisti voidaan jakaa pitkäaikaiseen ja lyhytaikaiseen muistiin.

Ensimmäinen sovellettavista metaheuristiikoista on nimeltään ahne algoritmi. Ahne algoritmi alkaa tyhjästä ratkaisusta, johon yksi kerrallaan kiinnitetään päätösmuuttuja. Jokaisen muuttujan arvon kiinnitys valitaan sen perusteella, mikä arvomuuttuja kombinaatio minimoi tai maksimoi (tapauksesta riippuen) kohdefunktion. Kauppamatkustajan ongelman tapauksessa tämän voi ajatella johtavan lähimmän naapurin valintaan. Toisin sanoen, jokaisella kierroksella seuraavaksi vierailtava kaupunki on se kaupunki, johon on tämänhetkisestä kaupungista lyhin euklidinen etäisyys.

Seuraava sovellettava metaheuristiikka oli muurahaisalgoritmi. Tämä lähestymistapa matkii havainnoitua muurahaisten ruoanhakua luonnossa. Kyseessä on populaatiopohjainen metaheuristiikka, jossa muurahaisiksi kutsutut agentit keskustelevat keskenään epäsuorasti feromonin välityksellä. Muurahaiset jättävät kulkemalleen reitille feromonia löydetyn ruoanlähteen tai reitin laadun perusteella. Etsiessään ruokaa ne myös liikkuvat todennäköisemmin feromonin suuntaan. Metaheuristiikan idea perustuu feromonin haihtumiseen, jolloin ajan saatossa parhaat reitit voimistuvat ja huonoimmat heikentyvät.

Työssä tarkasteltu mehiläisalgoritmi perustuu puolestaan mehiläisten ruoanhakuun. Tämän työn implementaatiossa mehiläiset jaetaan kolmeen luokkaan: tiedustelijoihin, aktiiviisiin sekä epäaktiivisiin työläisiin. Tiedustelijat ovat vastuussa algoritmin erilaistamisesta, sillä niiden tehtävänä on etsiä uusia lupaavia alueita ratkaisuavaruudesta. Aktiiviset työläiset taas hoitavat vahvistuksen tehdessään lokaaleja hakuja hyväksi todetuilta alueilta. Tiedustelijat kommunikoivat reittinsä epäaktiivisille työläisille, jotka sitten päättävät lähtevätkö ne tutkimaan aluetta. Mitä paremman alueen tiedustelija on löytänyt, sitä todennäköisemmin epäaktiivinen työläinen sen valitsee. Valinnan myötä työläisen status vaihtuu aktiiviseksi ja se alkaa suorittaa lokaalia hakua. Kun aktiivinen työläinen on tutkinut jotain aluetta riittävästi, se epäaktivoituu ja jää seuraamaan tiedustelijoiden reittejä.

Geneettisiä algoritmejä on ajan saatossa kehitetty hyvin monia erilaisia. Tässä työssä sovelletaan verrattain yksinkertaista ensimmäisen asteen risteytystä (1st order crossover). Reitit alustetaan ensin probabilistisella ahneella periaatteella. Toisin sanoen, mitä lähempänä seuraava kaupunki on nykyistä kaupunkia, sen todennäköisemmin sinne siirrytään. Jokaisella kierroksella populaatio risteytetään keskenään. Tämän jälkeen parempi puolikas populaatiosta jatkaa seuraavalle iteraatiokierrokselle, kun taas huonompi puolikas alustetaan uudestaan.

Edellä mainittujen metaheuristiikoiden implementaatiota sovellettiin kahteen kauppamatkustajan ongelmaan, jotka sisälsivät 280 ja 105 kaupunkia. Tuloksista selvisi, että muurahaisalgoritmi sekä ahne algoritmi saivat huomattavasti parempia ratkaisuja, kuin mehiläisalgoritmi tai geneettinen algotirmi.

Ehkäpä mielenkiintoisin havainto tuloksista oli, että ahne algoritmi pärjäsi niin hyvin verrattuna muihin verrattuna. Sen ajoaika oli vain murto-osan muista. Se ylsi sijoituksille yksi ja kaksi reitin pituudella mitattuna.

Lukuunottamatta ahnetta algoritmia, tämän työn metaheuristiikoiden suorityskykyä voi parantaa optimoimalla niihin liittyviä parametrejä. Esimerkiksi mehiläisalgoritmi johtaa eri tuloksiin, mikäli tiedustelijoiden ja työläisten suhdetta vaihtelee. Tämä tekee implementaatioiden vertaamisesta työlästä. Lisäksi voidaan tunnistaa muita vertailuun vaikuttavia tekijöitä, kuten koodin laatu, kääntäjät ja niin edelleen. Tästä syystä vertailua ei ole toteutettu kovinkaan perusteellisesti.

Huomioitavaa on, että nämä tulokset eivät anna osviittaa metaheuristiikkojen keskinäisestä paremmuudesta. Edellisen kappaleen perusteluiden lisäksi on muistettava, että tässä työssä tutkittiin vain metaheuristiikkojen implementaatioita. Kuten alussa mainittiin, metaheuristiikka taas on määritelmän mukaan vain abstrakti kuvaus ongelmanratkaisuprosessista.