EXAMENSARBETE INOM TEKNIK,
GRUNDNIVÅ, 15 HP
STOCKHOLM, SVERIGE 2017

Ant Colony Optimization Algorithms

Pheromone Techniques for TSP

ADEL BAVEY, FELIX KOLLIN

KTH
SKOLAN FÖR DATAVETENSKAP OCH KOMMUNIKATION

Ant Colony Optimization Algorithms - Pheromone Techniques for TSP

ADEL BAVEY & FELIX KOLLIN

Degree Project in Computer Science
Date: June 5, 2017
Supervisor: Jeanette Hällgren Kotaleski
Examiner: Örjan Ekeberg
Swedish title: Ant Colony Optimization Algoritmer - Feromontekniker för TSP
School of Computer Science and Communication

Abstract

Ant Colony Optimization (ACO) uses behaviour observed in real-life ant colonies in order to solve shortest path problems. Short paths are found with the use of pheromones, which allow ants to communicate indirectly. There are numerous pheromone distribution techniques for virtual ant systems and this thesis studies two of the most well known, Elitist and Max-Min.

Implementations of Elitist and Max-Min ACO algorithms were tested using instances of the Traveling Salesman Problem (TSP). The performance of the different techniques are compared with respect to runtime, iterations and approximation quality when the optimal solution could not be found. It was found that the Elitist strategy performs better on small TSP instances where the number of possible paths are reduced. However, Max-Min proved to be more reliable and better performing when more paths could be chosen or size of the instances increased. When approximating solutions for large instances, Elitist was able to achieve high quality approximations faster than Max-Min. On the other hand, the overall quality of the approximations were better when Max-Min was studied after a slightly longer runtime, compared to Elitist.

Sammanfattning

Ant Colony Optimization (ACO) drar lärdom av beteende observerat hos riktiga myror för att lösa kortaste vägen problem. Korta vägar hittas med hjälp av feromoner, som tillåter myror att kommunicera indirekt. Det finns flera tekniker för att distribuera feromoner i virtuella myr-system och denna rapport kommer studera två av de mest kända, Elitist och Max-Min.

Implementationer av Elitist och Max-Min ACO algoritmer testades med instanser av Handelsresandeproblemet (TSP). Prestandan hos de olika teknikerna jämförs med avseende på körtid, iterationer och approximeringskvalité när den optimala lösningen inte kunde hittas. Det konstaterades att Elitist strategin fungerar bättre på små TSP instanser där antalet möjliga stigar är begränsade. Däremot visade det sig Max-Min vara bättre och mer pålitlig när instansernas storlek ökades eller när fler stigar kunde väljas. När lösningar approximerades för stora instanser kunde Elitist uppnå approximationer med god kvalité snabbare än Max-Min. Däremot var den generella kvalitén hos approximationerna bättre när Max-Min studerades efter en lite längre körtid, jämfört med Elitist.

Contents

Chapter 1

Introduction

Efficient problem solving techniques are desired in Computer Science in order to fully utilize hardware and perform complex computations in real time. Dynamic and responsive systems allows us to integrate technology in our dynamic and fast paced everyday life. An extensively studied field in Computer Science is therefore NP-complete problems. It is believed that there are no algorithms able to find solutions to NP-complete in polynomial time, which often limits real time systems to approximations (Widjaja, 2004).

In order to find new approaches in fields of science it is common to study nature, for example in medicine where medications build upon chemicals produced by plants (Raskin et al., 2002). In contrast, studying coordinated behaviour in nature allows us to identify behaviour which aims to solve a specific problem in real time. One such seemingly complicated behaviour is ant colonies' ability to coordinate navigation. They achieve this using chemicals called pheromones, which allows them to communicate indirectly without their auditory and visual senses. The ants' foraging behaviour is especially interesting because the problem closely resembles the NP-complete problem called the Traveling Salesman Problem (TSP). Consequently, it is possible to simulate the ants' behaviour in order to solve this problem in real time systems, so called bio-inspired computing.

Ant Colony Optimization (ACO) is a field which builds upon observations of real life ants in order to construct algorithms which solves shortest path problems. The field is extensively studied which has re-

sulted in several algorithmic approaches being deduced. The aim of this study is therefore to investigate two of the pheromone distribution techniques of ants used in ACO algorithms so that future implementations may see the benefits of either technique when solving the TSP.

1.1 Research Question

In order to identify the key aspects of different pheromone techniques the following research question was chosen for the study:

> Which pheromone distribution technique used in Ant Colony Optimization, Elitist or Max-Min, has the best performance on the Traveling Salesman Problem?

Performance refers to the runtime needed to solve TSP, but this study also considers the number of iterations needed in order to evaluate the performance in further detail. The problem instances of TSP are described in further detail in section 3, Methods. They vary in size to represent a large portion of the spectrum of possible instances and impose restrictions on movements in order to capture specific scenarios that the algorithms may encounter in other applications. Additionally, the measurement of performance also considers the quality of the solution because large instances of TSP may not be solved optimally within a reasonable timeframe using ACO.

TSP was chosen because it is a well known complex problem that can can be solved using ACO. By using it in our experiments we hope to convey the power of bio-inspired computing and encourage its further use in fields outside of biological simulations.

1.2 Purpose

By comparing the different pheromone distribution techniques, the core of ACO, it is possible to deduce specific situations where one pheromone technique is favoured over another. Problems in Computer Science are rarely solved by a single optimal algorithm, but instead the algorithm adapts to the problem at hand. This study there-

fore explores the possibility of identifying TSP instances where the results of pheromone techniques differ.

Using ACO to solve TSP is an extensively studied subject and there are many different approaches to the problem. The study does not break any new ground, but rather serve as a reference when choosing an algorithm implementation with respect to likely TSP instances.

1.3 Scope

This is a limited study on which variant of ACO has the best performance. Only implementations of Elitist and Max-min pheromone techniques are studied in order to deliminate the thesis to a reasonable scope as a result of a limited development timeframe. The same rationale explains why only pheromone techniques are considered, and not other aspects of optimizing the meta-heuristic, such as local search.

Chapter 2

Background

The background of this study supplies the reader with the biological background required to understand how ACO is inspired by biology. The reader will also find the technical background describing the basis of ACO and the techniques investigated in the study. Furthermore, prior studies and their results are presented to convey the current state of the field.

2.1 Biological Background

Ants in general have limited individual capabilities due to the fact that they have adapted to living in colonies over time, thus favouring traits beneficial to the colony instead of individuals. Colony societies are biological phenomenons, following the mantra of strength in numbers and therefore gaining resistance to partial mortality such as disease, predetation and other environmental factors (Winston, 2010). However, the field of ACO focuses on the social interactions of ant colonies, mainly foraging behaviour, in order to solve problems using optimal foraging behaviour found in nature. Studying ants' foraging solutions can therefore shed light on complex problem sets because they have already gone through natural optimization. This is simply following the principle of food relative to energy expended whilst acquiring food, thus optimizing for short and efficient food trails.

Communication between ants differ from that of humans because their visual and auditory communication is limited. Instead they rely on chemicals called pheromones in order to exchange information amongst

each other. When ants are foraging they lay down pheromones along the trail they are traveling and favour pheromone rich trails at decision points. They are therefore communicating indirectly by altering the environment so that future ants are influenced by this change. This form of self-organization is called stigmergy and produces complex networks of trails without the need for planning (Ramos et al, 2005). The main path eventually converges towards the shortest trail because pheromones are more often distributed along that trail. This solution to path problems is desirable in computer science and is expanded upon in this study.

2.2 Technical Background

This section provides a rundown of the technical information needed to understand the report. This includes the ant simulation technique that ACO employs, a brief description of TSP, and explanations of the different pheromone techniques.

2.2.1 The Simulation Technique

The simulation technique for ACO is explained here. First, the idea of how the meta-heuristic ACO works is explained. This is followed by how it is implemented using artificial ants.

The Meta-Heurustic ACO

In 1992, Marco Dorigo devised an algorithmic interpretation of ant colonies' foraging behaviour (Dorigo et al, 1992). The main idea was to simulate the stigmergy of ant colonies. He later coined the term ACO to describe this area of Computer Science (Dorigo and Stützle, 2009).

The idea Marco devised was the following: To begin with, ants will walk randomly in an environment and drop pheromones along their path. Pheromones dissipate over time, and are reinforced only when an ant walks over the point. When an ant has found its target, it will walk back the same path to its starting point and reinforce the pheromones along that path. If the path the ant took was short, the pheromone density will be stronger on that path. The path with the strongest pheromone density must therefore be the shortest path, and

the one the ant colony should use to find their target.

Thus far, there have not been any specific mention of how to implement ACO as an algorithm, and the above explanation isn't something that can clearly be translated to code. This is because ACO is a meta-heuristic (Blum and Roli, 2003), a general idea of how to construct an algorithm for a problem. A greedy algorithm can be used to solve the shortest path problem; at every branching point in the search, choose the path with the least cost. This is an example of a heuristic, as we used some understanding of the problem to create a simple rule for solving it. A meta-heuristic takes this one step above in its abstraction, which is to define a heuristic for creating heuristics. In this case, take a problem, think of it in terms of ACO, and see what heuristics can be derived from it.

While the meta-heuristic definition of ACO results in a degree of flexibility for which problems it can solve, it also introduces issues. Two important ones relevant to this report is: which problem instances can it solve, and how well can it solve them? Due to the relative newness of ACO, no theoretical foundation that determines when it is appropriate to use has been found (Sörensen, Sevaux and Glover, n.d.). Nor can we be certain that it will perform well for a given problem until it has been tested experimentally on that problem. Luckily, some problems have been studied, and ways of implementing ACO have been tested and proven to work well (Dorigo, Caro and Gambardella, 1999; Mavrovouniotis, 2013).

Artificial Ant Algorithms

While the meta-heuristic ACO covers how an ant simulation may be used to solve problems it does not provide specific algorithm implementation details. This is mainly because there are multiple ways of constructing an algorithm. It is therefore necessary to identify the algorithm's intended use before choosing any of the approaches. In this study, the algorithm is not constructed as a way of simulating ants, but rather as a way to make use of the inherited natural optimization. Consequently, the conventions of the algorithms are not entirely supported by studies of real life ants and the behaviour of artificial ant is instead derived from virtual environments. An example of such conventions

are the retracing and memory aspect, where artificial ants only remember the found solution and retraces its steps instead of considering the pheromones on its way back.

As previously mentioned, the first set of algorithms to implement ACO were constructed by Dorigo in 1992, which expanded upon previous ant simulation techniques (Goss et al, 1989). Dorigo proposed three algorithms; ANT-quantity, ANT-density and ANT-cycle. The first two mentioned produce similar results, although ANT-density considers the length of every edge in the graph whilst distributing pheromones. The pheromone distribution technique is similar to previous work and in line with the most basic idea of the heuristic, where ants lay down pheromones when searching for a path as well as retracing the found solution. It was however concluded that this way of pheromone application was prone to slow convergence in more complex environments. It was also discovered that using the algorithm with a low population of ants, 4-16 individuals, did not reach an optimal solution to the minimum path problem.

The third algorithm, ANT-cycle, outperformed the other two, especially in more complex problems. In this implementation ants no longer distribute pheromones when constructing a possible solution, thus decreasing the possibility of loops as a result of autocatalytic behaviour. This is because it can not influence itself or other ants in the same iteration while searching for a solution, and only consider previous solutions when choosing their path.

```
Initialize all pheromone values on edges or use the ones from
 previous iteration.
Place the ants at their starting position.
 while !end do
                    ▷ can be n cycles or when all nodes are visited
   for each node i do
     for each ant on node i at time t do
       Move ant to next node with possibility given by
        equation 1.
     end
   end
   Compute the pheromone trails to be distributed over edges.
end
```

Algorithm 1: Simplified Ant System

.

The simplified and more general version of Dorigo's Ant System pseudo code in Algorithm 1 gives a general sense of how one iteration of the procedure behaves. The equation which is used to decide how an ant will move at a specific state is given by the equation:

$$P_{ij}(t) = \frac{\tau_{ij}(t)^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{j=1}^{n} \tau_{ij}(t)^{\alpha} \cdot \eta_{ij}^{\beta}} \qquad \text{(Equation 1)}$$

Where $P_{ij}(t)$ is the probability to move from node i to node j at time t. The $\tau_{ij}(t)$ is the amount of pheromones on the edge from node i to j at time t. The η_{ij} value is a distance factor, which limits the importance of pheromones farther away, referred to as visibility. The constants alpha (α) and beta (β) are chosen experimentally and allow the user to control the relative importance of a trail versus its visibility. This probability equation was in fact derived from observations of real ants (Goss et al 1989). Furthermore, the equation differs between implementations of ACO and improvements have been made since it was first introduced in 1992 (Stützle and Dorigo, 1999). One possible approach is a pseudo random equation which factors in previous pheromone trails more heavily. However, in order to keep finding alternate paths a local pheromone update was introduced. The local pheromone update removes a portion of the pheromone along a trail whenever an ant moves across it.

Moreover, another attempt to improve the initial ant system is the introduction of global pheromone evaporation (Stützle and Dorigo, 1999). After some iterations a portion of the pheromones along each trail is removed. This encourages convergence through elimination of long paths. The search routine can forget errors and poor quality solutions in favor of better options. A constant rho (ρ) is used to control the evaporation, and like α and β it is derived experimentally and differs depending on the pheromone distribution technique.

2.2.2 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) can be described as follows: Given a list of cities and the distance between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city. TSP is an NP-hard problem, which means that there is no known algorithm that can solve it in polynomial time.

A more useful way of describing TSP is as a graph problem: the cities can be thought of as vertices, and the distances between the cities can be thought of as weighted undirected edges. The paths distance will then be the sum of the edge weights on the path. The problem is then an minimization problem; starting and finishing at the same vertex after having visited every other vertex in the graph, with as little distance as possible.

Often, the graph will be a complete graph. If two vertices do not have an edge between them (such as when using real-world data), an arbitrarily long edge can be added without affecting the optimal tour.

As there is no known efficient algorithm for solving TSP, approximation algorithms are often used. Which is to say, if an optimal path can not be found, at least make sure a good path is found. A good path is a path that fulfills the requirements for a solution, and is within some margin of the optimal path. Getting an approximate solution as close to the optimal is often the aim when dealing with NP-hard problems (Cook, 2013). This is also the aim of the ACO algorithm.

Initialize trails* and place each ant in a random city.
for *each ant* **do**
  while *cities unvisited* **do**
    Move ant to next city with possibility given by equation 1.*
  end
  Set ant position to initial city.
end
Update pheromone trails.*

*Varies depending on pheromone technique.

Algorithm 2: One iteration of solving TSP with ACO

The stopping criteria depends on what the user specifies. It can either be a set amount of time, or that the program stops when a known optimal value is reached.

2.2.3 Pheromone Techniques

Brief explanations of the pheromone techniques referred to in this report are found in this section.

Elitist (EAS)

In Ant System, every ant is considered equal in the amount of pheromone it drops. In an Elitist Ant system, the ant that has found the shortest path so far (the elitist ant) drops more pheromones along its path. This is to encourage faster convergence on a possible optimal solution. The downside is that it might find a sub-optimal solution, and stagnate on this solution for a number of iterations. In this case, it will either stay on this sub-optimal solution for the duration of the runtime, or it will manage to break away and find a better solution (Dorigo and Stützle, 1999).

Parameterization decides how large the elitist ant's influence is, and needs to be set according to the problem. If a sub-optimal solution is acceptable, a stronger elitist ant might be used to encourage faster convergence. If the optimal must be found, a relatively weak elitist ant may be used; though in this scenario, a different pheromone technique

may be warranted.

Multiple elitist ants may also be used, with a parameter e deciding how many of the top e ants with the current best solutions should drop more pheromones.

The benefit of elitist is how simple it is, which makes both implementation of the system and analysis of its performance relatively easy. The drawback is its tendency to converge on sub-optimal solutions and stagnate (Stützle and Hoos, 2000).

A study has identified the optimal parameter settings to be: $\alpha = 2$, $\beta = 5$ and $\rho = 0.5$ (Valdez and Chaparro, 2013).

MaxMin (MMAS)

The main idea of the MMAS distribution technique is to apply further limitations on pheromone distribution in order to increase control (Mohammadrezapour, 2014). To achieve this, the pseudo random decision equation, covered earlier, is used with local pheromone update. In order to avoid premature stagnation, the pheromone trail along each edge is limited to a range [min, max], which is what gives the technique its name (Stützle and Hoos, 2000). In other words, MMAS's main focus is to continually find alternate paths when constructing a solution. Additionally, edges are initialized to the max value in order to favour high exploration of alternate paths at the start of the algorithm. The pheromone distribution also differs from EAS by only allowing the global best or iteration best solution to deposit pheromone along its included edges.

Furthermore, MMAS is often coupled with local search which uses a strategy to improve the current solutions. The local search mainly considers the global best solution instead of the iteration best (Stützle and Hoos, 2000).

The optimal conditions such as number of ants and parameter settings for running MMAS has been found by conducting numerous tests (Dorigo and Stützle, 1999). It was found that 25 ants performs best, any more ants causes runtime slowdown. The optimal settings

for the parameters was $\alpha = 1$, $\beta = 2$ and $\rho = 0.2$.

Chapter 3

Methods

The aim of this study is to analyze the performance of different optimizations of ACO on TSP. For this reason an existing implementation was used, rather than a self-built one, in order to conduct tests. The chosen implementation is called ACOTSP, by Thomas Stützle (Stützle, 2004). Instances of TSP is used as input, and different outputs can be gained from it, depending on what the user is after, such as overall statistics or results for an individual run. In this case, the number of iterations it takes to find an optimal solution is recorded.

To get TSP instances, TSPLIB was used ("Symmetric TSPs", 2007). TSPLIB contains samples from various sources, either real-world or man-made, and provides a large enough dataset in order to get meaningful results from ACOTSP. It is also worth mentioning that the name of each TSP instance contains the number of vertices it consists of, for instance the problem eil51 which has 51 vertices.

Instances from TSPLIB are given as input into the above implementation, and the results are recorded in the Results section.

Two primary parameters were used in the experiment, the setting for the number of ants and the setting for nearest neighbours, which is detailed below. Where the "Ants" parameter decides the number of ants that will wander in the graph trying to find the optimal tour. How these ants behave differs by pheromone technique. Overall, having a high amount of ants results in more independent agents trying to find the optimal tour, but having too many ants can lead to a "chain-

reaction" where too many ants gather around a sub-optimal tour and keep reinforcing the pheromone density there. Having too few ants means they won't affect each other too strongly, but will result in less ants trying to find an optimal tour. The right amount of ants coupled with the right pheromone technique is the best way to achieve the best results.

The "Nearest Neighbour" (NN) parameter is used restrict movement between nodes, which essentially restricts the amount of edges considered in the graph. This parameter decides the number of cities in a priority queue (ordered by smallest distance from current city) that each ant will consider moving to. If NN is set to 5, each ant will only consider moving to the 5 nearest cities according to Equation 1. If NN is set to 1, each ant will only consider moving to the closest city (and will thus behave similarly to a greedy algorithm). If NN is set to the number of nodes in the graph, every possible edge can be probabilistically chosen. Setting NN to a low value results in ants closing off long edges, but won't consider that one of these long edges can be a part of the optimal tour. Setting it to a high value will result in ants possibly choosing very long edges that have a high chance of not being a part of the optimal tour, but won't close off long, but beneficial, edges.

Every combination of values for Ants and NN values (detailed in 3.2) are tested for every TSPLIB instance. How the program is run, and which values for parameters are used, is detailed in the following subsections.

3.1 ACO Software

The implementation ACOTSP is written in C, and takes in parameters from a unix terminal. It was compiled for a Mac operating system, and was run through a Mac terminal. The terminal command used looks like the following:

```
./acotsp -i input/att532.tsp -eas/-mmas -o 27686
      -g 3 -m 5 -b 5 -e 0.5 -a 1 -t 30 -r 5 -c 1
```

Parameter	Explanation
.-i input/att532.tsp	initialize file input/att532.tsp (and clear if it already exists)
-eas/-mmas	specify the pheromone technique (EAS or MMAS)
-o 27686	set the optimal value to be found to 27686
-g 3	set NN to 3
-m 5	set number of ants to 5
-b 5	set β to 5
-e 0.5	set ρ value to 0.5
-a 1	set α to 1
-t 30	set maximum runtime to 30 seconds
-r 5	run 5 independent trials
-c 1	set 1 elitist ant (not used for mmas)

Table 3.1: Explanation of a program run

This produces three files, whereas only two are used. A file (stat.att532 in the above case) detailing the independent trials and how the best solution found so far changes over the iterations for each trial, and a summary file (best.att532) detailing the best, average, and worst solutions found over all trials. These files are then used as input for a second (self-built) program built in Java (see 3.3) that uses the files to produce the charts found in Results.

3.2 Test Suite

For the different implementations, the following parameters are used:

Both: Maximal runtime = 30 seconds, no. independent trials = 5
EAS: $\alpha = 1$, $\beta = 5$, $\rho = 0.5$
MMAS: $\alpha = 1$, $\beta = 2$, $\rho = 0.2$

The maximal runtime and the no. independent trials were chosen experimentally, as they provided a reasonable amount of time and a reasonable number of independent trials to be able to run all the tests and be sure that statistical anomalies are ignored. Running all the tests to completion under these parameters took approximately 12 hours and we chose this timeframe for the tests in order to generate new ones overnight if the testing proved faulty. The parameter settings for the

pheromone techniques are derived from previous studies which are covered in the technical background.

The optimal value parameter depends on the TSPLIB instance used, and have been gathered from previous findings ("Symmteric TSPs", 2007). The ones used for this report can be found in table 3.2.

Instance name	Optimal tour length
eil51	426
kroA100	21 282
d198	15 780
pcb442	50 778
att532	27 686
rat783	8 806
pcb1173	56 892
d1291	50 801
pr2392	378 032

Table 3.2: Optimal tour length for TSP instances.

The number of ants and the NN value, as previously mentioned, is varied to test different parametrizations. The values used are:

Ants: 5, 10, 15, 20, 25, 30, 35
NN: 5, 10, 15

The end result is (9 TSPLIB instances) · (7 Ants values) · (3 NN values) = 189 different test runs. Each of these 189 tests generate two files of interest, although only some of these tests are used for the charts in Results. In particular, the result files for d198, rat783, pr2392.

3.3 Representing Results

Due to the amount of results, a program generating charts for each series of tests was used. It was developed using Java and the free charting library JFreeChart ("jfreechart", 2014).

The result files from the ACO runs are parsed by the program and converted into data usable by the charting library. The charts

were also configured to give a clear representation of the results. The tool was used multiple times when conducting tests in order to identify suitable parameters which produced conclusive results which can be used within this thesis.

There are two types of charts produced by the tool, bar charts and line charts. Bar charts are used to represent iteration and time data for each set of ants. Line charts are used to display convergence of the tests and the overall comparison results.

In order to produce convergence charts, a set number of ants needs to be used. This choice was made for each instance after analyzing the initial time and iteration results for the two pheromone distribution techniques.

Chapter 4

Results

The results of the study are presented in this section, grouped according to the size of the TSP instances. The instances are in groups of three, and they are referred to as small, medium and large sized throughout the study. Small consists of the first three instances, eil51, kroA100 and d198. Medium covers the instances pcb443, att532 and rat783. Finally, large, which refers to the instances pcb1173, d1291 and pr2392.

The charts presented here have been picked to represent the instance group as a whole, in order to provide the reader with a clear summary of the findings.

The bar charts presented uses color hues to differentiate best, average and worst results. Where best is represented by the regular color, average by lightest color and worst by the darkest color. Some bars may only be outlined, this indicates that the result did not reach the optimal solution and the data displayed is from the best solution found before the test terminated.

Data represented in convergence charts are gathered at the points where a new best solution is found, this means that any stagnation of the algorithm may either be represented by a line with little incline or when the line ends prematurely, displaying the case when no new solutions are found.

4.1 Small Sized Instances

4.1.1 Runtime

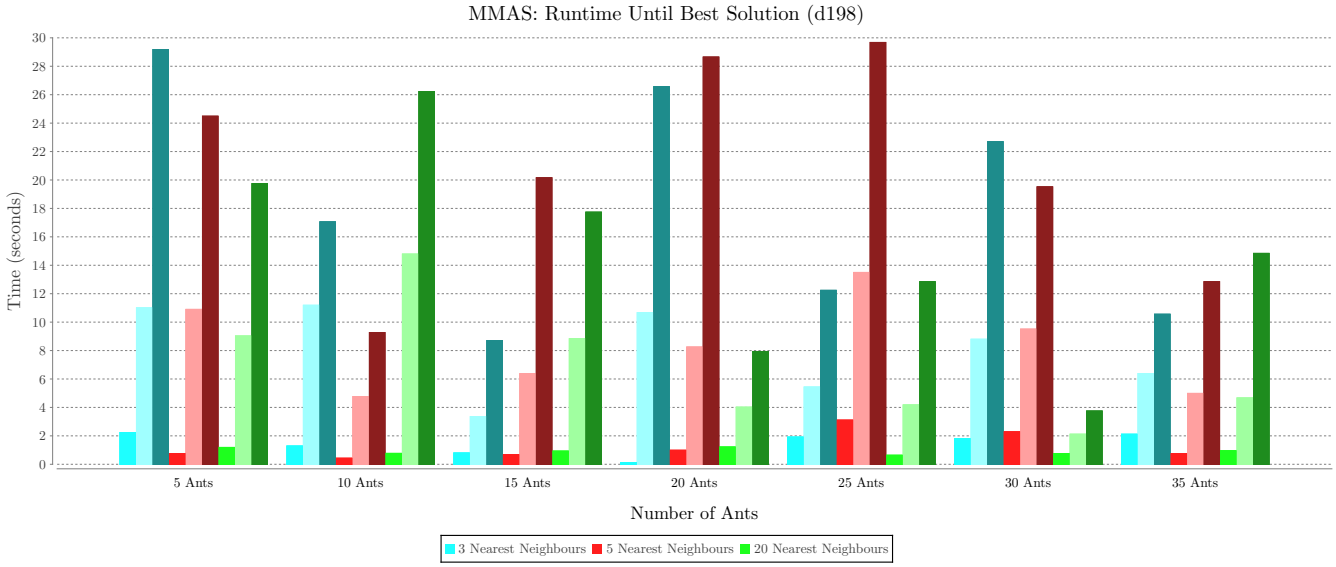

Figure 4.1

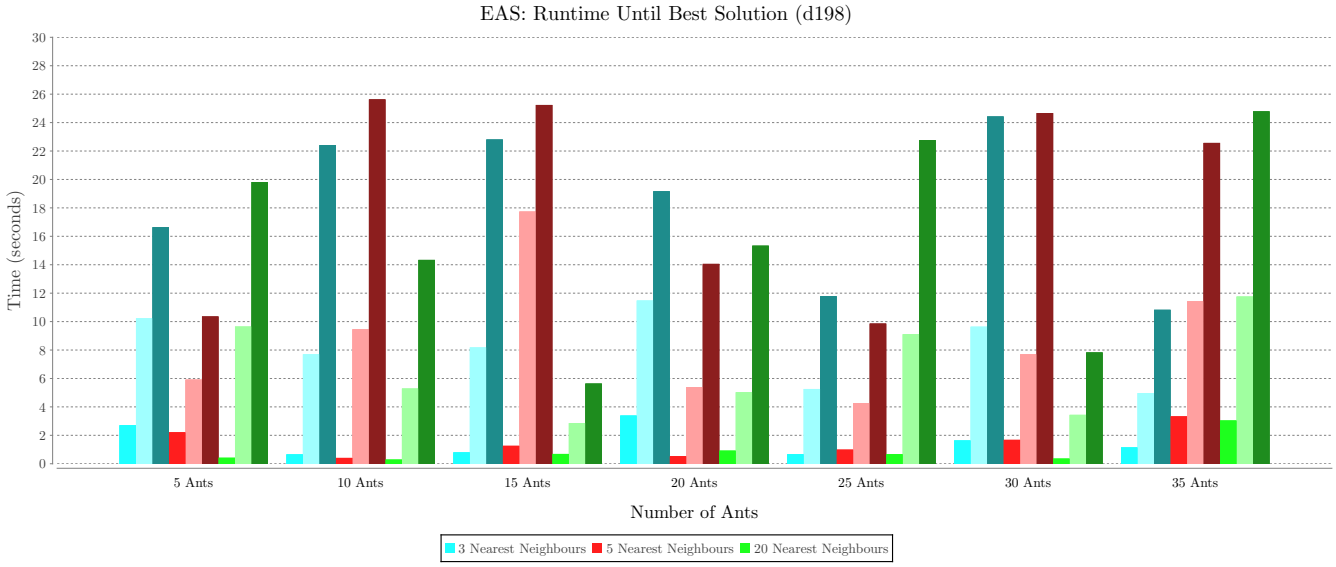

Figure 4.2

4.1.2 Iterations

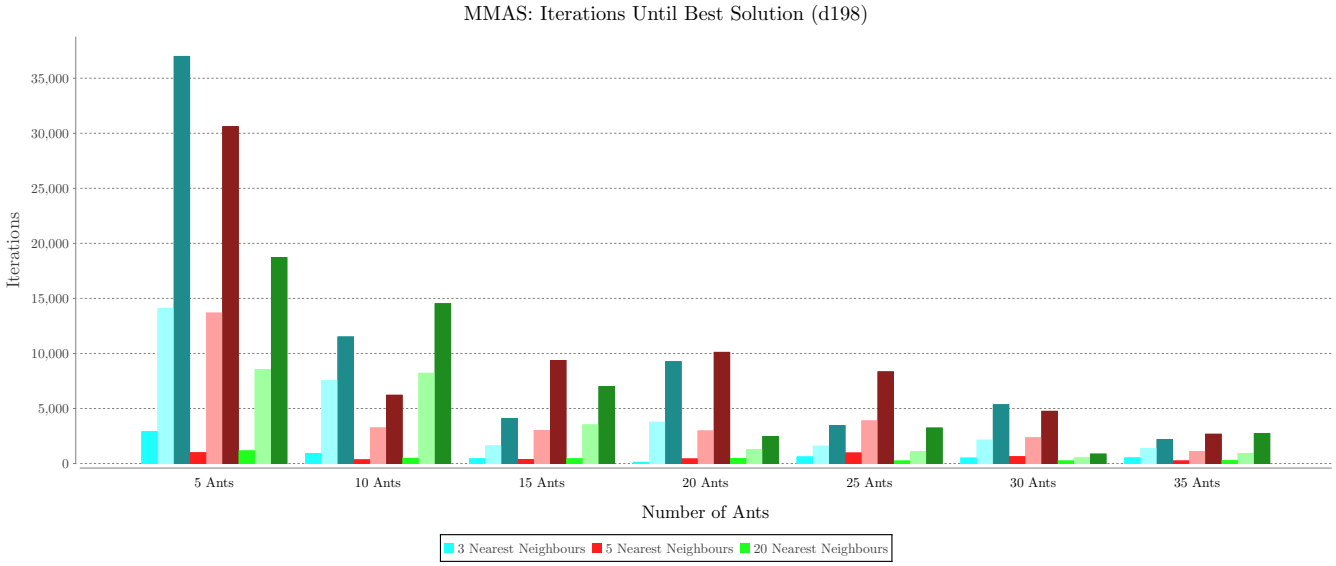

Figure 4.3

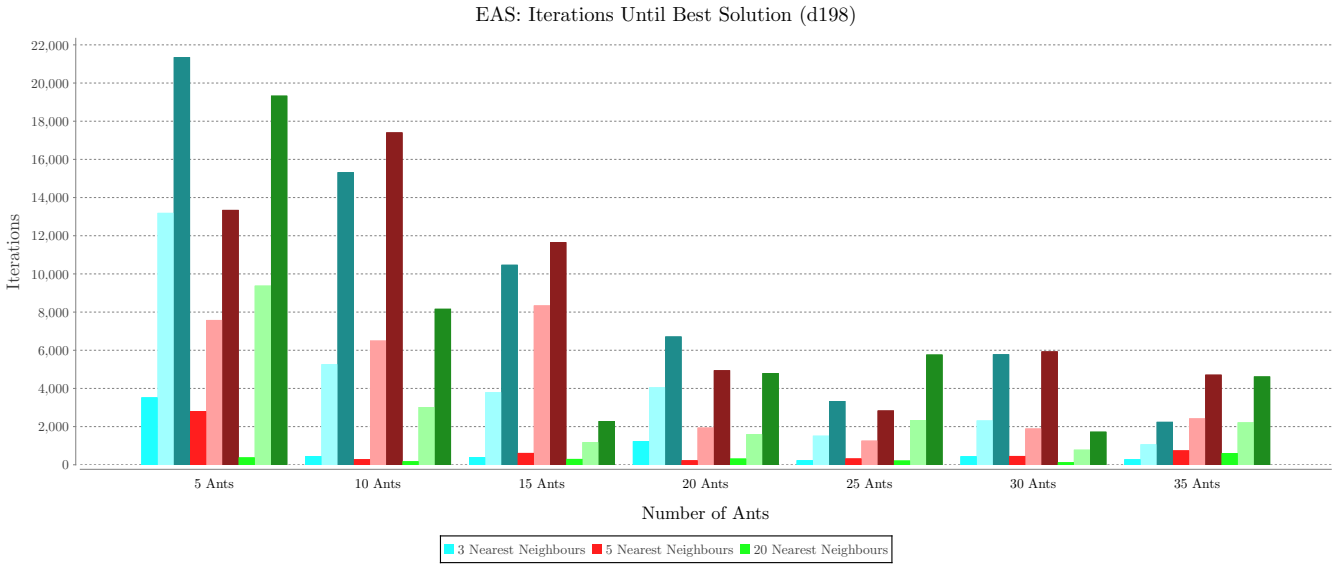

Figure 4.4

4.1.3 Convergence

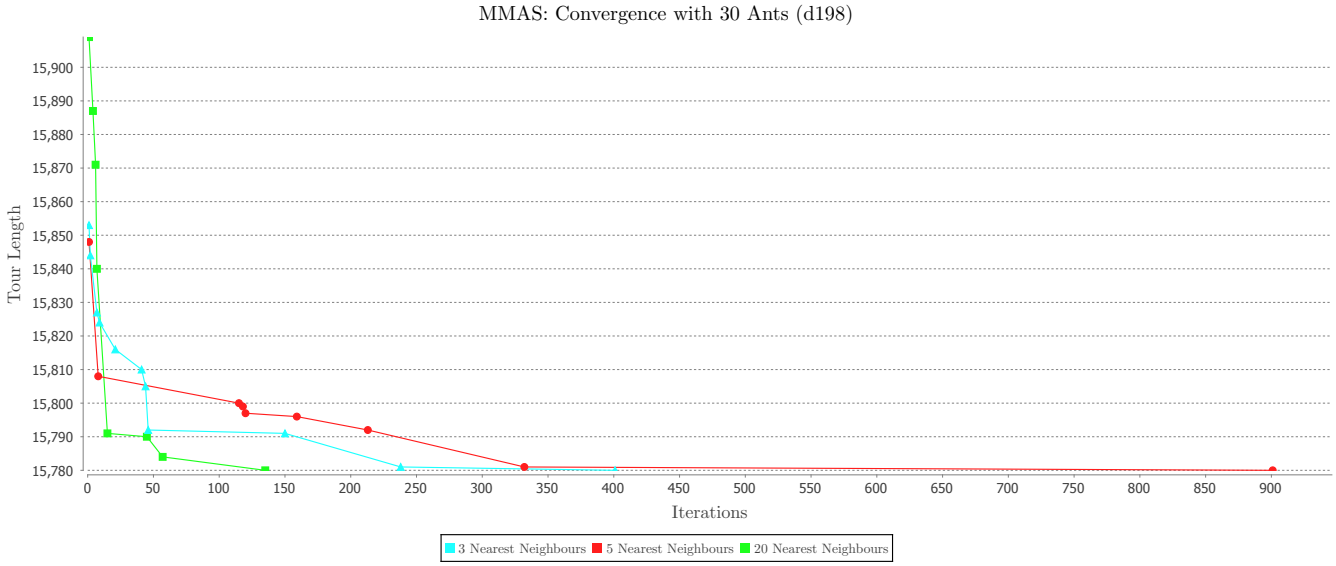

Figure 4.5

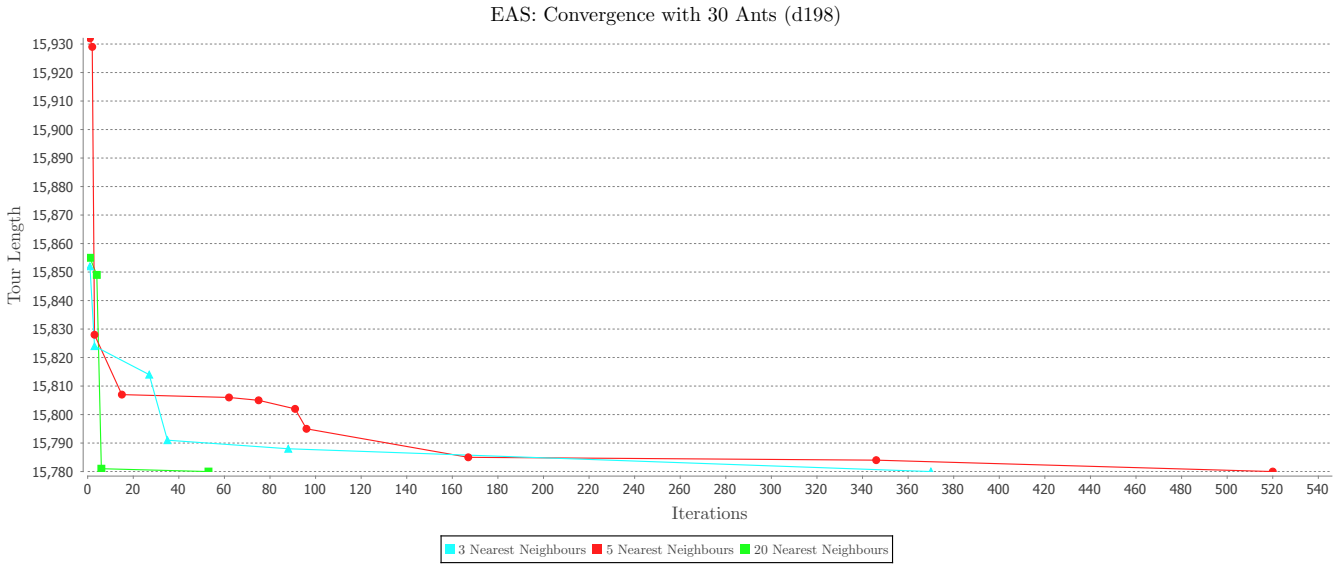

Figure 4.6

4.2 Medium Sized Instances

4.2.1 Runtime

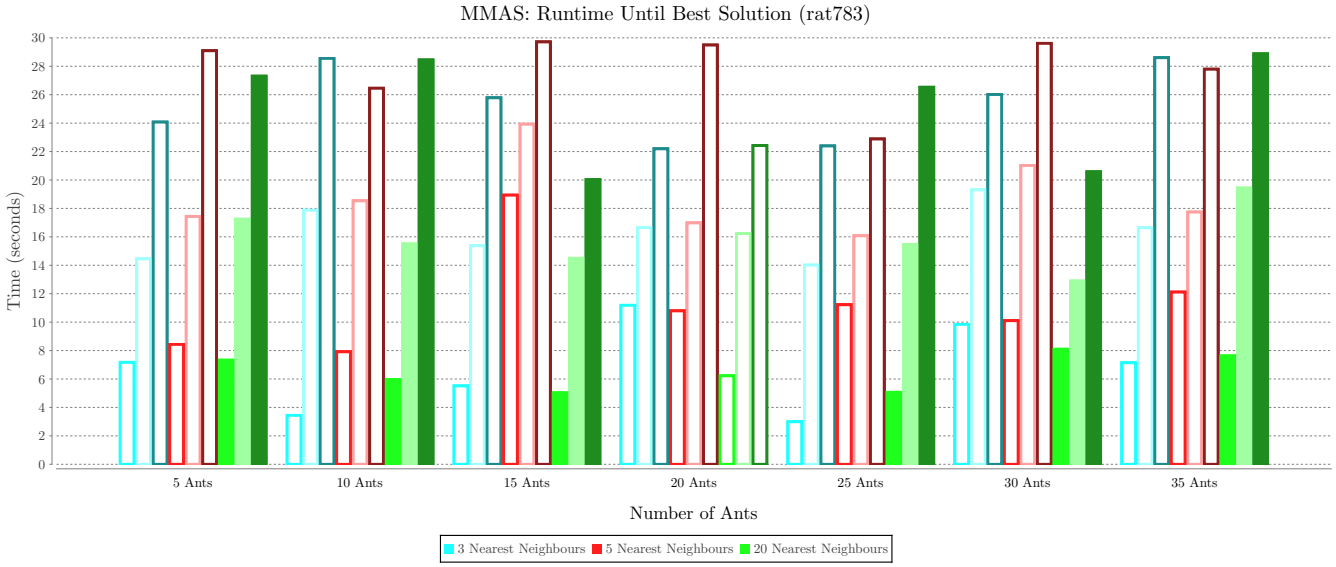

Figure 4.7

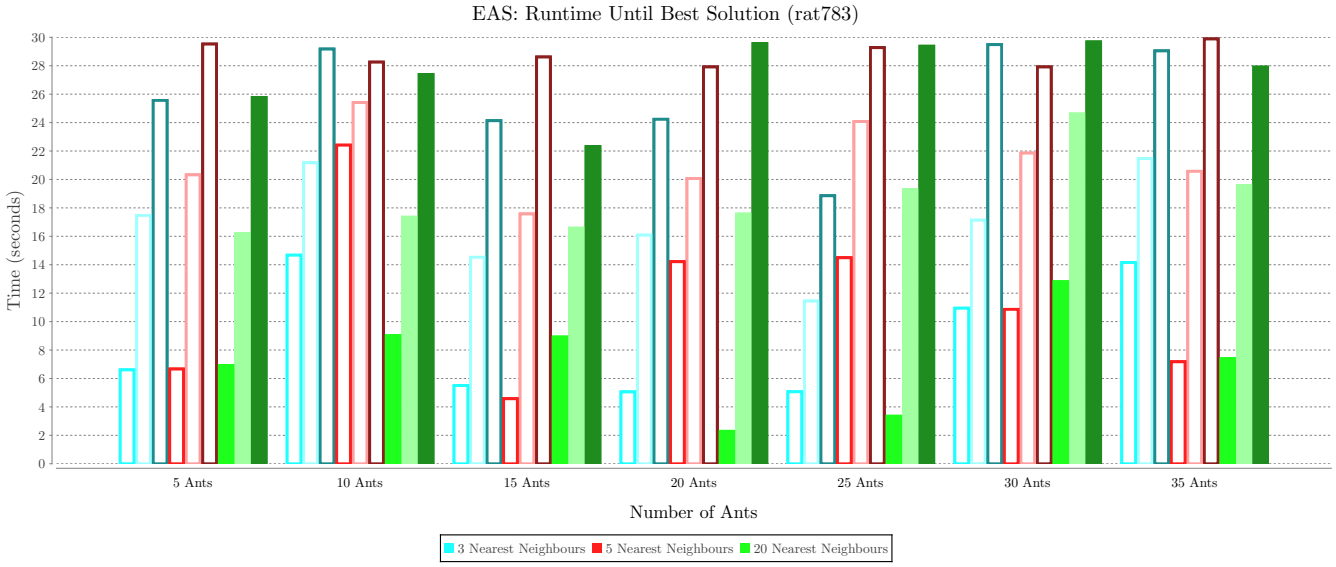

Figure 4.8

4.2.2 Iterations

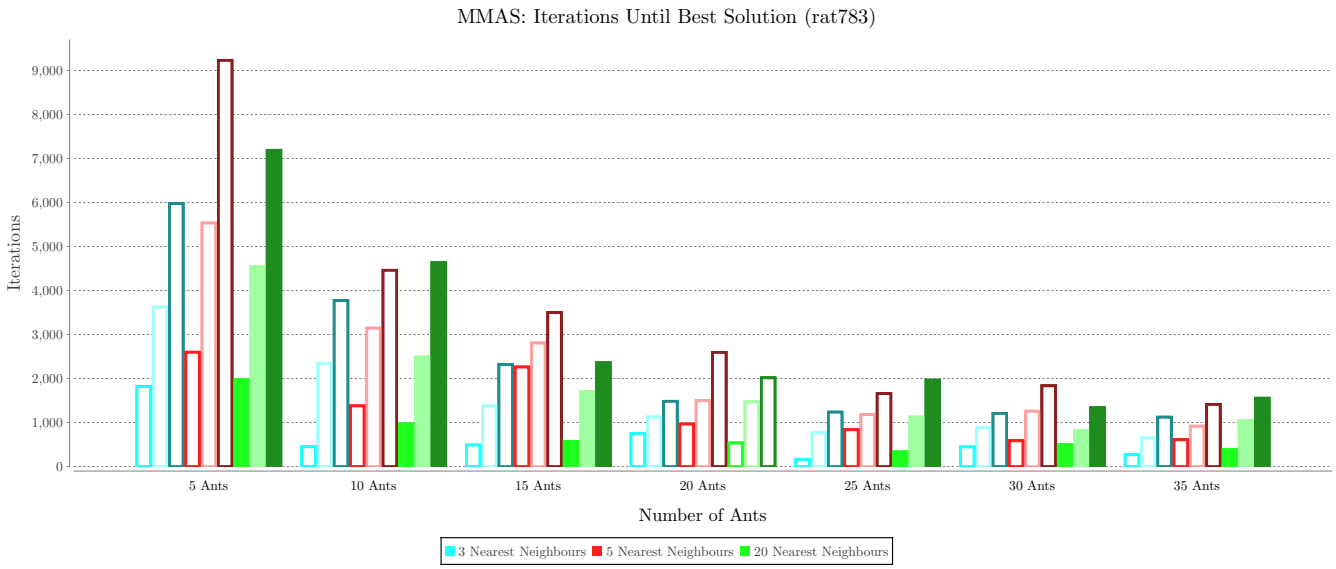

Figure 4.9

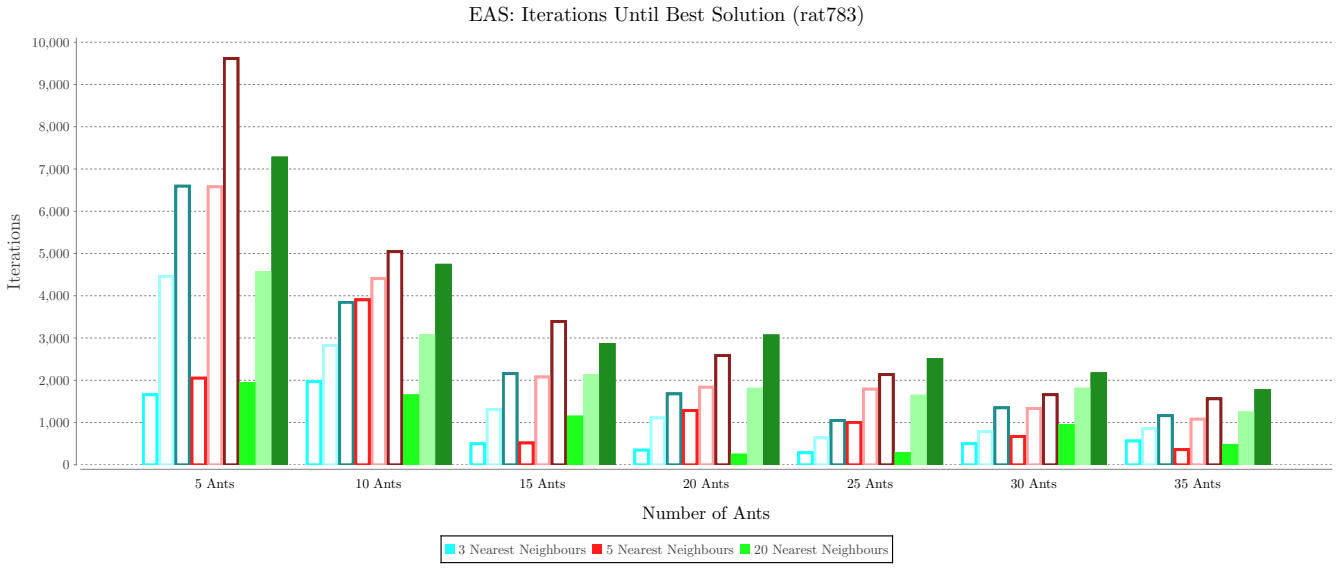

Figure 4.10

4.2.3 Convergence

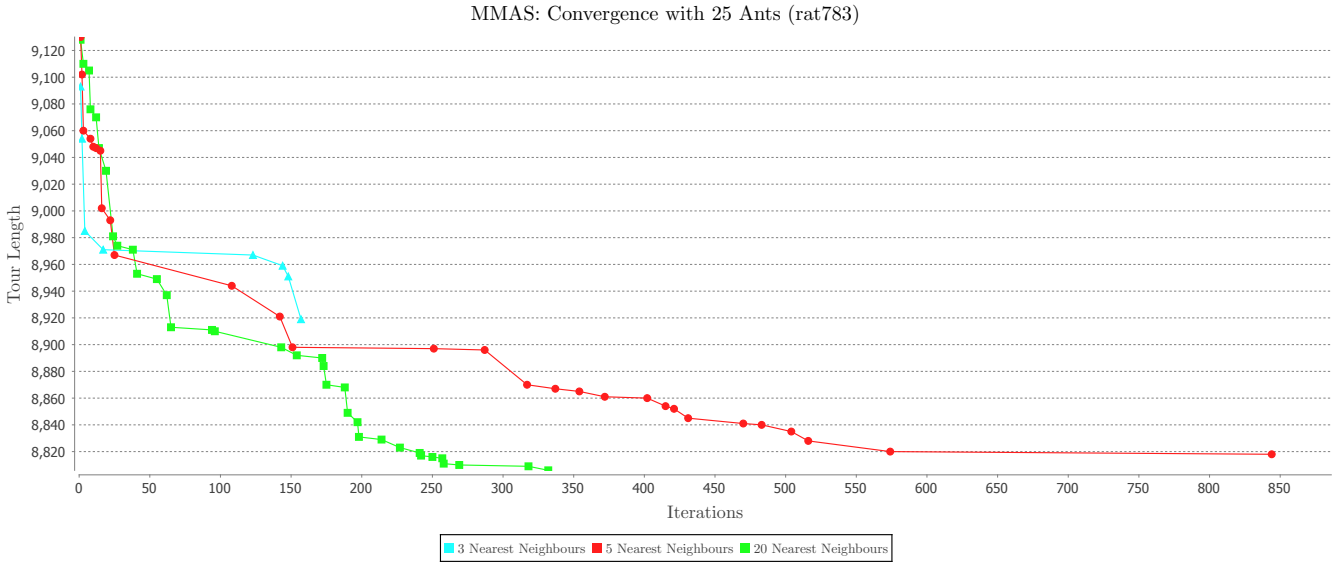

Figure 4.11

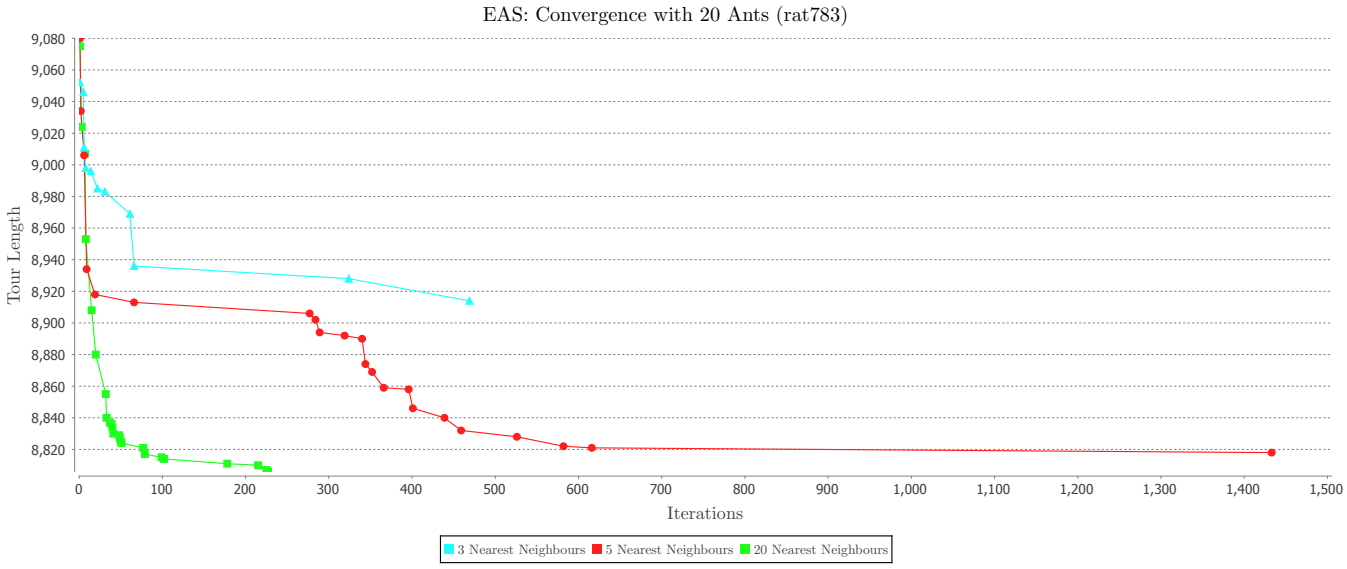

Figure 4.12

4.3 Large Sized Instances

4.3.1 Runtime

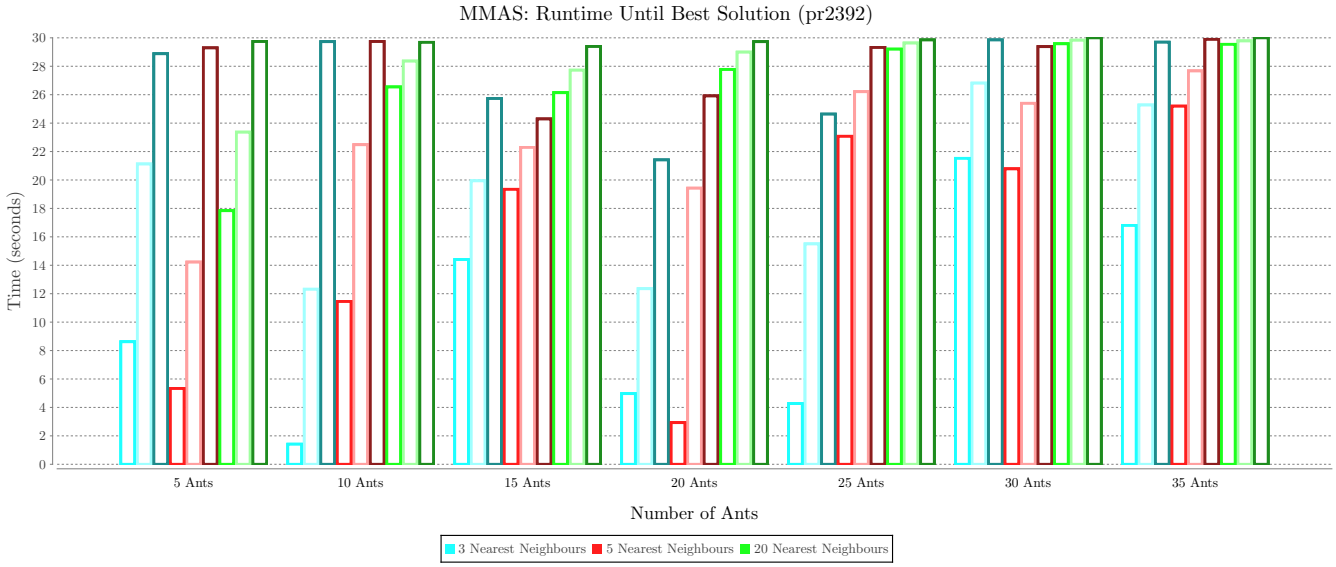

Figure 4.13

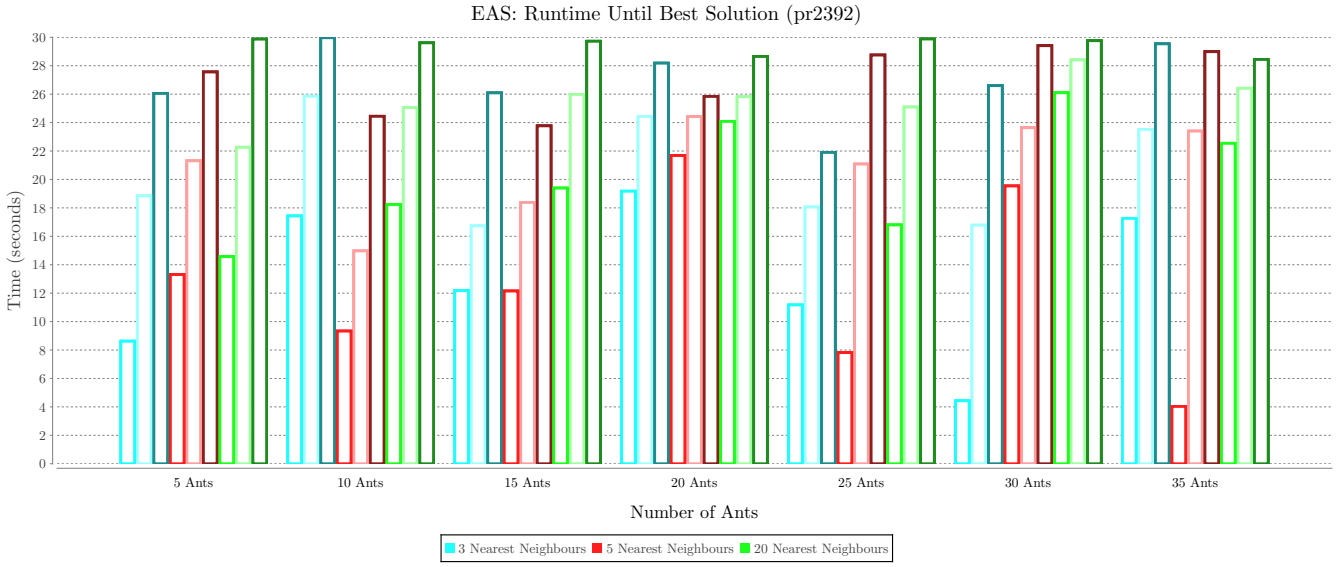

Figure 4.14

4.3.2 Iterations

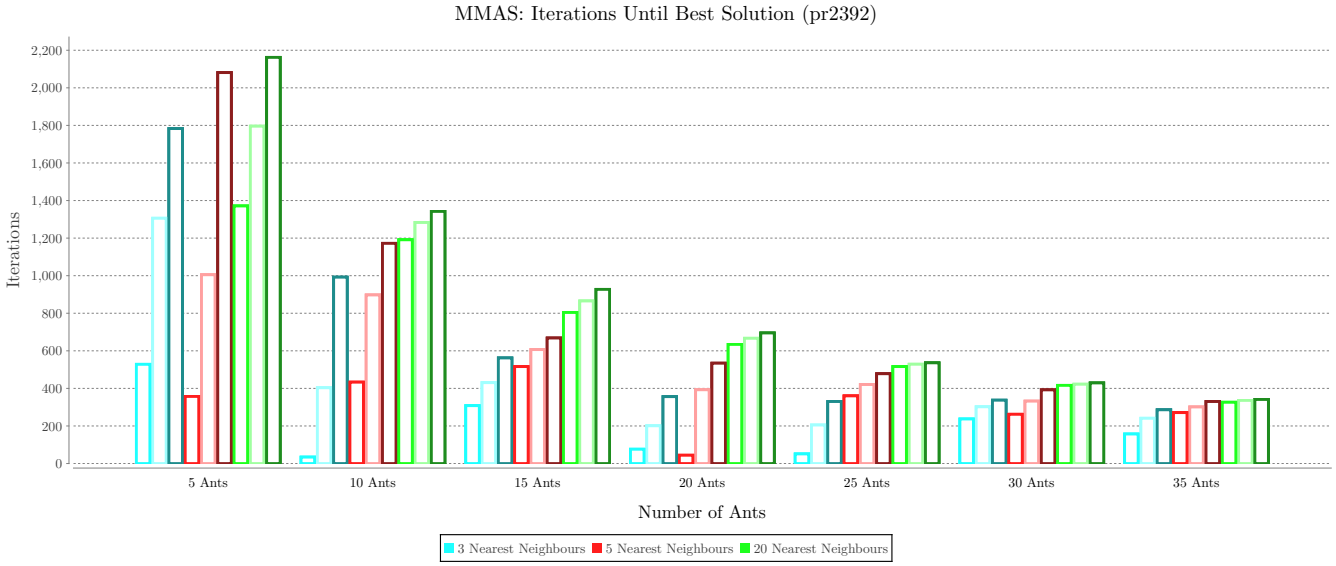

Figure 4.15

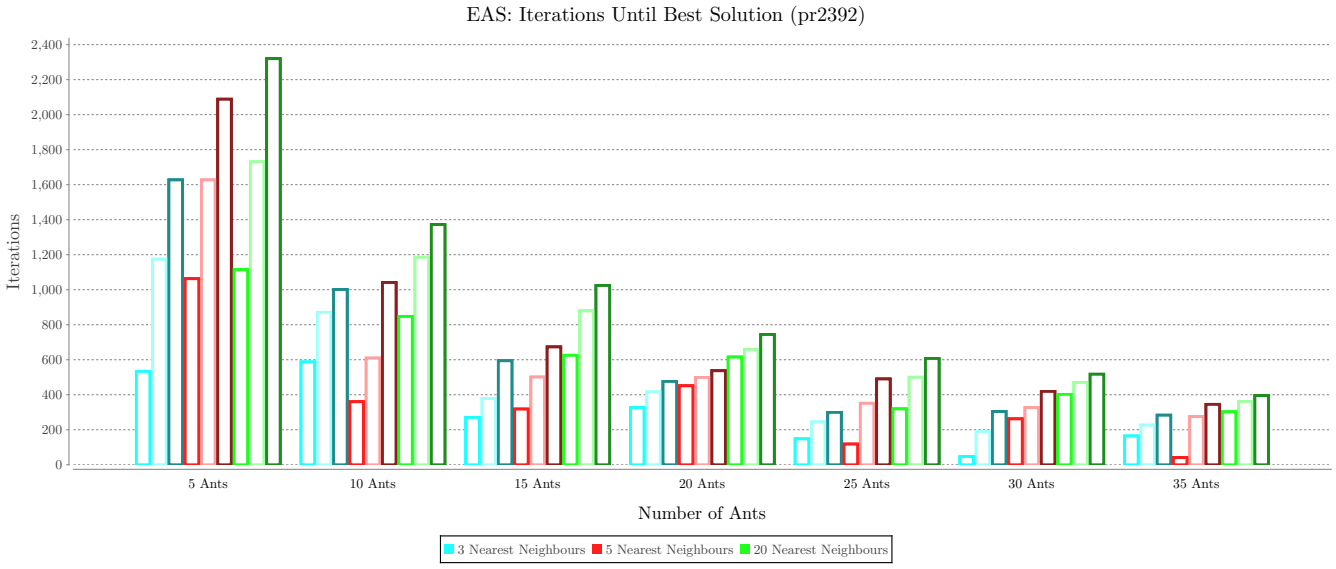

Figure 4.16

4.3.3 Convergence

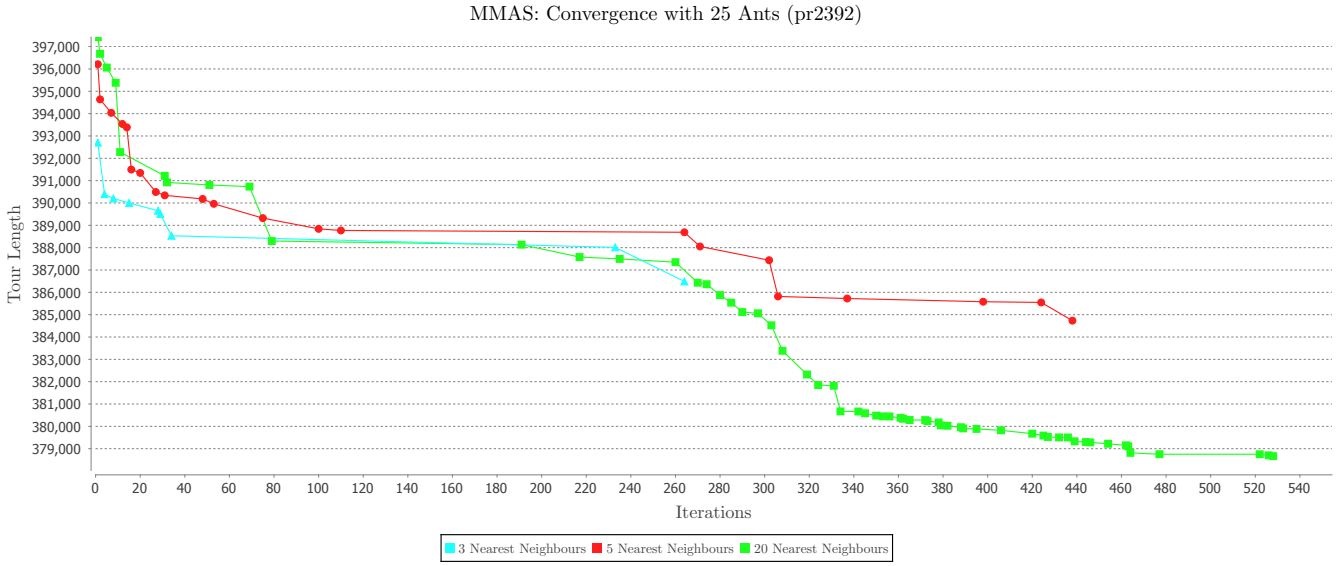

Figure 4.17

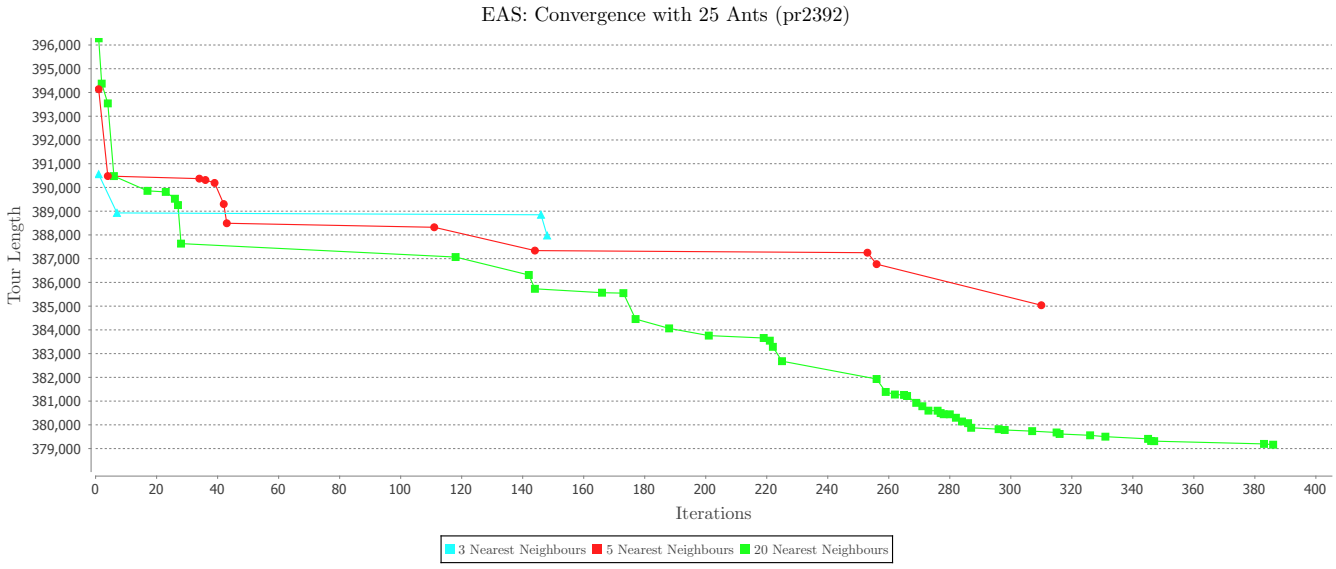

Figure 4.18

Chapter 5

Discussion

The discussion section focuses on the strengths and weaknesses of MMAS and EAS individually, and then compare the two based on performance and quality of the approximations for larger instances when neither of the pheromone techniques were able to find an optimal solution.

5.1 MMAS

It is possible to identify several trends in the results using the MMAS distribution technique. The first one being the number of ants required to achieve the best performance on different instance sizes. The number of ants does not have a noticeable effect on the iteration results of small instances once the number of ants exceeds 15 ants (Figure 4.3), which is similar to the results observed by Dorigo in his initial results using ANT-cycle. This points towards the similarities between MMAS and Ant System, because although they may handle pheromone distribution differently, the core of the algorithms remains the same. However we also measure runtime, which in this case differs depending on the number of ants. It shows that for 30 ants achieves the overall best result when considering 20 nearest neighbours, and 35 when the nearest neighbours were reduced (Figure 4.1). This is most likely because the algorithm runs faster in general when it considers fewer neighbours, thus a greater number of ants can be used without increasing the slowdown, and as seen from the iteration chart we want to use as many ants as possible because they find the optimal in fewer iterations. The convergence chart for

the small instance shows that the algorithm gradually finds better solutions, but suffers from stagnation when fewer nearest neighbours are used (Figure 4.5). It seems therefore like the neighbour limitation limits the algorithm's capability of finding alternate paths that also leads to an optimal solution. One possibility is that there are several optimal solutions and some of these are removed from the solution set once the amount of neighbours checked are limited. The algorithm therefore stagnates when there are fewer solutions that are optimal.

For medium sized problems the best result was found with 25 ants (Figure 4.7). Although 30 ants results in a better average for 20 nearest neighbours and has less spread (difference between worst and best). Although, 25 ants performs better when the number of nearest neighbours were reduced below 20. Additionally, 25 is the number of ants found to be optimal by Stützle and Dorigo as well (Stützle and Dorigo, 1999). The reason to why the algorithm performs better with a slightly higher amount of ants on smaller instances is very likely due to the runtime being more affected when each individual ant needs to traverse a longer tour each iteration.

When approximating the large problems it was found that the best solution within 30 seconds is achieved using 25 ants as well (Figure 4.13). An outliers was identified using 35 ants, because the best test resulted in the shortest path overall, but the average path found was worse than for 25 ants. This is largely due to the fact that the algorithm suffers from large slowdowns for increasing number of ants, and the outlier at 35 ants is most likely due to fortunate alternate paths found by the 35 ants in the early iterations of the algorithm.

5.2 EAS

Running EAS on the small instances of TSP, it was found that the best runtime was achieved with 30 ants (Figure 4.2). When restricting nearest neighbours, 25 ants was found to perform best. However, the results does not show a general trend for runtime on small instances, except that 35 ants results in slower runtimes for all different nearest neighbour settings. Interestingly there is a considerable trend in the iteration results (Figure 4.4). More ants results in less iterations

required to find the optimal solution. This is because the elitist ant's influence is reduced with more ants in the system, meaning that new best solutions are more likely to be found as there are more independent agents. Fewer ants means that the currently best found solution is more likely to stay as the best found solution. Thus the runtime results can be explained, because even though each iteration is slowed down by a higher number of ants simulated, less iterations are required throughout the run.

The trend regarding the way more ants impact the runtime is similar for medium sized instances. Although the turning point where more ants slows down the algorithm is earlier, at 25 ants, due to the instance size slowing down large scale simulations (Figure 4.8). Yet again, very small numbers of ants have the advantage of fast iterations, which allows them to improve the solution more times. For instance, 10 ants requires around 3000 iterations on average running on rat783, while 20 ants requires about 1750 iterations to find the optimal solution. However, 20 ants only runs slightly faster than 10 on average so the average performance difference is barely noticeable in our tests. Although, it is worth noting that 20 ants were observed to have a lower worst case running time than 10 ants, and the overall best time was considerably better with 20 ants.

As for the largest instance examined, the best approximation results was found with 25 ants. The convergence chart show that the algorithm has enough time to converge fairly close to the optimal solution, something that is missed while running with a higher number of ants (Figure 4.18). A higher number of ants essentially interrupts the algorithm while it is still finding new solutions, and thus it is not able to converge within the given timeframe. Lower number of ants instead struggle with finding good solutions even though they are able to run more iterations. This is mainly because the impact of the elitist ant is larger and not as many alternate paths are explored.

5.3 Comparison

When comparing the pheromone distribution techniques, the best number of ants found for each algorithm is used. Thus the two are

compared with respect to their optimal conditions, which essentially is what future work is interested in.

When looking at the performance for the techniques on small instances, MMAS finds the optimal solution in less time than EAS on average. However, MMAS do require more iterations than EAS even though they run with the same number of ants. This is probably because MMAS continually favour alternate paths, seeking new alternate solutions throughout, whilst EAS favours the absolute shortest path even from the very beginning. The convergence chart in Figure 4.6 shows that EAS finds a very short path initially and therefore favour it in future iterations, leading to a steep drop in convergence, followed by stagnation. When comparing this to MMAS chart (Figure 4.5) which gradually converges it is evident that their solving technique differs. This is largely the reason to why MMAS has a better overall quality while EAS displays a larger spread of results. The difference between the best and worst case in EAS is in fact due to the characteristic drop and stagnation, if no solution close to the optimum is found initially, EAS struggles because it continues to favour that solution. MMAS is therefore less prone to bad runs and is less dependant on a unfortunate start.

Both techniques share the fact that reducing the number of nearest neighbours considered limits the possible optimal solutions to the complete problem because some solutions may be removed when limiting the number of ways ants can travel. This results in worse performance overall. However, MMAS performs best with 10-15 ants compared to EAS which runs best with 25 ants. This is because the need to search for alternate paths with MMAS is not as great when there are in fact less of them. The best technique for the small sized instances with nearest neighbours being less than or equal to 5 seems to be EAS in terms of runtime. This is probably because it does not focus on finding as many alternate paths.

The techniques' results on medium sized instances shows that none of the techniques are able to find the optimal solution for the problem when the nearest neighbours are reduced. It is not possible to say whether they are simply too slow, or the optimal solution does not exist when less neighbours are used. Regarding the overall

performance it is clear that MMAS performs better than EAS in average. The fact that EAS relies too heavily on randomness is clear because it is less likely to find the optimal tour in a larger instance simply through random choices. However the absolute best run of EAS performs considerably better than MMAS, but the fact remains that this may be a fortunate run as the first iteration of EAS found a path 1000 units shorter than the best run for MMAS. MMAS can therefore be considered superior to EAS as it is more reliable for medium sized problems.

Regarding the approximation quality on large instances, our results found that MMAS is able to find a better approximation than EAS for all different nearest neighbours settings. The convergence charts show that both algorithms stagnates before the test is terminated, but this does not guarantee that neither of the distribution techniques are able to find an optimal solution. Using the runtime charts we see that EAS generally stagnates earlier, which corresponds to the drop and stagnate characteristic discussed earlier. The case may therefore be that EAS may be preferred when the quality of the solution is not as important and ACO is preferred to supply sub-optimal solutions faster, rather than better solutions that takes a bit longer.

Generally, the tradeoff between number of ants and fewer iterations seems more drastic for EAS. More ants reduce runtime by performing less iterations, while less ants require more iterations to complete, but reduces runtime due to each iteration running faster. This results in EAS performing better with less ants than MMAS, indicating that MMAS is slightly more inclined to be affected by iteration slowdown when there are more ants. This may be due to MMAS being a more complex algorithm which considers range of the pheromones in the graph independant on the number of ants.

Chapter 6

Conclusions

Our study has identified that the Elitist pheromone distribution technique performs better than MMAS on small instances with restricted path choices (≤ 5 edges from each node). However, other TSP instances are preferably solved with MMAS which performs better in cases when the path is not as restricted. MMAS is also favoured for medium sized problems of with all nearest neighbour settings studied in this thesis because it is more stable and predictable than EAS. EAS displays a drop-and-stagnate characteristic in its convergence toward an optimal solution, where the drop is largely dependant on the quality of the initial results. In the case when EAS is unfortunate, it can stagnate early which makes it struggle with finding new solutions.

The techniques displays different strengths for larger instances when ACO seeks an approximation because it can not find the optimal solution within a reasonable timeframe. A reasonable timeframe was in this case set to 30 seconds because of the test timeframe limitation mentioned in the Methods section. MMAS was observed to achieve the best approximations, although EAS was able to find fairly good solutions faster than MMAS. They are therefore good in different cases, EAS when sub-optimal solutions are fine and the application rather requires them faster, and MMAS for a slightly slower and more stable solving technique.

Furthermore, EAS have been observed to be more negatively affected by an increasing number of ants than MMAS. MMAS is therefore recommended in large scale simulations, for example Swarm AI

and multi agent systems. On the other hand, EAS performs better with fewer ants (< 20), so EAS is preferred if the implementation is restricted to a few ants, or AI agents.

Chapter 7

References

Blum, C. and Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys. ACM, Inc.

Cook, W. (2013). World Traveling Salesman Problem. [online] Math.uwaterloo.ca. Available at: http://www.math.uwaterloo.ca/tsp/world/ [Accessed 5 Apr. 2017].

Dorigo, M., Caro, G. and Gambardella, L. (1999). Ant Algorithms for Discrete Optimization. Artificial Life, 5(2), pp.137-172.

Dorigo, M., Colorni, A., Maniezzo, V. "Distributed Optimization by Ant Colonies", *Proc. European Conference on Artificial Life*. Paris, France. pp. 134-142, 1992.

Dorigo, M. and Stützle, T. (2009). Ant Colony Optimization: Overview and Recent Advances. IRIDIA – Technical Report Series. IRIDIA, Institut de Recherches Interdisciplinaires et de Dèveloppements en Intelligence Artificielle, Universitè Libre de Bruxelles.

"Jfreechart". *jfree.org*. Object Refinery Limited, 2014. Web. 28 Apr. 2017.

Mavrovouniotis, M. (2013). Ant Colony Optimization in Stationary and Dynamic Environments. Ph.D. University of Leicester.

Mohammadrezapour, O., Mohammad, J. Z. "Comparison of ant colony, elite ant system and maximum – minimum ant system algorithms for optimizing coefficients of sediment rating curve (case study: Sistan river)". *Journal of Applied Hydrology*, Iran, 2014.

Ramos, V., Fernandes C., C. Rosa, A. "Social Cognitive Maps, Swarm Collective Perception and Distributed Search on Dynamic Landscapes". *Brains, Minds Media – Journal of New Media in Neural and Cognitive Science*, NRW, Germany, 2005.

Raskin, I., Ribnicky, D.M., Komarnytsky, S., Ilic, N., Poulev, A., Borinker, N., Moreno, D.A., Ripoll, C., Yakoby, N. "Plants And Human Health In The Twenty-First Century". *Trends in Biotechnology* 20.12 (2002): pp. 522-531.

Stützle, T. ACOTSP, Version 1.3. Available from *http://www.aco-metaheuristic.org/aco-code*, 2004.

Stützle, T. and Dorigo M. "ACO Algorithms For The Traveling Salesman Problem". *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications*, John Wiley Sons, 1999.

Stützle, T. and Hoos H. H. "Max-Min Ant System". *Future Generation Computer Systems*, pp. 889–914, 2000.

Sörensen, K., Sevaux, M. and Glover, F. (n.d.). A History of Metaheuristics. In: R. Martí, P. Panos and M. Resende, ed., Handbook of Heuristics, 1st ed. Springer.

"Symmetric TSPs". *iwr.uni-heidelberg.de*. N.p., 2007. Web. 28 Apr. 2017.

Valdez, F., Chaparro, I. "Ant colony optimization for solving the TSP symetric with parallel processing", *Proc. Joint IFSA World Congr. NAFIPS Annu. Meeting*, pp. 1192-1196, 2013.

Widjaja, A. The Limits Of Computation. Melbourne University, 2004.