Ant Systems for Dynamic Problems
The TSP Case – Ants caught in a traffic jam

University of Twente
Casper Joost Eyckelhof
August 2001

Dr. J. Zwiers
Prof. dr. ir. A. Nijholt
Dr. M. Poel
Ir. M. Snoek

Summary

Ant System is an algorithm based on the foraging behaviour of real ants and it can be used to solve or approximate a variety of combinatorial optimisation problems. Most previous studies on Ant Systems have focussed on static problems. This report answers two questions about Ant Systems when applied to dynamic problems, the dynamic travelling salesman problem in particular.
On the one hand the suspicion by Bonabeau et al. [Bon99] that Ant System can be successfully applied to dynamic problems is confirmed. On the other hand a number of adaptations to the original algorithm are presented that improve the performance of Ant System on dynamic problems. The most important change is the introduction of 'shaking' into the algorithm. This is a technique that stimulates exploration in those parts of the problem that have been affected by a change in the environment. In our example the changes in the environment are traffic jams, that sometimes force the salesman to change his route.

Samenvatting

Ants System is een algoritme dat gebaseerd is op het gedrag van mieren wanneer ze voedsel zoeken. Het is zeer geschikt om allerlei combinatorische optimalisatie problemen mee op te lossen of in ieder geval te benaderen. Eerdere onderzoeken gingen meestal over het gebruik van Ant Systems voor statische problemen, maar dit verslag gaat over de toepassing op dynamische problemen, in het bijzonder het dynamische handelsreizigerprobleem.
Het vermoeden van Bonabeau etc. [Bon99], dat Ant System succesvol kan worden toegepast op dynamische problemen, zal worden bevestigd. Tevens zullen een aantal aanpassing op Ant System worden gepresenteerd die de prestaties van Ant System aanzienlijk kunnen verbeteren bij dynamische problemen. De belangrijkste verandering is het toevoegen van 'schudden' aan het algoritme. Dit zorgt ervoor dat in de gebieden waar de veranderingen optreden meer zal worden gezocht naar nieuwe oplossingen, in plaats van de eerder gevonden oplossingen te gebruiken. De veranderingen zijn files in onze voorbeelden. Deze dwingen de handelsreiziger soms om een andere route te kiezen.

Preface

About a year ago I was required to decide on a topic for my thesis. It did not take me very long to realise that I wanted to do something with “natural algorithms”. After reading a few books and articles on artificial life, social agents and genetic algorithms, I was given the hint to look at swarm intelligence and ant colony optimisation (ACO). That triggered my interest in ant system, one of the first applications in the field of ACO. It was very simple and elegant, but already ‘over-researched’. Many versions have been applied to all kinds of optimisation problems. However, all those studies have focussed on static problems, though various authors suspected that ant system could be applied to dynamic problems very well. So that is what I did: attempting to prove this supposition and try to make a version of ant system that was even better on dynamic problems.

My fascination for ant systems mostly comes from the fact that they are so simple, yet still can be applied to numerous problems by just changing some minor details. I have thought a while about designing a wizard-like application that builds an ant system for a certain problem, just by asking the user some basic questions about the problem.

After a slow start, the committee pushed me on track by setting some deadlines. And every time I thought the research got stuck, they handed me many hints about what might have gone wrong and what to do about it, and I always left our meetings full of fresh ideas. In particular the dialogues with Marko Snoek during the last month were very inspiring; one time we planned to have a half hour meeting - to discuss some minor details - that ended up lasting over two hours leaving me with a weeks worth of plans and him with a full whiteboard.

First of all I would like to thank the committee for their support.
There are many people whom I had discussions with regarding my thesis, but I would especially like to thank Alex Lubberts and Remco van de Meent, both of whom are working on their thesis as well.

My thanks also go out to a few people for proof reading this document: Brandan Yares, Ben Brooks and Wieger Opmeer

I have learned many things about doing research this year, ranging from small technical details to motivating myself.

Casper Joost Eyckelhof

Table of Contents

1 Introduction

In nature there are many systems that consist of very simple parts, but can have great complexity as a whole. An example of this is ant colonies: every single ant just seems to walk around independently, however the colony itself is organised very well. This effect is also known as emergent behaviour. Their foraging behaviour shows that ants can travel in a very optimised way between their nests and sources of food. Based on this phenomenon, ant colony optimisation (ACO) algorithms have been developed [Stu99], [Bon99], [Dor99], of which *ant system* was the first [Dor96]. These ACO algorithms have been used for all kinds of combinatorial optimisation problems, but they were always for static problems: the problem itself did not change while solving it.

For static problems ACO algorithms have proven their use, but little is known about the behaviour of ant system on dynamic problems.

The goal of this research is twofold: to confirm the hypothesis that ant system can be used on dynamic problems and to try to enhance the original ant system algorithm to perform better on dynamic problems. As a benchmark, a dynamic version of the well-known travelling salesman problem will be used.

We will give a brief introduction to both ant system and the dynamic travelling salesman problem. After that we will introduce some changes to ant systems that should improve performance on dynamic problems. This is followed by several experiments to test our theories and we will see that the proposed changes do improve the performance.

2 Travelling Salesman Problem

2.1 Classic Travelling Salesman Problem

The travelling salesman problem (TSP) is a well-known problem among computer scientists and mathematicians. The task basically consists of finding the shortest tour between a number of cities, visiting every city exactly once.

A more formal definition:
Given a set of n nodes and distances for each pair of nodes, find a roundtrip of minimal total length visiting each node exactly once. The distance from node i to node j is the same as from node j to node i. [www1]

It is also possible to discard that last condition and allow the distance from node i to node j to be different from the distance between node j and node i. We refer to that case as the asymmetric travelling salesman problem.

The travelling salesman problem was probably first published by the mathematician and economist Karl Menger in the 1920's [www2] and became popular after some time as a prototype for a class of problems called NP-hard [Rus95].

The popularity of TSP probably comes from the fact that it is a very easy problem to understand and visualise, but it is very hard to solve. Many other problems in the NP-Hard class are not only hard to solve, but also hard to understand. With TSP, having n cities, there are (n-1)! possible solutions for asymmetric TSP and (n-1)! / 2 possible solutions for symmetric TSP. For small instances it is no problem to generate all solutions and pick the shortest, but because the number of possible solutions 'explodes' when the number of cities increases, we need smarter ways to solve TSP, or at least find acceptable solutions.

And the search for reasonable solutions is another way to look at TSP and other hard optimisation problems: rather than trying to find the best solution, find an acceptable solution in a (much) shorter time than would be needed to actually solve it. It is a trade-off between the costs involved in finding the best solution and using a sub-optimal solution.

2.2 Dynamic Travelling Salesman problem

There are several reasons to look at the dynamic case of problems, and TSP in particular. The most important is that dynamic problems are more realistic usually: real world problems are hardly ever completely static. Another reason to look at dynamic problems is that they are even harder to solve than their static counterparts, which makes them more interesting as well.

There is another definition for the travelling salesman problem that uses the term *cost* instead of *distance*, which may be a better way to look at the dynamic travelling salesman problem.

The Dynamic Travelling Salesman Problem (DTSP) that was used in this project is a variation on TSP that also knows traffic jams; the distance between nodes is not constant, but changes over time. One could say that the road becomes longer but that is quite unlikely when you try to map the problem back to the real world. Somehow the costs to travel between two nodes increases, and the most natural metaphor for this would be increased travel time or traffic jams.

Another example of a dynamic travelling salesman problem is a situation in which cities are inserted and/or deleted from the map. A group of researchers from the University of Karlsruhe, Germany has been working on the application of Ant System on that form of dynamic travelling salesman [Gun01], [Gun01-2].

3 Classic Ant Systems

3.1 Introduction to Ant Colony Optimisation systems

When studying social insects like wasps, bees and in particular ants, one can see that the colony itself is very organised, though it seems that the individual insects are just walking around, doing their own things. Somehow in the interaction between those very simple entities, patterns emerge. The term used for this phenomenon is self-organisation (SO)[Kau95].

Self-organisation is basically a set of dynamical mechanisms where structures appear at a higher level because of interactions between components at a lower level. This means there is no supervisor to guide the lower level components. As a result of this, the decisions made by these components are solely based on local information. They do not have access to the global knowledge contained in the system.

Systems like this are found in nature at various places and on different levels. This is an indication that algorithms based on it might be very powerful: through natural selection only systems (colonies, animals, systems within animals) that are both efficient and robust keep on living. If they are not efficient enough, another species will take over and they have to be robust to keep on living when the environment changes.

With ants, which have been quite successful in earth's evolution, we see self organisation happen at various levels within the colony, one being foraging: when searching for food, networks of trails are formed between the nest and various sources of food. Natural barriers are usually no problem because the ants find their way around them. These networks of trails are usually very efficient. That is one of the main reasons to look at foraging ants as an example for optimisation techniques.

According to [Bon99] there are four important aspects of SO, given here with ant specific examples:

1. Positive feedback. To reinforce the better solutions, some kind of positive feedback is needed. For ants looking for food, trail laying is a positive feedback. The pheromone trails get 'bigger' towards food sources and more ants will go there, which will increase the amount of pheromone even more.
2. Negative feedback. This is needed to counterbalance the positive feedback and stabilise the collective pattern. Examples are the exhaustion of food sources and the limited number of available foragers. In the ant systems we will see later, this negative feedback is embedded in the algorithm by evaporation of pheromone trails (the trails get smaller over time).
3. Amplification of fluctuations. Some random factor is often crucial for finding better solutions. New, even better, food sources can be discovered if an ant sometimes tries a new path, instead of following old, but safe, ones.
4. Multiple interactions. SO only works if individuals can interact with each other. Ants use each other's pheromone trails to interact. The trails are some kind of collective memory.

The information about the problem is stored in the environment itself in the form of pheromone trails. When ants change the environment by changing a trail, they are indirectly talking to each other. This interaction is called stigmergy, and is one of the most important aspects of ant colony optimisation. It is the main reason that all the individuals operate as a colony.
As said before, the paths that foraging ants create can be very efficient. So when trying to imitate the behaviour of the ant colony in an algorithm, it is a natural first application to do a path optimising problem: TSP.
But many other problems have been successfully attacked using ant colony optimisation (ACO): e.g. Quadratic Assignment Problem (QAP) [Man94], Graph Colouring Problem [Cos97], Vehicle Routing Problem [Bul99], Sequential Ordering Problem [Gam97], Shortest common Sequence Problem and various job scheduling problems. Another field in which Ant System has been recently applied is data mining [Par01].

These algorithms are mostly academic, but according to an article in the Daily Telegraph [www5][1] Unilever is already using ant algorithms in one of their plants to "*find the best locations for and movements between the storage tanks, mixers and packing lines in a large Unilever factory"*.
The same article speaks of several other commercial parties planning to use ant based systems in the (near) future.

3.2 The Ant System algorithm

In their book [Bon99], Bonabeau et al. give a good explanation of Ant System (AS), the algorithm we will use as a basis for our own algorithm.
Their version of the (high level) algorithm will be presented in this section and the most important parts of it will be explained.
It has become common practice to name the algorithms with their field of application, so this version of Ant System is called AS-TSP.

[1] The article is mirrored via [www3] because we are not sure how long the online archives will be available.

/* **initialisation** */
For every edge (i,j) **do**
$\tau_{ij}(0) = \tau_0$
End For
For k = 1 to m **do**
Place ant k on a randomly chosen city
End For
Let T^+ be the shortest tour found from beginning and L^+ its length
/* **Main loop** */
For t = 1 to t_{max} **do**
For k = 1 to m **do**
Build tour $T^k(t)$ by applying n-1 times the following step:
Choose the next city j with probability

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

$$p_{ij}^{k}(t) = 0, \text{if } j \notin J$$

where i is the current city.
End For
For k = 1 to m **do**
Compute the length $L^k(t)$ of the tour $T^k(t)$ produced by ant k
End For
If an improved tour is found **then**
Update T^+ and L^+
End If
For every edge (i,j) **do**
Update pheromone trails by applying the rule:
$\tau_{ij}(t) \leftarrow (1 - \rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t) + e \cdot \tau^{e}_{ij}(t)$ where

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t),$$

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q / L^k(t), if\,(i,j) \in T^k(t) \\ 0, otherwise \end{cases}$$

and

$$\Delta\tau_{ij}^{e}(t) = \begin{cases} Q / L^+, if\,(i,j) \in T^+; \\ 0, otherwise, \end{cases}$$

End For
For every edge (i,j) **do**
$\tau_{ij}(t+1) = \tau_{ij}(t)$
End For
End For
Print the shortest tour T^+ and its length L^+
Stop
/* **Values for parameters** */
α=1, β=6 , ρ = 0.5 , m=n , Q = 100, $\tau_0=10^{-6}$, e = 5

In Ant System all ants construct (candidate) solutions. They do this based on two main components: pheromone trails and a heuristic, which is *visibility* in TSP. Visibility is defined the inverse of the distance. ($\eta_{ij} = 1/d_{ij}$)
At the start all possible roads are initialised with a certain amount of pheromone (τ_0). After that all ants (one at the time) construct a solution by choosing the next city based on pheromone levels (τ) and visibility (η) until they have visited all cities exactly once.

The choice for the next city is a probabilistic one. The more pheromone there is on a certain road the bigger the chance that that road will be taken. The same goes for visibility: closer cities have a higher chance of being next. Cities that have already been visited have a zero chance of being visited again, because of a blacklist mechanism. The parameters α and β control the relative weight of pheromone trail intensity and visibility. If α=0 the algorithm behaves as a standard greedy algorithm, with no influence of the pheromone trails. If β=0 only pheromone amplification will occur and the distance between cities has no influence on the choice. Usually a trade-off between these two factors is best.

When all ants have completed one tour, the amounts of pheromones are updated. On every road some pheromone "evaporates" and on all roads that have been visited by at least one ant new pheromones are deposited. That amount of pheromone that is deposited is based upon the number of ants that visited the road and the length of the tour the road is part of. The shorter the tour, the more pheromone is deposited. This amount is also influenced by the (constant) parameter Q: a higher Q means more pheromones. The amount of pheromone that evaporates is solely based on the parameter ρ, which is the percentage that evaporates.

All roads on the best tour so far get an extra amount of pheromone. This is called elitist ants: the best ants can reinforce their tour even more.

This process goes on until a certain condition is satisfied (number of iterations, amount of CPU time, certain solution found).

The number of ants (m) is equal to the number of cities (n) in our version of the algorithm

3.3 An implementation of AS-TSP

To get a feeling for the algorithm at the beginning of this research, we made a quick sample implementation[2] and tried to repeat some of the experiments mentioned in [Bon99].
These experiments consist of running AS-TSP on problems from TSPLIB, a collection of travelling salesman problems [www1]. In 'Appendix I: Some results of classic AS-TSP' some results of our experiments are shown.
For the Eil-51 benchmark we reached a better score than any experiment in [Bon99] (423 vs. 425), for KroA100, which is a much bigger problem, we came quite close to

[2] In Borland Delphi: that language was chosen because it is very easy for rapid prototyping.

the best score in [Bon99] (23145 vs. 21282). Especially for KroA100 we did not do as many runs.
All these experiments were mainly done to get the feel of AS-TSP and to better understand its internals. We were not aiming to get great scores, although the fact that we came close to the results of [Bon99] does indicate that we did not make any implementation faults.

Figure 1 shows the prototype in action on the Eil-51 benchmark. It is has only run for a few seconds and just started its 8th loop. Current best score is 471 found by ant number 48.

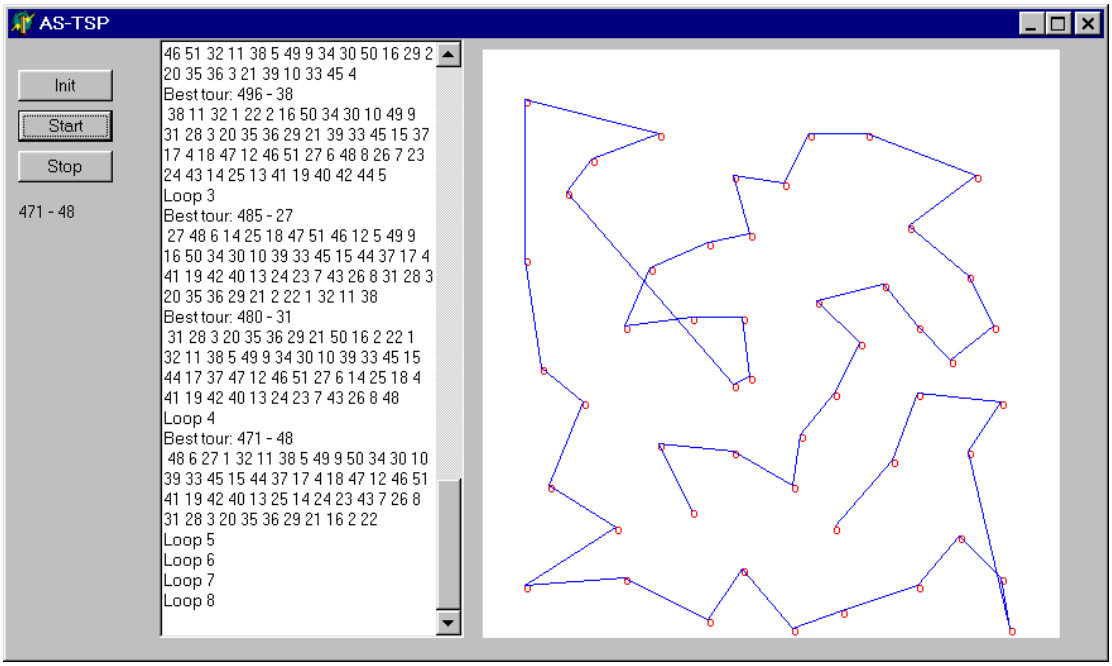


Figure 1

It also has to be mentioned that the algorithm used in [Bon99] was a slightly modified version of AS-TSP, called Ant Colony System (ACS-TSP). This algorithm is AS-TSP modified on four points: transition rule, pheromone trail update rule, local updates of pheromone trails and the use of a candidate list. These modifications result in an increase in exploration and a better performance on large problems, according to the results.

To get even more acquainted with AS-TSP we did some experiments with changing parameters too. These results are also presented in Appendix I: Some results of classic AS-TSP.

4 Ant Systems for DTSP

4.1 Changes to Ant Systems for dynamic problems

In their book [Bon99] Bonabeau et al. say they believe that ant based algorithms should perform very well on dynamic problems (they mention TSP with the insertion or deletion of cities) but that the conjecture has not been confirmed. Preliminary studies by Guntsch et al. [Gun01] show that this is indeed true. Quoting from their conclusions: "*Overall, it can be said that even when simply keeping the pheromone matrix, ant algorithms seem to be well suited to cope with dynamic optimisation problems."*

We acknowledge their conclusion, but we will see for ourselves whether the same is true for a slightly different kind of dynamic TSP.

When trying to use Ant Systems on dynamic problems, one encounters a few problems that might prevent it from performing very well. As said before, in theory AS should be very flexible and might even be able to cope with a changing environment, but as we will see that is not always exactly the case. Actually there is one problem: the way AS works by positive reinforcement of good solutions causes it to discard certain bad solutions after running for some time. The negative feedback component of the algorithm, evaporation, slows down this process in order not to constrain the search to a few good solutions too soon: the evaporation of pheromones prevents certain roads from becoming extremely attractive for ants. Without evaporation there could be an amplification of pheromones on just a few roads that happened to be found right at the start of the algorithm.
If the algorithm would focus on only a few solutions too soon, it would probably get stuck in a local optimum: there is very little exploration and only exploitation of those few early solutions.
The combination of positive and negative reinforcement works well for static problems: there is exploration, especially in the beginning, but after a while all roads that are not promising will be slowly cut off from the search because they do not get any positive reinforcement and all pheromones have evaporated over time.
In dynamic situations however, certain solutions can be bad before a change in the environment and they might be good afterwards. But because they were considered bad before that change, the evaporation has already minimised the chance of those roads being chosen by ants. However, the environment has changed, and those roads might be very promising in the new situation. During our research this was one of the first things we ran into.

In this chapter we will discuss the evolution of Ant Systems in order to try to make it perform better on dynamic problems, DTSP in particular.

4.2 Ant System for Dynamic Travelling Salesman Problem

4.2.1 4 city problem

When starting the research on DTSP we first picked the smallest possible non-trivial problem: a 4-city travelling salesman problem. Having fewer than 4 cities is not useful, because there is only 1 possible solution for 1, 2 or 3 cities[3]. The 4-city problem is actually quite interesting already.

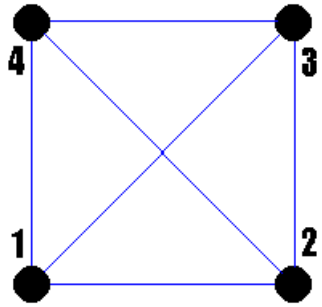


Figure 2

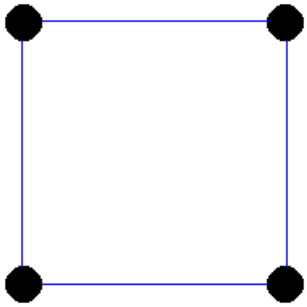


Figure 3

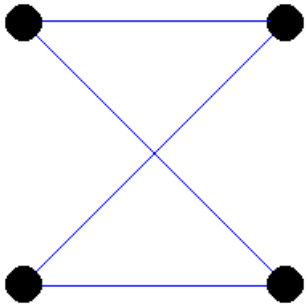


Figure 4

We assume that the cities are positioned in a square with edges of length 10, as shown in Figure 2. All cities are connected to each other. The shortest tour for the non-dynamic case is shown in Figure 3 and has length 40, following the outer-edges. If we increase the length of one of the edges, the shortest tour will change to the one shown in Figure 4, the 'hourglass' shaped tour[4]. The transition between these optimal tours occurs when the edge with increased length gets longer than 18.28. The length of the "hourglass-tour" is $2*10 + 2*10*\sqrt{2} = 48.28$.

We tested the standard Ant System algorithm on the following dynamic problem:

- *The map used is the one shown in Figure 2, with edges length 10 and the diagonals length $10\sqrt{2}$ (Cartesian length).*
- *Every 5th round the length of edge 2-3 is increased by 1.*

This means that in the ideal case, the solution would change from Figure 3 to Figure 4 immediately after the 45th round, when the length of the square tour becomes 49. But even after 150 rounds the solution found by AS was the square-solution, having reached a length of 69.

Looking at the internal state of the algorithm, we discovered that the chances for an ant to follow the diagonals during their tour had become 0 after some time. This always happened before the diagonals became a viable solution, after round 45 in this case.

A small example of what the four ants were "thinking" during round 45:

Round: (45)
45.1 (The ant that starts on city #1)
City = 1 Chance array = [0,50,0,50]. Value =62
City = 4 Chance array = [0,0,100,0]. Value =79

[3] This is only true for symmetric TSP. Asymmetric TSP only is non-trivial from 3 cities and up.
[4] When looking at this figure at a 90 degree angle, it looks more like a butterfly than an hourglass.

City = 3 Chance array = [0,100,0,0]. Value =45
45.2 (The ant that starts on city #2)
City = 2 Chance array = [97,0,2,0]. Value =63
City = 1 Chance array = [0,0,0,100]. Value =46
City = 4 Chance array = [0,0,100,0]. Value =37
45.3 (The ant that starts on city #3)
City = 3 Chance array = [0,2,0,97]. Value =23
City = 4 Chance array = [100,0,0,0]. Value =71
City = 1 Chance array = [0,100,0,0]. Value =59
45.4 (The ant that starts on city #4)
City = 4 Chance array = [50,0,50,0]. Value =6
City = 1 Chance array = [0,100,0,0]. Value =83
City = 2 Chance array = [0,0,100,0]. Value =36
Shortest tour in round 45 by ant 1: 49.0. Route: 1, 4, 3, 2

To explain the way one should interpret these traces[5] let's look at the 3rd line ("*City = 1 Chance array = [0,50,0,50]. Value =62"*).
The array, having 4 positions, defines the chance of the ant going to the city with the number of the position: the first value in the array is the chance of going to city number 1 from the current position. So we see that the ant that started on city number 1 has 0% chance of going to city 1[6], 50% chance of going to city 2, 0% chance of going to city 3 and finally 50% chance of going to city number 4: [0,50,0,50].
This is also illustrated in Figure 5.

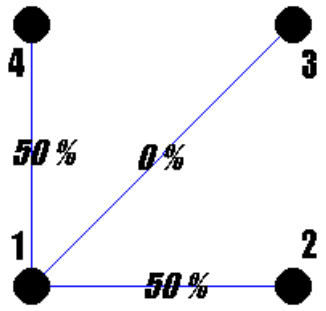


Figure 5

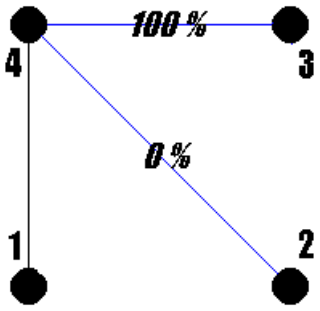


Figure 6

The ant chooses the next city based on the random value: it sums the chance-values in the array until this value is greater or equal to the random value. As soon as that happens, it chooses that city. Because the value in this case is 62, the ant chooses to go to city number 4.
An explanation using the numbers (*[0,50,0,50]. Value =62*):
0 < 62 not to city number 1;
0+50 < 62 not to city number 2;
0+50+0 < 62 not to city number 3;
0+50+0+50 >= 62 the ant goes to city number 4

[5] All numbers in the traces are rounded to an integer value; small rounding errors can occur.
[6] The blacklist mechanism, explained in 3.2 prevents the ant to go to cities where it has been before this round. The ant starts on city number 1, so the chance of going to city number 1 will automatically be 0.

Being in city number 4, the choice is very easy according to the 4th line of the trace: there is a 100% chance of going to city number 3. (Figure 6)
After this, there are no chances anymore, because of the nature of TSP, the solution is now known: 1-4-3-2.

4.2.2 Non-zero chances

The (near) zero chances are caused by the way the algorithm cuts unsuccessful branches from its search. This is fine for static problems, but bad when the problem changes over time.
The level of pheromone decreases by a factor two ($\rho = 0.5$) on every edge that is not visited by any ant during the last tour and isn't on the best tour ever,

And because the chance for ant k to go from i to j at time t is given by the formula

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in J_i^k} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}}$$

it is clear that a rapidly decreasing value of τ_{ij} for a road from i to j will cause the chance for an ant k being in i to go to j to become almost zero: the numerator will be zero, or very close to zero. (For a more detailed explanation of this formula, see 3.2)

The first thing that came to mind in preventing the chances dropping to zero, was setting a lower boundary on the amount of pheromone on every edge. This prevents the chances to go all the way down to zero, so every road will have a chance of being taken, although this chance can still be very low.

The value used for this lower boundary is τ_0. This value was chosen for two reasons:

- The initial value of pheromone is there for exactly the same reason: don't let the chances be zero. It only seems logical to use the initial value as lower boundary.
- Empirically: using τ_0 as a lower boundary seemed to do what we wanted after running a few tests.

4.2.3 Shaking

4.2.3.1 Introduction

Another change to AS-TSP we introduced in AS-DTSP was what we called 'shaking'. It is a technique in which the environment is 'shaken' to normalise the pheromone levels in a certain way.

This is because after a while the amount of pheromones on certain edges would be extremely high compared to some other edges. This is fine for the static case, but again a problem for dynamic problems. If the amount of pheromones on an edge becomes much higher than all other edges going out of a city, this edge will almost certainly always be chosen. That is a way for the static case to ensure a good road will always be followed, but it prevents ants from taking another road when a traffic jam occurs.

Looking again at the formula that describes the probability for an ant to take a certain road, we see that a high value for τ_{ij} (pheromones) can have a big impact on the result. If the ratio between τ_{ij} and $\tau_{i'j'}$ is 10^6, this road i,j has a very high probability to be chosen over i',j', even if the visibility is bad because of a traffic jam[7].

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in J_i^k} [\tau_{il}(t)]^{\alpha} \cdot [\eta_i l]^{\beta}}$$

When we shake the environment i.e. normalise all edges, this effect is reduced, but we still profit from the information that was stored in the pheromones: if an edge 'a' contained a higher value of pheromones than edge 'b' before shaking, edge 'a' will still have a higher value after shaking. This has two advantages: after shaking all roads have a chance of being chosen again, which is good for exploration, but not all information from the past is thrown away.

4.2.3.2 Global shaking

The formula used for global shaking is a logarithmic one:

$\tau_{ij} = \tau_0 \cdot (1 + \log(\tau_{ij} / \tau_0))$

This formula will cause the value of τ_0 to stay the same and higher values to get much closer to τ_0 again. Note that τ_0 is the minimum value for τ, forced by the algorithms, so $\tau_{ij} \geq \tau_0$. Figure 7 shows the values of τ_{ij} before and after shaking using the given formula.

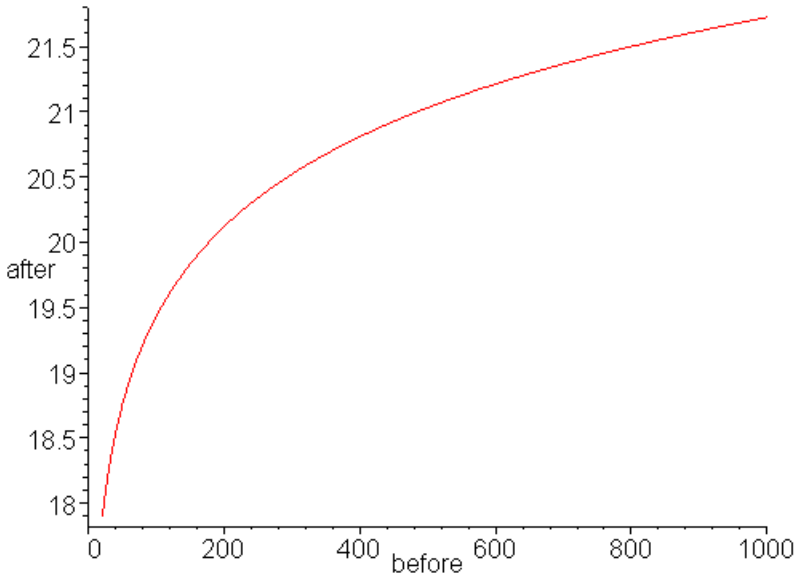


Figure 7

[7] Traffic jams increase the 'distance' between two cities, hence they reduce the visibility.

To illustrate this some values are compared in Table 1.

Table 1

τ_{ij} **before**	τ_{ij} **before** / τ_0	τ_{ij} **after**	τ_{ij} **after** / τ_0
10^{-6} ($=\tau_0$)	1	10^{-6} ($=\tau_0$)	1
10^{-3}	1000	$0.79 \cdot 10^{-3}$	7.9
1	1000000	$0.148 \cdot 10^{-4}$	14.8
10	10000000	$0.171 \cdot 10^{-4}$	17.1
1000	1000000000	$0.217 \cdot 10^{-4}$	21.7

It is obvious that when the amount of pheromone on a certain edge would be 1000 units, the τ_{ij} before value is 1000000000 times τ_0. After shaking it is still higher than τ_0, but only 21.7 times higher: the extreme differences between the amount of pheromone on the edges are gone.

4.2.3.3 Local shaking

A possible problem with the global shake is that too much information is lost in the shaking process. When we have a big problem where one traffic jam occurs somewhere near the edge of the map there is a high probability that only the route in the vicinity of the traffic jam will change [stu99]. Shaking the whole environment might have the wrong effect. This is mostly an "educated guess" for DTSP and has to be proven with experiments. For these experiments we changed the global shake to be able to only normalise edges within a certain distance of the traffic jam.

The same formula is used, but it is only applied to edges that are closer than p · MaxDist to one of the two cities where the traffic jam is formed between.

The parameter p is a percentage (we will use 0.1 and 0.25 in the experiments). Notice that p = 1 will give us exactly the same thing as global shake.
MaxDist is the maximum distance between any two cities in the original problem (without traffic jams).

In pseudo code:

```
If the distance between 2 cities (a , b) changed then
    For every edge (i,j) do
        If (dist(a,i) < p · MaxDist) ∨ (dist(b,i) < p · MaxDist) ∨
           (dist(a,j) < p · MaxDist) ∨ (dist(b,j) < p · MaxDist) then
           τij = τ0 · (1 + log (τij / τ0) )
        End If
    End For
End If
```

4.3 Performance measures and benchmarks

The biggest problem in evaluating our results is the complete lack of benchmarks. For static TSP there are benchmarks available: usually just the best scores and sometimes the time it took to reach those scores with a particular algorithm (on particular

hardware). For dynamic TSP there are no such benchmarks. This means that we had to choose our own criteria to measure the performance of an algorithm.

We did not investigate time and space results. We obtained the results of the algorithm in some kind of black box: the algorithm ran a number of iterations and we did not measure the time it took for a run to complete or the amount of memory it took[8].

There are some obvious things to look at for dynamic problems. Because the problem changes we want our algorithm to respond to the changes rapidly. In our tests this would mean that when a traffic jam is created the ants quickly find a route that does not contain the edge with the problem. Because all traffic jams are introduced in a few steps, this is visible in the graphs as low peaks: before the traffic jam is at its highest point, the ants have already changed their route. If traffic jams would be put into the problem at once, we would observe a very high peak or just a small bump in the graph. E.g. we have a traffic jam of 100 units. If we would introduce it at once, we would see a peak that is 100 units higher than the last value before the traffic jam or the ants compensated it within one tour, and we would see a small peak, because the new route is very likely to be longer than the one before the traffic jam. If we introduce traffic jams in a few steps, which we do, we see a peak that goes up during a short time, and at some point the ants find a better path and the length of the tour goes down again. E.g. if the traffic jam grows in 10 steps of 10 units, we might observe that peaks are 30 units higher than before the traffic jam using algorithm A and 50 units higher using algorithm B.

After the ants changed their route to avoid the traffic jam, usually some more parts of the route have to be adjusted: a single change has impact on a bigger part of the route. This means that we would like to see a high (negative) gradient after a peak, meaning that some more optimisations are being performed.

And finally the values for the length of the tour should be as low as possible. We **are** trying to find the shortest possible tour, so we do want to find short tours. In highly dynamic problems, this would mean a low average value. When discussing the results of various experiments, we will explain which values were exactly compared to each other.

Another thing that we did was comparing different versions of AS-(D)TSP to each other. That way we can at least say that certain versions of the algorithm are better (in specific situations) than others.

The versions we compare are:

- The original AS-TSP, which we called ‘Nop’ (nothing special happens during execution)
- Reset: this is an original AS-TSP, but when traffic jams occur, we reset the complete pheromone matrix to τ_0.
- Shaking: both global shake and local shake (two versions) are tested.

[8] While implementing the various algorithms, the time/space trade-offs were always decided for time and against space. Memory was never a problem during the entire research.

All of the above can be run with and without elitist ants, i.e. setting the parameter *e* to 0 (no elitist ants) or 5 (with elitist ants). This results in 10 different test cases, which have all been run 10 times. After 10 runs the average values are calculated and plotted.

Another problem in benchmarking is the type of problems that we use. Both the size of the problem and the dynamics can have big influences on the performance of algorithms. Especially the time between changes can have a big influence on the way the algorithms respond. If the changes are closer together than the algorithm can compensate for we will see a different behaviour than when there is just an occasional change in the environment.
For the size we tried a few problems with a different number of cities. For the dynamics of the problems we chose a situation with only a few changes and a problem that was constantly changing. Every problem that changes over time is a dynamic problem, but both the number of changes and the size of each change can vary. One issue that we did address was the fact that the problem would behave the same way every time it is run. This is the only way to compare various algorithms in a fair way.

4.4 Resulting algorithm

In this chapter we have introduced a few techniques to add to AS-TSP to try to improve its performance on dynamic TSP. These adaptations result in the following high level algorithm for AS-DTSP. This one is still based on the classic Ant System Algorithm as explained in chapter 3.2 and [Bon99].

/* **initialisation** */
For every edge (i,j) **do**
  $\tau_{ij}(0) = \tau_0$
End For
For k = 1 to m **do**
| Place ant k on city k
End For
Let T^+ be the shortest tour found from beginning and L^+ its length
|**Let** MaxDist be the maximum distance between any pair of cities

/* **Main loop** */
For t = 1 to t_{max} **do**
  For k = 1 to m **do**
    Build tour $T^k(t)$ by applying n-1 times the following step:
    Choose the next city j with probability

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

$$p_{ij}^{k}(t) = 0, \text{if } j \notin J$$

    where i is the current city.
  End For
  For k = 1 to m **do**

Compute the length L^k (t) of the tour T^k (t) produced by ant k
End For
If an improved tour is found **then**
Update T^+ and L^+
End If
For every edge (i,j) **do**
Update pheromone trails by applying the rule:
τ_{ij} (t) ← (1 - ρ) · τ_{ij} (t) + $\Delta\tau_{ij}$ (t) + e · τ^e_{ij} (t) where

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t),$$

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q / L^k(t), if\,(i,j) \in T^k(t); \\ 0, otherwise, \end{cases}$$

and

$$\Delta\tau_{ij}^{e}(t) = \begin{cases} Q / L^+, if\,(i,j) \in T^+; \\ 0, otherwise, \end{cases}$$

End For
For every edge (i,j) **do**
| τ_{ij} (t+1) = Max (τ_{ij} (t), τ_0)
End For
|**/*** external influences on the environment ***/**
|**If** the distance between 2 cities (a , b) changed **then**
| **For** every edge (i,j) **do**
| **If** (dist(a,i) < p · MaxDist) ∨ (dist(b,i) < p · MaxDist) ∨
| (dist(a,j) < p · MaxDist) ∨ (dist(b,j) < p · MaxDist) **then**
| τ_{ij} = τ_0 · (log (τ_{ij}) + correction) where correction = 1- log (τ_0)
| **End If**
| **End For**
|**End If**
End For
Print the shortest tour T^+ and its length L^+
Stop
/* **Values for parameters** */
α=1, β=6 , ρ = 0.5 , m=n , Q = 100, $\tau_0=10^{-6}$, e = 5, p=(0.1 or 0.25)

We will summarise the differences between AS-TSP and AS-DTSP briefly.
All lines with changes are marked in the algorithm by a vertical bar ('|') in front of it.

- Placement of ants. This is not done randomly as suggested, but evenly over the available cities (one ant on every city). This is done to spread the pheromones slightly more evenly, especially at the start[9].
- Added MaxDist variable. This variable is needed by the shaking-part of the algorithm. It defines the biggest distance available between any two given cities in the original problem (no traffic jams).
- Minimum τ level. An edge will never contain less pheromone than τ_0

[9] It is easier to implement too, and has no (bad) influence on the results, especially when using the same number of ants as there are cities (m=n).

- Shaking in case of traffic jam. When the environment changes, i.e. the distance between two cities is altered, shaking occurs. Only the amount of pheromones on edges that are ‘close enough’ to the location of the change will be normalised.

Some of these changes have ideas in common with *MAX-MIN* Ant System (MMAS) [Stu99]. MMAS was developed to create an ants based algorithm that has a better performance on (static) problems. They used several techniques to accomplish these better results, one of which was allowing pheromone levels only to be in a certain interval, hence the min-max.
The introduction of a minimum τ level is taken from MMAS. Shaking has a different effect than using a τ_{max} value, but is similar. A τ_{max} value will never allow the pheromone level to get very high, and shaking adjusts the pheromone matrix when a change occurs in the environment.

5 Experiments

5.1 Introduction

A problem with testing a new problem is the complete lack of benchmarks. Because there have been no other (public) studies into this area, it is impossible to compare our results to others. There is no benchmark problem defined for DTSP. This is why we compared different versions of the algorithm on self-constructed problems. The first problem consists of 25 cities. After looking at the results and drawing some conclusions from them, we will scale up the problem to 100 cities.

5.2 The prototype: about the software used

The prototype used to perform all tests is written in Java[10] for the biggest part. The main reason we still call it a prototype is that not everything is configurable in the user interface: some things have to be changed in the source code (e.g. generation of traffic jams and the choice what kind of algorithm to use).

Although Java has a bad reputation for speed, in our test it wasn't really important: the difference between waiting 11 seconds or 20 seconds for a result is not critical. The full source is available via [www3].

The Java program, called DTSP, creates two files for every run: a data file containing the results of the run and a script file for gnuplot[11] [www4], to be able to generate graphs from the data very easily. The format of those two files is described in Appendix III: Data files created by DTSP.

The data files are also used as input for a perl script that combines multiple data sets into other data files that contain average values, standard deviations and corresponding gnuplot scripts. This way we could easily combine multiple experiments into one graph and if necessary add more experiments later without any problems.

Because of the small size of the script and to give the reader insight in the way the combined graphs are created, we put it in Appendix IV: perl script. This is also available for download via [www3].

We will not go into detail on the Java program: it is basically a fully Object Oriented implementation of AS-TSP and AS-DTSP, containing classes for GUI, environment, cities, tours, calculation and visualisation. A screenshot is in Figure 8. While the program is running, it can graphically show the current best solution in a window like Figure 9.

[10] The choice for Java was made because that is currently the only fully supported language in the faculty.

[11] gnuplot is a command-driven interactive function plotting program. It can be used to plot functions and data points in both two- and three-dimensional plots in many different formats, and will accommodate many of the needs of today's scientists for graphic data representation. gnuplot is copyrighted, but freely distributable; you don't have to pay for it. (source: gnuplot FAQ, see [www4])

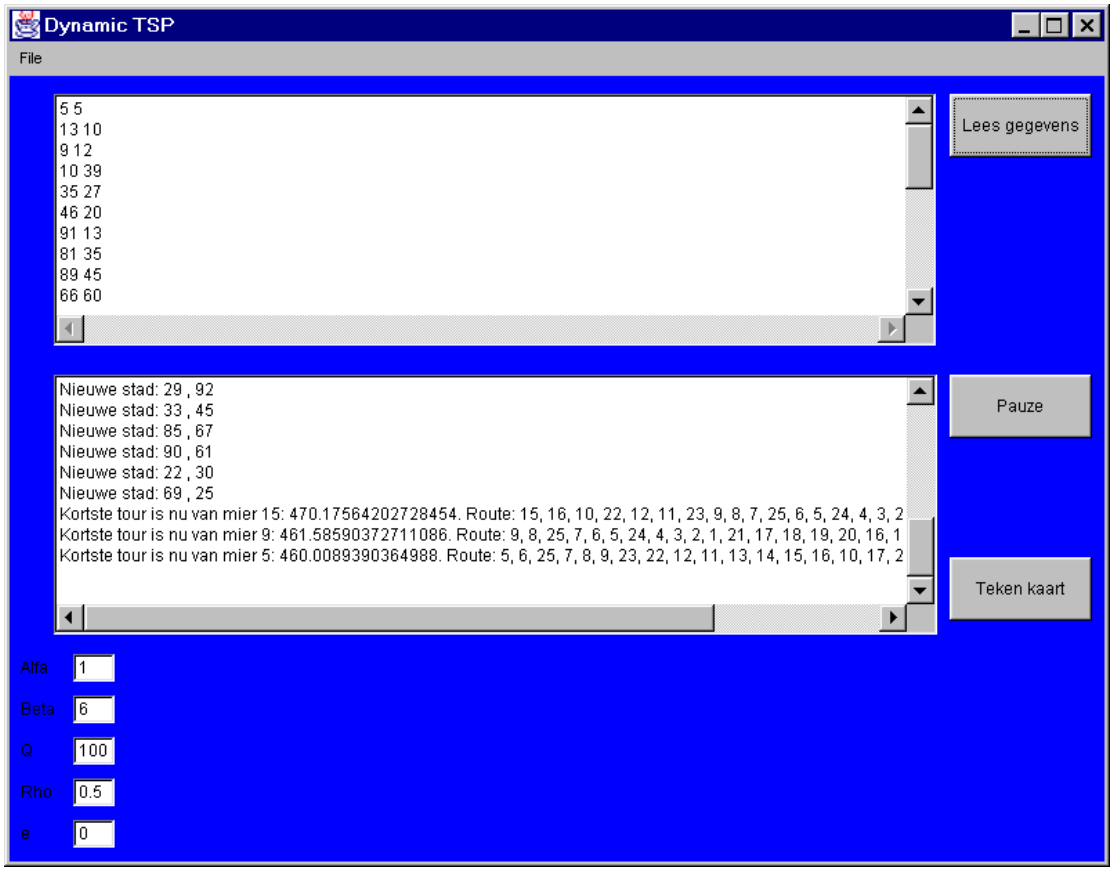


Figure 8

5.3 The test problem: 25 cities

A problem was constructed with 25 cities on a 100x100 grid. This was done by randomly[12] placing dots on a piece of paper and converting it to a map that our implementation of the algorithm understands.
All cities are located on an integer intersection of the gridlines. At certain points in time, traffic jams occur between cities that are very close to each other and are likely to be on the ideal route. These locations have been chosen empirically by running the initial problem through a classic AS-TSP implementation. This way we found certain pairs of cities that are almost always next to each other ("roads") in the best solution found. Using this technique we are sure that all traffic jams have influence on the best solution so far.
In the problem there are two interesting intervals: from iteration 100 to 150 a traffic jam is introduced between the cities with indices 17 and 21. From iteration 200 to 250 the first traffic jam disappears and 2 others are introduced between the cities 11,12 and 2,3.

All traffic jams appear and disappear in increments and decrements of 10 units, one incremental step every 5 iterations[13]. This way the algorithms get a better chance to

[12] Randomly: The author used his own idea of randomness, but this probably is biased by all kinds of factors. The 100 city problem was made using the computers random functions.
[13] A graphical representation of this process can be seen in Figure 12.

respond to small changes. If a traffic jam is introduced at once, an algorithm can not show that it is able to switch its route even while the traffic jam is still growing.

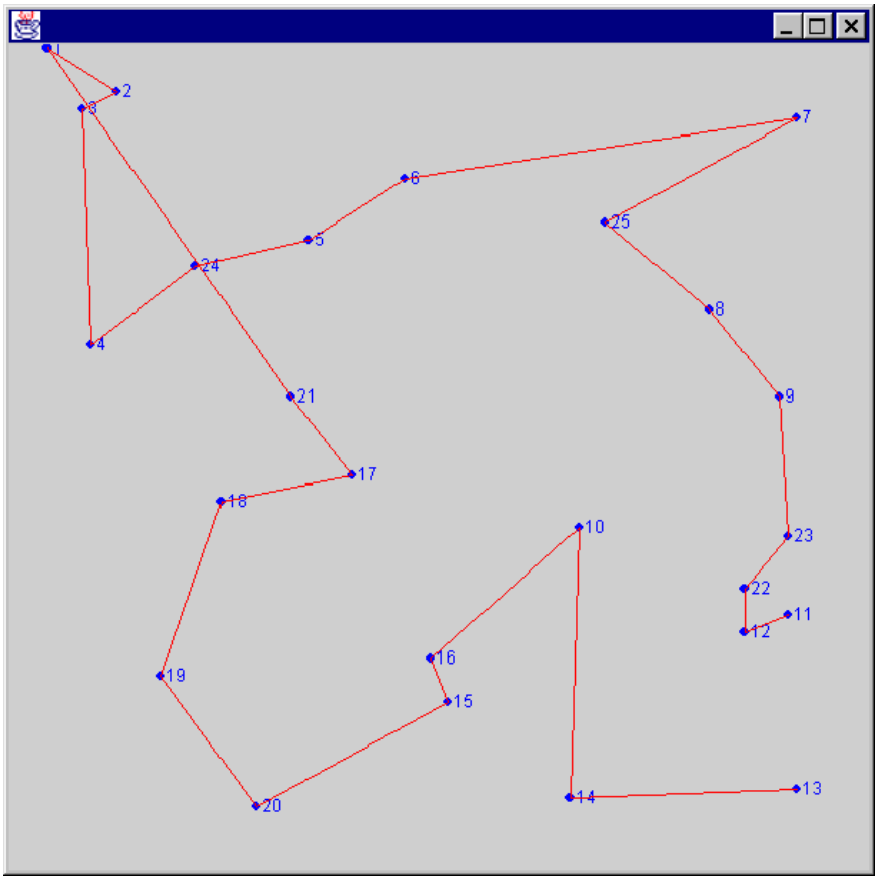


Figure 9

Figure 9 shows a graphical representation of the 25-city problem that we used. It is taken at the beginning of a run. The best solution so far clearly isn't anywhere near optimal. The exact locations of the cities in the grid, is given in Appendix II: The 25-city problem.

5.4 Results on 25-city problem

When running the problem without any traffic jams, the average solution after 500 iterations is 441.4. It would be nice if we get to that same score when running our algorithms on the problem with traffic jams. After the last traffic jams are formed, there are 250 iterations left to reach that value.

But the two points to focus on are located around 100 and 200 iterations: those are the dynamic parts of the problem. At both times, we see a peak in the graphs and those peaks should be as low as possible. Two other interesting values that say something about the capability to recover from changes are the point right before the second peak and the value after the last iteration.

Five interesting values are shown in Table 2 for all 10 situations. The corresponding graphs are in Appendix V: Full results for 25 cities. To indicate where the 5 points are located the graph for 'Shake, elite' has been edited to show them (Figure 10).

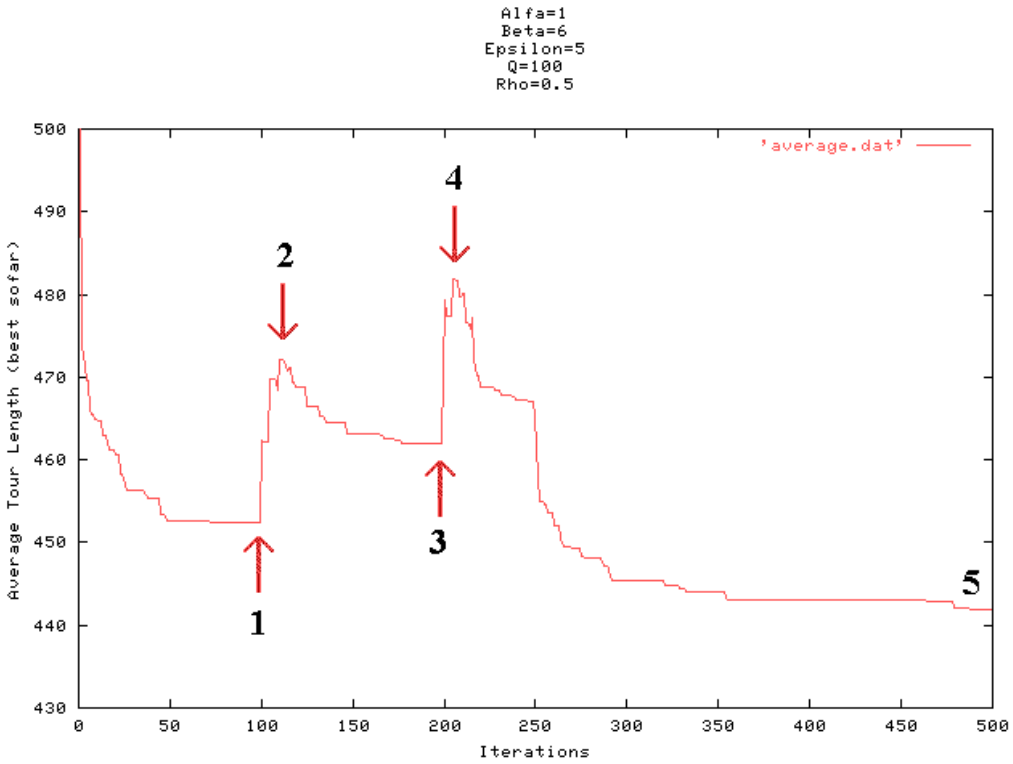


Figure 10

Table 2

	1 (100-)	2 (100+)	3 (200-)	4 (200+)	5 (end)
Nop, elite	452.7	485.2	465.4	498.7	**440.1**
Nop, no elite	456.1	480.5	472	492.1	443.8
Reset, elite	449.6	483.6	**461.2**	**480**	441.6
Reset, no elite	457.4	485	470.9	488.9	451.6
Shake, elite	452.4	**472**	461.8	481.8	441.9
Shake, no elite	458.5	477.6	468	485.3	449
Shake 10%, elite	450.4	477.7	463	489	443.4
Shake 10%, no elite	457.1	477.4	467	498.6	451.2
Shake 25%, elite	**443.9**	479.5	465	485	**440.6**
Shake 25%, no elite	458.6	487.2	469.5	489.5	450.2

In the table, the best values for every point are bold. Looking at just these results, there is no obvious winner. Although 'reset, elite' has multiple best values, some others are very close, like 'Shake, elite'.
An interesting value is the best score for 'Nop, elite' at point 5, because it shows that the original AS-TSP[14] isn't that bad for dynamic problems. We assumed in chapter 4

[14] 'Nop, elite' is not 100% original AS-TSP, it does contain the $\tau_{ij} \geq \tau_0$ change to AS-TSP.

that AS-TSP would not work (well) on dynamic problems. During the traffic jams 'nop, elite' performs worse than all other algorithms, but in the end it reaches the best score. The most probable cause for this is the fact that the minimum τ value has some influence on the dynamic performance and that the assumptions in chapter 4 might not be entirely true: they were for the 4-city problem, but on bigger problems, AS-TSP seems sufficiently robust to handle the changes.

To rank the algorithms, we might have to scale the points in importance for dynamic problems, because not all five points we are looking at are equally important in this problem. We can also look at the gradient for the descending edges in the graphs, because that is another important aspect for dynamic problems, but not visible in Table 2. The gradient is an indication of the speed of recovery of the algorithm. A high negative gradient means that the algorithm was able to recover fast from the change in the environment. The points 3 and 5 are also an indication for this: they tell us how low the values eventually became after a traffic jam, but not the speed at which it happened.

Another simple observation is that the results without elitist ants (no elite) are generally worse than their elite counterpart. This sounds counterintuitive: elitist ants reinforce the best route found so far, which could make algorithms less flexible. But the positive effects of this reinforcement are bigger than the loss of flexibility. This behaviour was also observed by [Gun01].

The best solution, considering both the values in Table 2 and the shape of the graphs, is 'Shake Elite': the first peak (point number 2) is by far the lowest and after recovery (point 3) it is almost the best, only losing by 0.6 units from 'Reset, elite'. It has a moderate value for point 4, but it recovers from that very fast and has a nice end score.

Putting the various algorithms, only considering the elitist versions, in order from best to worst, using the given criteria, gives the following list: Shake, Shake 25%, Shake 10%, Reset, Nop. Note that the local shakes and reset are very close to each other.

This is not a very surprising result. The shaking techniques seem to work on this kind of problem as was to be expected. The Nop and Reset, which are both very trivial, are performing worse. It has to be said that the differences between the results are not very big and even the Nop is still performing quite well. We support the claim made by [Bon99] and [Gun01] that ant systems are very robust and even able to solve dynamic problems, but we have also shown that tweaking AS-TSP does improve performance, at least for the class of problems that our 25 city problem is part of[15].

To summarise the results for this problem instance, the best (Shake), the worst (Nop) and the no traffic jam results for the 25-city problem are plotted in Figure 11.

[15] Static optimisation problems with a few 'disturbances'

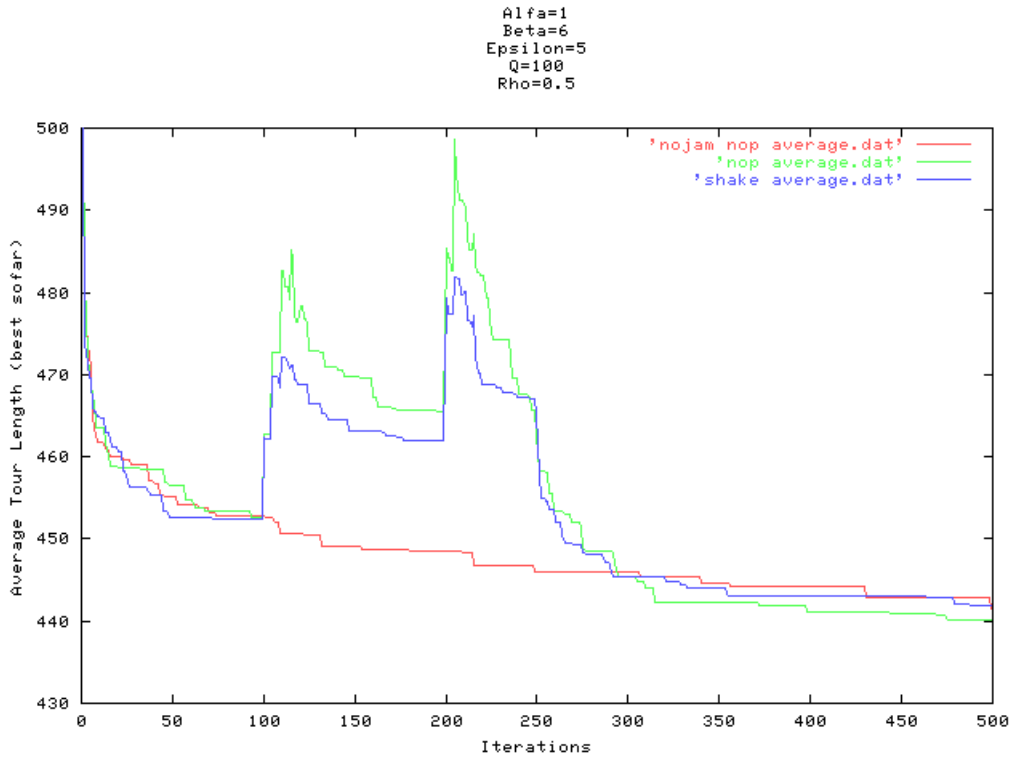


Figure 11

It is clear that the three algorithms exhibit the same performance on the first interval up to 100 iterations, they are actually the same up to that point. After that, the two versions with traffic jams each have a similar form, but Nop is just higher than Shake. Finally on the last 250 iterations they all come together again and have a very similar value at the end. Although Nop is able to out-perform all others, it is to be expected that if the three algorithms would have run for many more iterations, they would probably all have ended up at the same value.

5.5 A 100-city problem

To check how DTSP performs on a similar, but different problem, we constructed a bigger and more dynamic test case. In this problem 100 cities were placed randomly in a 100x100 grid[16]. After a short start-up period, we create a new traffic jam every 50 iterations by increasing the length of a road by a number of units in 5 equal steps. During the same 25 iterations the last traffic jam created, disappears also in 5 equal steps. The traffic jam occurs between two cities on the currently best route. The first traffic jam on the outgoing road of city number 1, the second on the outgoing road of city number 2, and so on. And likewise, the traffic jams disappear while the next one appears. This way we have problems that are very similar each run. The exact locations of the cities in the grid, is given in Appendix VI: The 100-city problem.

Every experiment runs for 5000 iterations after the start-up period, this way exactly 100 traffic jams are created.

This process of appearing and disappearing traffic jams is shown in Figure 12.

[16] This was done only once, after that we used that map as the 100-city problem.

In theory, if during the start-up period a near-optimal tour is discovered (shown as x in the figure), the algorithm will keep an upper bound to the current best route at x+5y. There are always 5y units of traffic jam on the best route. Of course the algorithm tries to compensate for it and might find tours (much) better than x+5y.

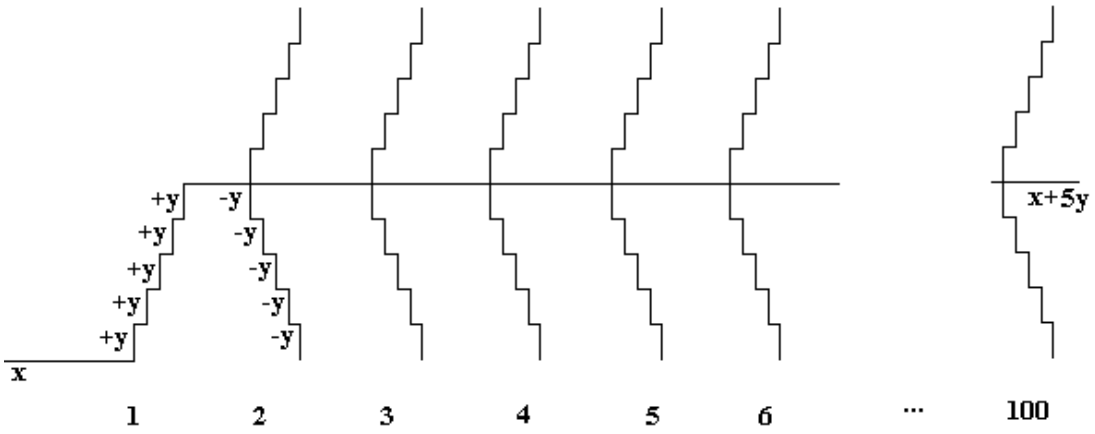


Figure 12: traffic jams in 100-city problem

One would expect our dynamic algorithms to perform at least better than x+5y, x being the average optimal tour length for the problem without traffic jams and y the value that the traffic jam increases by in every step. There is a trivial adaptation to AS-TSP that performs at that level: consider running AS-TSP on a non-dynamic version of the 100 city problem (e.g. by making a copy of the initial problem). This way one will find a near optimal solution for this (static) problem instance of length x. Using this tour as a solution for the dynamic case, will always result in a length of x+5y.

The dynamic algorithms must try to find solutions that are, on average, better than x+5y. From an external point of view, they do so by removing the road with the traffic jam from the solution, and finding another way around it that costs less than 5y units.

The average best solution for the 100 city problem is 781.6. This value stabilises after around 600 iterations. This is also why the first 600 iterations of every experiment are dropped from the statistics[17] to avoid start-up effects affecting the results.

We looked at two versions of the 100 city problem:
- Traffic jams of 25 units (y=5)
- Traffic jams of 50 units (y=10)

[17] The statistics including the first 600 values of every experiment are available too [www3]. In the y=5 set of experiments the start-up period was 50 iterations. In the y=10 set, that period was 600 iterations. A preliminary set of experiments showed that the start-up effects are minimal anyway.

There are a few minor other differences between those two problem instances. After the runs with 25 units were done, we decided to increase the start-up period from 50 to 600 iterations. And because we observed the same behaviour for this problem as we did with the 25 city problem, we decided not to do experiments with 'no elitists ants' for the 50 units problem. Experiments with elitist ants perform better than their 'no elitist' counterparts.

5.6 Results on the 100-city problems

5.6.1 The 50 units of traffic jam version

In this set of experiments the following settings were used:

- Start-up period of 600 iterations.
- Total of 5600 iterations per run[18]
- 10 runs for every type of experiment

All graphs for the experiments are in Appendix VII: All graphs for the 100-city problem
Using SPSS[19] to analyse the data we get the results in Table 3.

The average value for the 100 city problem without traffic jams is 781.6, so we would like to see the mean value for the experiments below 831.6 for them to perform better than the trivial algorithm explained before.

Table 3: Descriptive Statistics

	N	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
NOP	49990	**94.68**	778.54	**873.23**	813.8964	14.8659	220.996
RESET	49990	95.93	778.54	874.47	815.4591	14.2296	202.483
SHAKE	49990	98.74	779.28	878.03	815.6402	**14.1116**	**199.136**
SHAKE10	49990	104.75	**773.11**	877.87	**811.7543**	14.4548	208.941
SHAKE25	49990	100.34	776.05	876.40	812.6830	14.7713	218.191
Valid N	49990						

N is the total number of data points that were used for the statistics; the range is the difference between maximum and minimum.

The mean value is the most important in this type of problem. The algorithms must compensate for a constantly changing environment, and although it is nice not to have high peak values, the average value is what matters[20].

Looking at this table, we can rank the five algorithms in this order (best first): Shake10, Shake25, Nop, Shake, Reset.

[18] A careful reader will notice that the total number of iterations is actually 5599, this was a insignificant bug in the prototype. That is also why the 'N' is 49990 and not 50000.
[19] SPSS can be found at http://www.spss.com/
[20] Although one can imagine certain situations in which the peak values are at least as important.

An interesting thing is the fact that Nop is in 3rd place. A likely explanation of this fact is that in bigger problems it is very 'expensive' to throw away information. In a reset all information stored in the pheromone matrix is deleted. In a global shake this is not exactly the case, but still a lot of information is altered. The local shakes only alter the pheromone matrix near the traffic jam, which is the place where things have to change anyway. The Nop, which keeps all information, is doing better than the algorithms that do lose information. Another advantage for Nop in this type of problem is the fact that every traffic jam also disappears: if the algorithm is not able to recover from a certain traffic jam, it will only have a disadvantage for 75 iterations. After 75 iterations a traffic jam is completely gone again.

In Figure 13 the results for the best algorithm on this problem (Shake10) are combined with the 'average tour length without traffic jams + 50' (the x+5y from Figure 12) and the actual average value: 811.7.

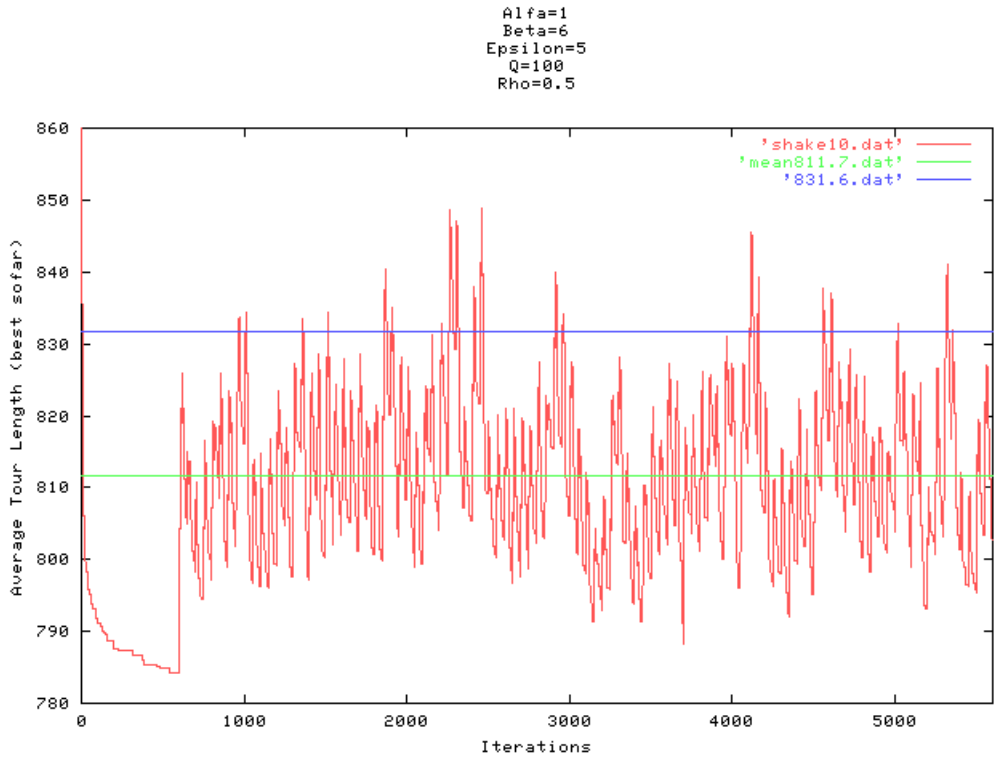


Figure 13

It is clear that the average value is much better than the expected upper bound, but we also see that there are a few peaks higher than this upper bound. This is not very strange: in a dynamic problem there are peaks, and sometimes they are just high for a variety of reasons, but the algorithm is able to get them down again every time. Using another algorithm would ensure values that are below a certain value, but worse on average.

5.6.2 The 25 units of traffic jam version

In this set of experiments the following settings were used:

- Start-up period of 50 iterations.

- Total of 5000 iterations per run[21]
- 9 runs for every type of experiment

All graphs for the experiments are in Appendix VII: All graphs for the 100-city problem
Using SPSS to analyse the data we get the results in Table 4.
The average value for the 100 city problem without traffic jams is 781.6, so we would like to see the mean value for the experiments below 806.6 for them to perform better than the trivial algorithm explained before.

Table 4: Descriptive Statistics

	N[22]	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
NOP	39591	80.57	**773.78**	854.35	809.5302	11.7317	137.634
RESET	39591	79.48	779.57	859.06	811.9289	11.3013	127.719
SHAKE	39591	76.90	774.89	851.79	811.2941	11.3859	129.639
SHAKE10	39591	**69.88**	778.98	**848.86**	808.4210	11.5541	133.497
SHAKE25	39591	78.48	775.37	853.84	**808.3932**	**11.1364**	**124.019**
Valid N	39591						

The first thing that we see is that the mean values are higher than 806.6. None of the algorithms is able to compensate for 25 units of traffic jam, although shake10 and shake25 come very close. This probably means that this problem is not suitable for the ant systems that we used. In a way it is almost a static problem: only very small changes to the environment are made, and ignoring them would yield better results than trying to keep up with them. Ignoring the fact that a better algorithm is possible, we can still look at the results.

The order in which they perform, best first, is: Shake25, Shake10, Nop, Shake, Reset. Which is the same as the 50 units traffic jam experiments, given that the Shake10 and Shake25 are very close in this experiment.

The same arguments can account for these results: resetting the pheromone matrix throws away too much information and only adjusting the values in the vicinity of the traffic jam gives the best results. And again we see that all versions of the algorithm perform well.

Figure 14 shows again that our best algorithm, shake25, is performing slightly worse than the trivial algorithm with the guaranteed upper bound of 806.6.

[21] Again: 4999 iterations
[22] (4999 – 600) * 9 = 39591

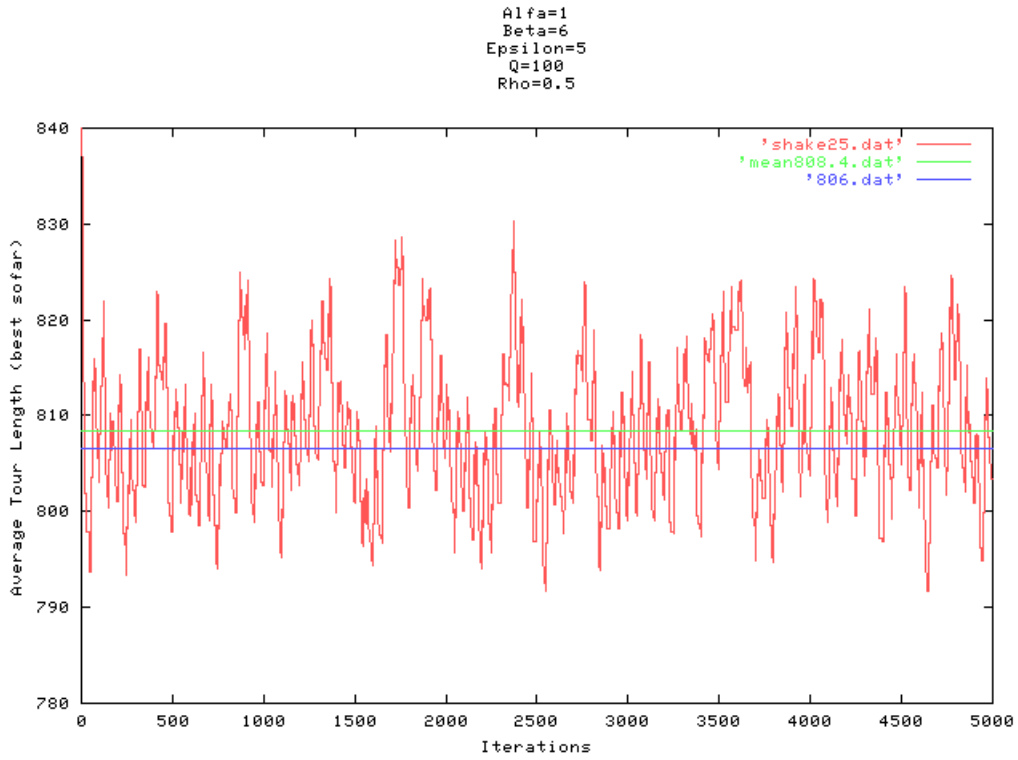


Figure 14

5.6.3 Comparing the two versions of the 100-city problem

There are many similarities between the two versions of the 100 city problem that we looked at. The only real difference is the total length of the traffic jams in the problem at any time[23], 50 units vs. 25 units.
The worst and best scores for both problems are in Table 5 and Table 6.

Table 5: Descriptive Statistics: 100 cities, 25 units of traffic jam

	N	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
RESET	39591	79.48	779.57	859.06	811.9289	11.3013	127.719
SHAKE25	39591	78.48	775.37	853.84	**808.3932**	11.1364	124.019

Table 6: Descriptive Statistics: 100 cities, 50 units of traffic jam

	N	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
RESET	49990	95.93	778.54	874.47	815.4591	14.2296	202.483
SHAKE10	49990	104.75	773.11	877.87	**811.7543**	14.4548	208.941
Valid N (listwise)	39591						

[23] Not considering the start-up period in which there are no traffic jams.

An obvious observation is the ranking of the different algorithms[24] for the two versions, which indicates that shaking does have a positive effect on the performance on (highly) dynamic problems and resetting gives the worst results.

The version with fewer traffic jams has slightly better results[25] on the 100-city problem when just looking at the values, but there is a logical explanation for that. The moment that an algorithm compensates for a traffic jam, the length has just increased by another y units. The real amount of traffic jam at which another route becomes better than the old one, is probably somewhere between the two values. This means that incremental steps of 5 units will result in an average error of 2.5 units and steps of 10 units will lose 5 units on average.
Of course this is not the only effect, but the difference between these two errors can explain most of the difference between the results. Another factor is time: usually it takes a few iterations before a new and better route is found. When traffic jams are twice as long, time is more expensive. E.g. when it takes n steps before the algorithm compensates for the traffic jam, the route is n·10 units longer instead of n·5. This effect might very well be responsible for the remaining difference.

[24] Ranking: local shake, nop, global shake, reset.
[25] Difference between the best score for the 2 versions of the 100-city problem: 811.7 – 808.4 = 3.3

6 Conclusions

We have looked at various versions of Ant System applied to three different instances of the dynamic travelling salesman problem. The most striking effect that we have observed, is that every algorithm we tried performed reasonably well on every problem. This supports the claim we were investigating: ant systems are able to perform well on dynamic problems.
But when looking closer at the results, we also were able to improve Ant System for Dynamic TSP by adding the Shake routine. Especially the algorithms in which 'local shake' was implemented performed very well on dynamic travelling salesman problems. The more dynamic[26] a problem gets, the more important it is to preserve information that was collected earlier in the pheromone matrix. This is why resetting the pheromone matrix gives the worst results for the 100-city problems: all information is lost and new changes occur before the algorithm has completely recovered.
The local shake algorithms have two strengths. On the one hand they keep as much information as possible by only altering the pheromone matrix near changes in the environment. On the other hand they preserve some information in those changed areas by not removing the information from the matrix, but merely normalising the amount of pheromones to stimulate exploration.

6.1 Recommendations and future research

A common problem when looking at Ant Systems is the enormous amount of parameters in the algorithm. We have not looked at the influence of them regarding dynamic problems. Only the parameter that we introduced ourselves, the shake-percentage p, was changed a few times (10%, 25% and 100%), but even this parameter requires more research in our opinion.

A next challenge would also be trying to classify dynamic problems and choose the right algorithm and set of parameters for that problem. We have already seen that the degree of dynamics in a problem makes a difference in what type of algorithm gives the best results. In some situations even using a trivial solution can be better than using a version of Ant System that is tweaked for dynamic problems.

A real challenge would be to build a 'wizard' that constructs a suitable ant algorithm for a certain task by asking some questions about the nature of the problem. A guideline for such a wizard is given by [Bon99] in chapter 2.5. Extending this recipe to include dynamics could be a very interesting research.

Looking at the performance of AS-DTSP on static problems is another thing we did not yet do. It might be possible that shaking has a positive influence on static problems too: it could be another way to prevent stagnation of the results in a local optimum.

Finally it would be nice if someone would prove that our theories still hold on much bigger problems, although there is no reason to doubt that based on our current results.

[26] In this context, 'more dynamic' means more changes during the same time-interval.

7 References

7.1 Bibliography

[Bon99] E. Bonabeau, M. Dorigo and G. Theraulaz*, Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, 1999

[Bul99] B. Bullnheimer, *Ant Colony Optimization in Vehicle Routing*, Doctoral thesis, University of Vienna, January 1999.

[Cos97] Costa D. and A. Hertz, *Ants Can Colour Graphs*, Journal of the Operational Research Society, 48, 295-305, 1997

[Dor96] M. Dorigo, V. Maniezzo, A. Colorni, *The Ant System: Optimization by a Colony of Cooperating Agents*. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41, 1996

[Dor99] M. Dorigo, G. Di Caro, *The Ant Colony Optimization Meta-Heuristic,* New Ideas in Optimization, McGraw-Hill, 11-32, 1999

[Gun01] M. Guntsch, J. Branke, M. Middendorf and H. Schmeck, *ACO Strategies for Dynamic Optimisation Problems*, In *[To be published]*, 2001

[Gun01-2] M. Guntsch, M. Middendorf and H. Schmeck, An Ant Colony Optimization Approach to Dynamic TSP, Proceedings of GECCO 2001, 2001

[Gam97] L. Gambardella, M. Dorigo, *HAS-SOP: An Hybrid Ant System for the Sequential Ordering Problem*, Tech. Rep. No. IDSIA 97-11, IDSIA, Lugano, 1997

[Kau95] S. Kaufman, *At Home In The Universe: The Search For Laws Of Self-Organization And Complexity*, Oxford University Press, 1995

[Man94] V. Maniezzo, A. Colorni and M. Dorigo (1994). The Ant System Applied to the Quadratic Assignment Problem. Tech. Rep. IRIDIA/94-28, Université Libre de Bruxelles, Belgium, 1994

[Par01] R. Parpinelli, H. Lopes, A. Freitas, *An Ant Colony Based System for Data Mining: Applications to Medical Data,* Proceedings of GECCO 2001, 2001

[Rus95] S. Russell, P. Norvig, *Artificial Intelligence, A Modern Approach,* Prentice-Hall International Inc., New Jersey, 1995

[Stu99] T. Stützle, H. Hoos, *MAX-MIN Ant System*, 1999

7.2 Web references

[www1] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/
A collection of links and articles about TSP and a library of benchmarks

[www2] http://www.keck.caam.rice.edu/tsp/
Some basic information about (the history of) TSP

[www3] http://www.eyckelhof.nl/
Personal homepage of the author, contains all data and sources on DTSP

[www4] http://www.gnuplot.org/
Homepage of gnuplot, used for most graphs in this report

[www5] http://money.telegraph.co.uk/money/main.jhtml?xml=/money/2001/06/06/cnantz06.xml
Article about real world applications of Ant System

8 Appendices

8.1 Appendix I: Some results of classic AS-TSP

In this appendix the results obtained with the prototype for AS-TSP, implemented in Borland Delphi, are presented.
Both results using the original algorithm and a few variations on it are given.

Original algorithm (applied to Eil-51[27], St-70 and KroA100 benchmarks):

```
Eil-51
-------
Best tour: 423 - 41
41 13 25 14 18 4 47 12 46 51 27 48 6 23 24 43 7 26 8 31 28 3 20 35 36
22 1 32 11 38 5 49 9 50 16 2 29 21 34 30 10 39 33 45 15 44 37 17 42
19 40

Loop 239
Best tour: 439 - 18
18 47 12 46 51 27 48 6 23 24 43 7 26 8 31 28 3 20 35 36 22 1 32 11 38
5 49 9 50 16 2 29 21 34 30 10 39 33 45 15 44 37 17 4 42 40 19 41 13
25 14

Loop 723
Best tour: 444 - 51
51 46 12 47 4 18 14 24 43 7 23 48 8 26 31 28 3 20 35 36 22 1 32 11 38
5 49 9 50 16 2 29 21 34 30 10 39 33 45 15 44 37 17 42 40 19 41 13 25
6 27

Loop 399
Best tour: 438 - 41
41 13 25 14 18 4 47 12 46 51 27 48 6 24 43 23 7 26 8 31 28 3 20 35 36
22 1 32 11 38 5 49 9 50 16 2 29 21 34 30 10 39 33 45 15 44 37 17 42
19 40

St-70
-----
Best tour: 685 - 65
65 56 51 50 58 37 47 16 23 36 29 13 31 69 70 35 59 38 22 66 63 57 15
24 19 26 49 55 28 14 20 30 44 68 27 46 40 61 3 39 8 45 25 9 7 2 4 62
6 54 33 12 34 21 17 43 41 42 18 32 53 10 52 60 5 67 48 11 1 64

Loop 133
Best tour: 692 - 31
31 69 38 59 63 22 66 57 15 24 19 26 49 55 28 14 20 30 44 68 27 46 40
61 3 39 8 45 25 9 7 2 4 62 6 54 33 12 21 34 17 43 41 42 18 32 53 10
52 60 51 56 67 5 48 11 65 64 1 50 58 37 47 16 23 36 29 13 70 35

KroA100
-------
Loop 75
Best tour: 23145 - 87
87 51 61 25 81 69 73 50 44 2 64 40 54 67 58 28 93 1 92 8 42 89 31 80
56 97 75 19 53 88 16 22 94 70 66 65 4 26 79 18 24 38 36 99 59 74 21
72 10 84 90 49 6 63 47 32 11 15 17 45 91 98 23 60 77 62 35 86 27 12
20 57 7 9 83 55 34 29 46 3 43 14 71 41 100 48 52 78 96 5 37 33 76 13
95 82 39 30 85 68
```

Without the elitist ants:

[27] If not mentioned otherwise, results are for the Eil-51 benchmark.

```
Eil51:
Loop 39
Best tour: 455 - 19
19 41 13 25 14 24 43 7 23 48 8 26 31 28 3 20 35 36 29 21 34 30 9 50
16 2 22 1 32 11 38 5 49 10 39 33 45 15 44 37 17 4 18 47 12 46 51 27 6
42 40

Loop 319
Best tour: 449 - 23
23 24 43 7 26 8 31 28 3 20 35 36 29 21 34 50 16 9 49 10 30 39 33 45
15 37 17 4 18 47 12 46 51 27 32 1 22 2 11 38 5 44 42 40 19 41 13 25
14 6 48

Loop 44
Best tour: 444 - 18
18 47 12 46 51 27 48 6 23 24 43 7 26 8 31 28 3 20 35 36 22 1 32 11 38
5 49 9 50 16 2 29 21 34 30 10 39 33 45 15 44 37 17 4 41 19 42 40 13
25 14
```

Without elitist ant; factor 2 in: $\Delta\tau_{ij}^{k}(t) = \begin{cases} Q/(2\cdot L^{k}(t)), if\,(i,j)\in T^{k}(t); \\ 0, otherwise, \end{cases}$

```
Loop 48
Best tour: 447 - 42
42 40 19 41 13 25 14 24 43 7 23 48 8 26 31 28 3 20 35 36 22 1 32 11
38 5 49 9 50 16 2 29 21 34 30 10 39 33 45 15 44 37 17 4 47 12 46 51
27 6 18

Loop 107
Best tour: 452 - 33
33 45 15 44 37 17 4 18 14 25 13 41 19 42 40 47 12 46 51 27 6 48 23 24
43 7 26 8 31 28 3 20 35 36 22 1 32 11 38 5 49 9 50 16 2 29 21 34 30
10 39

Loop 54
Best tour: 452 - 42
42 19 41 13 25 14 24 43 7 23 48 8 26 31 28 3 20 35 36 22 1 32 11 38 5
49 9 50 16 2 29 21 34 30 10 39 33 45 15 44 37 17 4 47 12 46 51 27 6
18 40

Loop 88:
Best tour: 450 - 18
18 47 12 46 51 27 32 11 38 5 49 9 50 16 2 29 21 34 30 10 39 33 45 15
44 37 17 4 42 40 19 41 13 25 14 24 43 7 23 48 8 26 31 28 3 20 35 36
22 1 6
```

$\alpha = 0$; Only visibility has influence on the probability of an ant choosing a certain edge.

```
Loop 17
Best tour: 464 - 50
50 16 2 22 1 27 32 11 38 9 49 5 12 47 4 17 37 44 15 45 33 10 39 30 34
21 29 20 35 36 3 28 31 26 8 48 23 7 43 24 14 25 13 41 19 40 42 18 6
51 46

Loop 24
Best tour: 479 - 47
47 12 46 51 14 25 13 19 41 40 42 44 15 45 33 39 10 49 38 16 50 9 30
34 21 29 20 35 36 3 28 31 26 8 48 27 32 1 22 2 11 5 37 17 4 18 24 23
7 43 6

Loop 156
```

```
Best tour: 473 - 47
47 12 5 38 9 50 34 30 10 49 39 33 45 15 44 37 17 4 18 6 27 46 51 11
32 1 22 2 29 16 21 20 35 36 3 28 31 26 8 48 23 7 43 24 14 25 13 41 19
42 40
```

8.2 Appendix II: The 25-city problem

The co-ordinates used in our 25-city problem. Every line contains the x and y co-ordinate of a city in a Cartesian system (100x100 grid)

City 1-5	City 6-10	City 11-15	City 16-20	City 17-25
5 5	46 20	90 70	49 75	33 45
13 10	91 13	85 72	40 54	85 67
9 12	81 35	91 90	25 57	90 61
10 39	89 45	65 91	18 77	22 30
35 27	66 60	51 80	29 92	69 25

The best solution we found for this (static) problem is:
Length: 434.397
Route: 23, 11, 22, 12, 13, 14, 15, 16, 20, 19, 18, 17, 21, 4, 3, 1, 2, 24, 5, 6, 25, 7, 8, 9, 10

The route is given in indices, in which city 1 is (5,5) ; city 2 (13,10) etc.

8.3 Appendix III: Data files created by DTSP

DTSP generates two data files for every experiment. The real data file, {timestamp}result.dat, contains a small header containing the parameters used in the experiment, a list of loop-value pairs and one last line containing the best found solution, and some more info about it.

An example ('986995218020result.dat'):

```
# Alfa =1
# Beta =6
# Rho =0.5
# Q =100
# Epsilon =0
1 506.06653135607655
2 500.6111539663709
3 471.0271091662125
4 467.9041465231706
5 457.11623748027444
6 457.11623748027444
[cut]
497 453.2232785116911
498 453.2232785116911
499 453.2232785116911
# Kortste tour van mier 5: 453.2232785116911. Route: 5, 24, 2, 1, 3,
4, 21, 17, 18, 19, 20, 16, 15, 14, 13, 12, 22, 11, 23, 9, 8, 25, 7,
10, 6,
```

The other file, {timestamp}plot.plt, is a script file that is understood by gnuplot. The format is almost self-explanatory. The file shown below ('986995218020plot.plt') is the plot file corresponding to the given data file.

```
set terminal png small color
set output "986995218020.png"
set data style lines
set xlabel 'Iterations'
set ylabel 'Tour Length (best sofar)'
set xrange [0:500]
set yrange [430:500]
set title "Alfa =1\nBeta =6\nRho =0.5\nQ =100\nEpsilon =0"
plot '986995218020result.dat'
```

Running this script through gnuplot [www4], will result in a graphic representation of the experiment (in .png format). The plot for this script ('986995218020.png') is added as Figure 15.

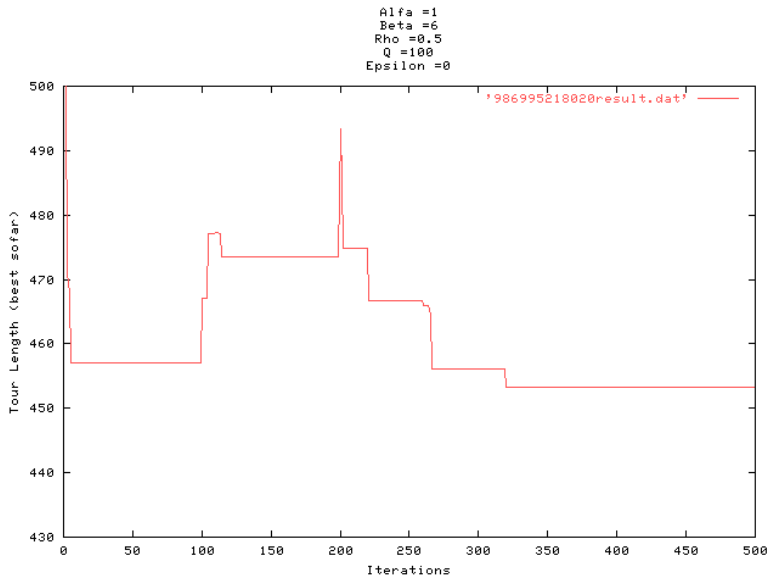

Alfa =1
Beta =6
Rho =0.5
Q =100
Epsilon =0
'986995218020result.dat'
Tour Length (best sofar)
Iterations
500
490
480
470
460
450
440
430
0
50
100
150
200
250
300
350
400
450
500

Figure 15

8.4 Appendix IV: perl script

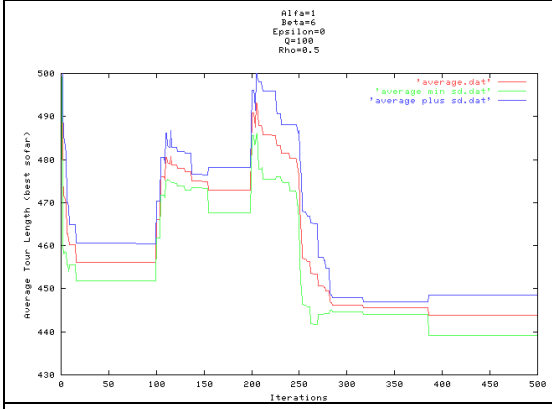
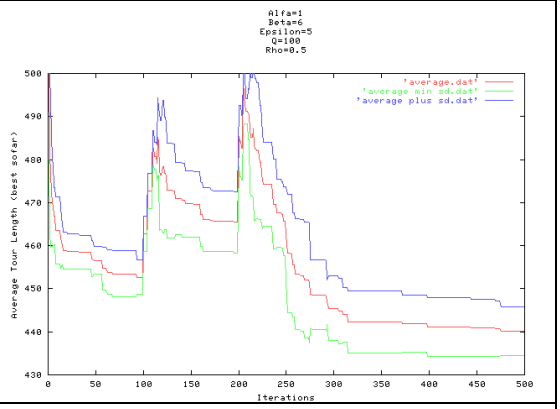
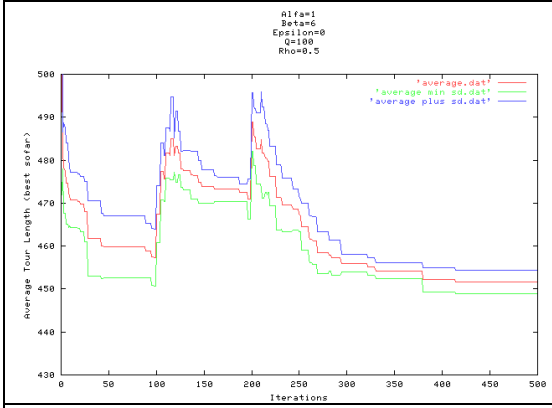
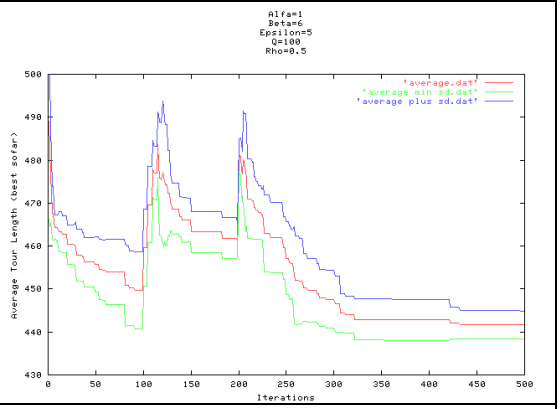
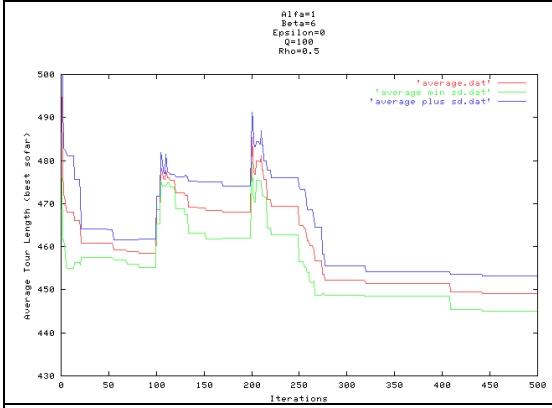
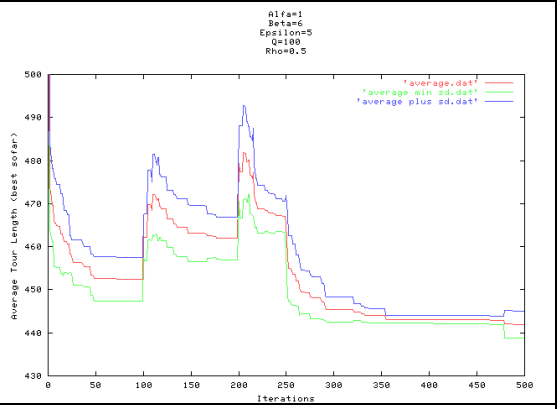
The perl script used to combine multiple results into one graph, source also available via [www3]:

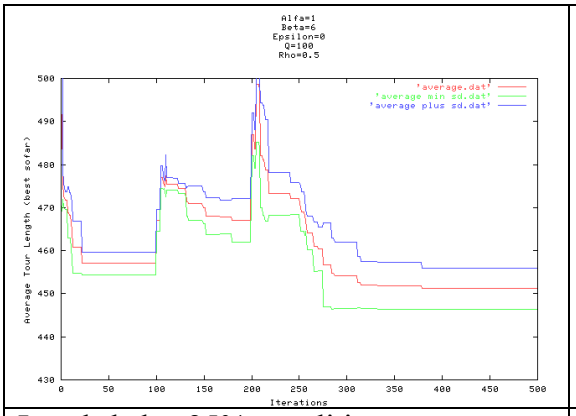
```
#!D:\perl\bin\perl.exe
# On most Osses this line would read "#!/usr/bin/perl"

# Creates an average gnuplot graph from multiple datasets

use Cwd;
open (OUTFILE_A, ">average.dat");
open (OUTFILE_B, ">average_min_sd.dat");
open (OUTFILE_C, ">average_plus_sd.dat");
$dirname = cwd();
my @data = ();
my $count =0;
my %params = ();
my @all;

opendir(DIR, $dirname) or die "can't opendir $dirname: $!";
while (defined($file = readdir(DIR))) {
       if ($file =~ /.*?result\.dat$/){
      print OUTFILE_A "# $dirname/$file\n";
      print OUTFILE_B "# $dirname/$file\n";
      print OUTFILE_C "# $dirname/$file\n";
      open (DATAFILE, "$dirname/$file");
      $count++;
      # Parse one of the result.dat files
      while ($line = <DATAFILE>){
          my @dataset =();
          if ($line !~ /^#.*$/){
                my ($number, $value) = split / /, $line;
                $data[$number] += $value;
                $all[$count][$number] = $value + 0;
          }
          if ($line =~ /# (.*?) =(.*?)$/){
                 $params{$1}=$2;
          }
      }
      close (DATAFILE);
      #$all[$count] =   $dataset  ;
    }
}


for ($i=1; $i < @data; $i++){
       #bereken standaard-deviatie rond $i
       my $gem = $data[$i]/$count;
       my $sd = 0;
       for ($datarij=1; $datarij <= $count; $datarij++) {
         $datapunt = $all[$datarij][$i];
         #print "$datapunt ";
         #print "i = $i ; datapunt = $datapunt ; gem = $gem \n";
         $sd += (($datapunt - $gem) * ($datapunt - $gem)) / ($count -1);
       }
       $sd = sqrt($sd);
    print OUTFILE_A "$i $gem \n";
    print OUTFILE_B "$i " . ($gem - $sd) . "\n";
    print OUTFILE_C "$i " . ($gem + $sd) . "\n";
    #print "\n $i " . $sd . " " . $gem . " " . ($gem - $sd) . " " . ($gem + $sd) .
"\n";
}

closedir(DIR);
close (OUTFILE_A);
close (OUTFILE_B);
close (OUTFILE_C);

############################## Alleen gemiddelde ######################
open (OUTFILE, ">average.plt");
print OUTFILE "set terminal png small color\n";
print OUTFILE "set output \"average.png\"\n";
print OUTFILE "set data style lines\n";
print OUTFILE "set xlabel 'Iterations'\n";
print OUTFILE "set ylabel 'Average Tour Length (best sofar)'\n";
print OUTFILE "set xrange [0:500]\n";
print OUTFILE "set yrange [430:500]\n";
print OUTFILE "set title \"";
```

```
foreach $param (sort(keys %params)){
   print OUTFILE "$param=$params{$param}\\n"
}
print OUTFILE "\"\n";
print OUTFILE "plot 'average.dat' ";

############################## Gemiddelde en SD ############################
open (OUTFILE, ">average_with_SD.plt");
print OUTFILE "set terminal png small color\n";
print OUTFILE "set output \"average_SD.png\"\n";
print OUTFILE "set data style lines\n";
print OUTFILE "set xlabel 'Iterations'\n";
print OUTFILE "set ylabel 'Average Tour Length (best sofar)'\n";
print OUTFILE "set xrange [0:500]\n";
print OUTFILE "set yrange [430:500]\n";
print OUTFILE "set title \"";
foreach $param (sort(keys %params)){
   print OUTFILE "$param=$params{$param}\\n"
}
print OUTFILE "\"\n";
print OUTFILE "plot 'average.dat' , 'average_min_sd.dat' , 'average_plus_sd.dat' ";
```

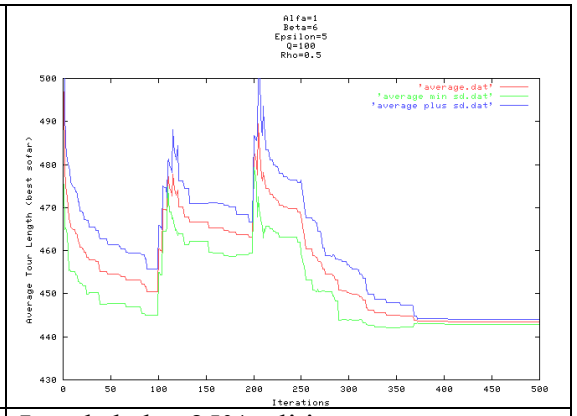
8.5 Appendix V: Full results for 25 cities

The average graph for every set of experiments. The 3 lines in every graph are average, average - standard deviation and average + standard deviation.

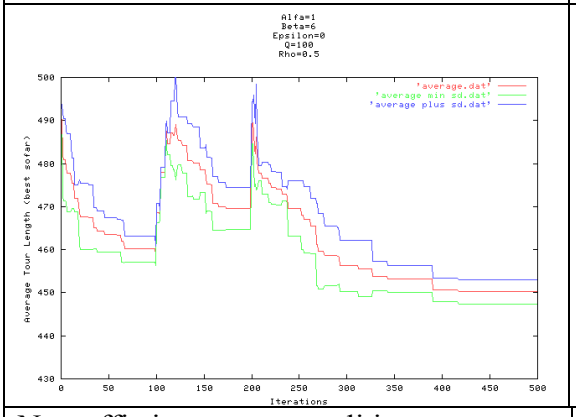
Nop, no elitist ant	Nop, elitist ant
	
Reset, no elitist ant	Reset, elitist ant
	
Global shake, no elitist ant	Global shake, elitist ant
	
Local shake, 10%, no elitist ant	Local shake, 10%, elitist ant



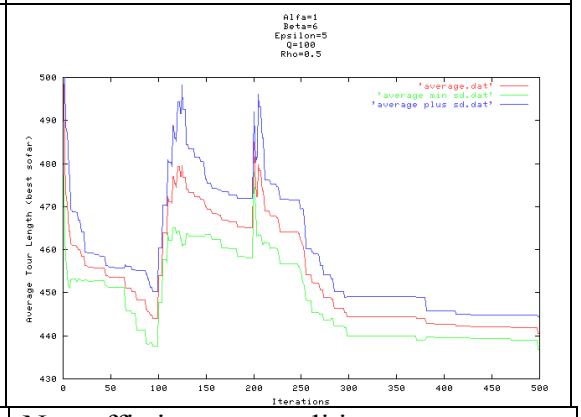
Local shake, 25%, no elitist ant



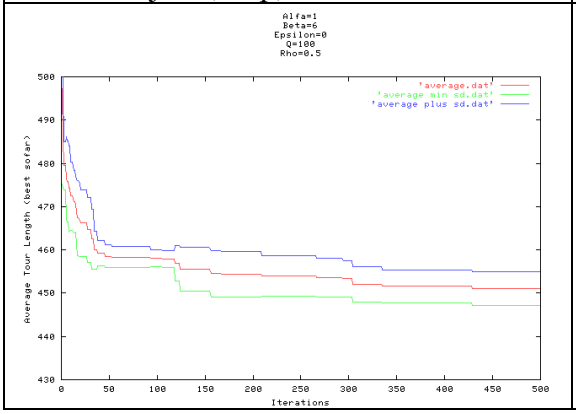
Local shake, 25%, elitist ant

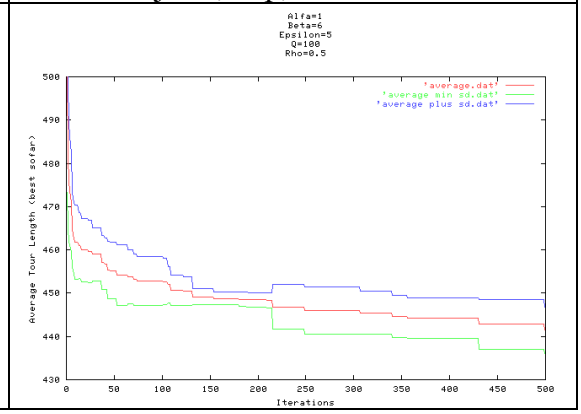


No traffic jams, nop, no elitist ant



No traffic jams, nop, elitist ant





8.6 Appendix VI: The 100-city problem

The co-ordinates used in our 25-city problem. Every line contains the x and y co-ordinate of a city in a Cartesian system (100x100 grid)

City 1-20	City 21-40	City 41-60	City 61-80	City 81-100
83 7	65 88	25 47	65 46	21 75
24 45	21 57	67 12	54 14	84 68
6 27	59 61	52 79	11 52	75 28
87 69	5 35	53 52	38 72	88 86
64 17	43 99	30 58	50 47	72 35
97 97	97 35	41 45	45 99	52 8
64 66	73 34	63 16	89 88	48 95
67 42	80 20	38 94	31 36	82 37
75 88	88 88	3 16	84 80	94 11
72 68	52 72	56 21	6 67	20 78
80 90	31 4	76 82	29 9	76 58
86 59	66 26	27 62	90 3	23 88
73 55	72 8	29 62	25 71	63 28
49 83	42 47	0 60	69 99	96 29
2 50	61 66	22 56	26 90	89 97
94 74	85 69	44 21	71 97	5 23
79 21	94 6	59 26	37 16	26 47
57 87	50 51	25 79	15 56	13 43
8 22	28 36	32 23	9 99	62 1
78 23	49 12	90 69	92 35	68 28

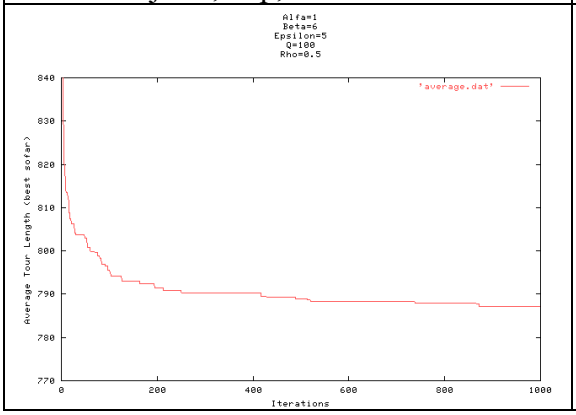
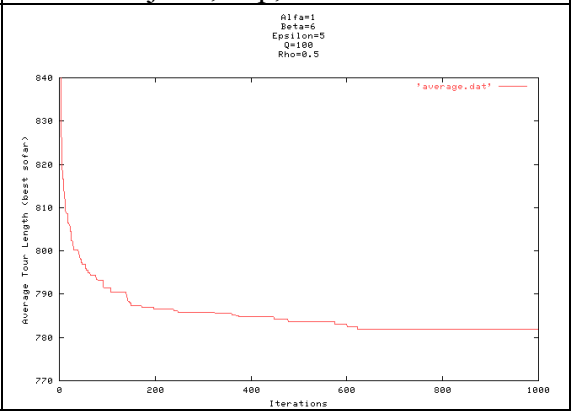
The best solution we found for this (static) problem is:
Length: 777.61
Route: 62, 40, 86, 99, 33, 42, 5, 47, 57, 93, 32, 100, 83, 20, 17, 28, 1, 72, 37, 89, 94, 26, 80, 88, 27, 85, 8, 61, 13, 91, 12, 82, 36, 4, 60, 16, 69, 84, 29, 67, 95, 6, 11, 9, 51, 10, 7, 35, 23, 44, 38, 65, 34, 46, 68, 39, 2, 41, 97, 98, 63, 78, 22, 55, 45, 53, 52, 73, 81, 90, 58, 64, 30, 43, 14, 18, 21, 76, 74, 87, 66, 25, 48, 75, 92, 79, 70, 54, 15, 24, 3, 96, 19, 49, 71, 31, 77, 59, 56, 50

The route is given in indices, in which city 1 is (83,7) ; city 2 (24,45) etc.

8.7 Appendix VII: All graphs for the 100-city problems

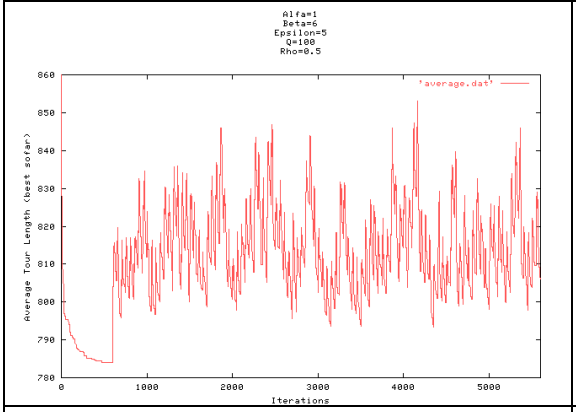
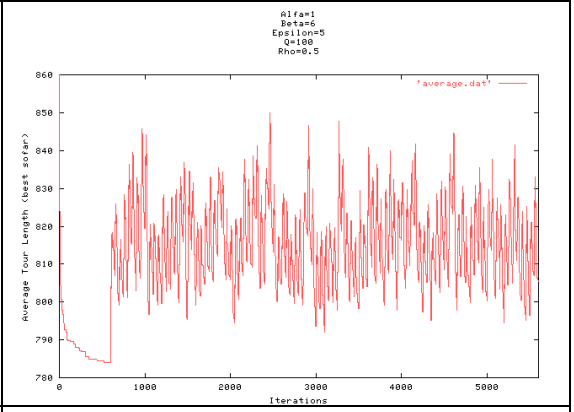
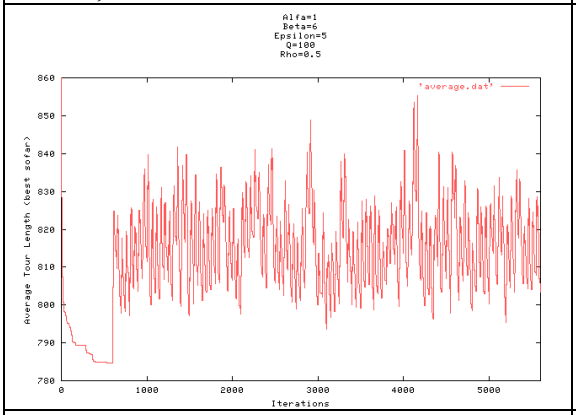
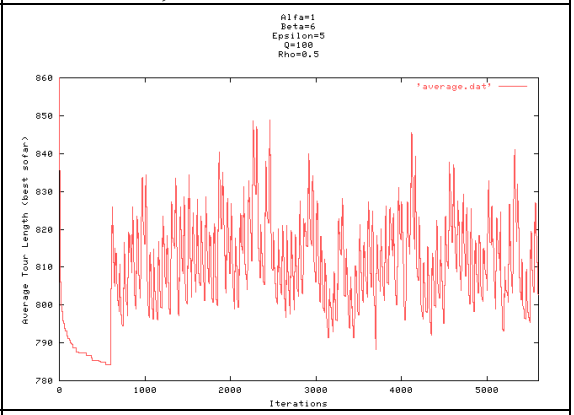
8.7.1 No traffic jams

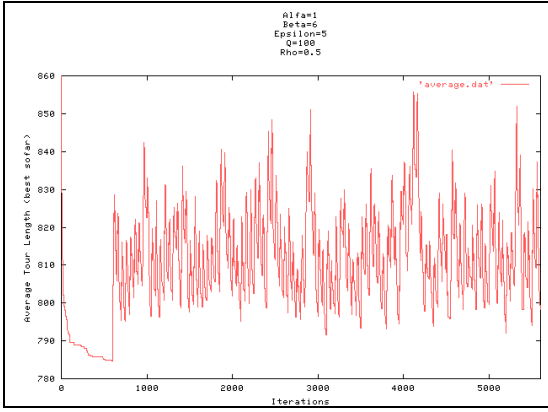
The 100 city problem without traffic jams.

No traffic jams, nop, no elitist ant	No traffic jams, nop, elitist ant
	

8.7.2 50 units of traffic jams

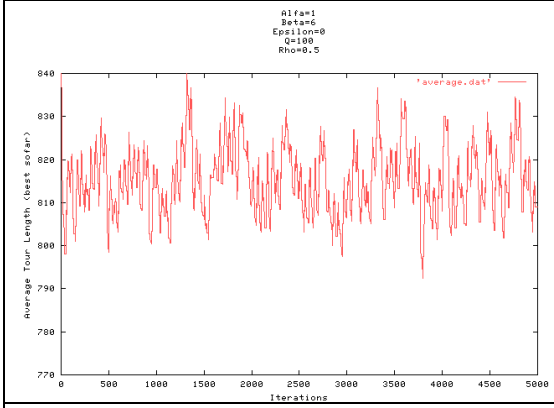
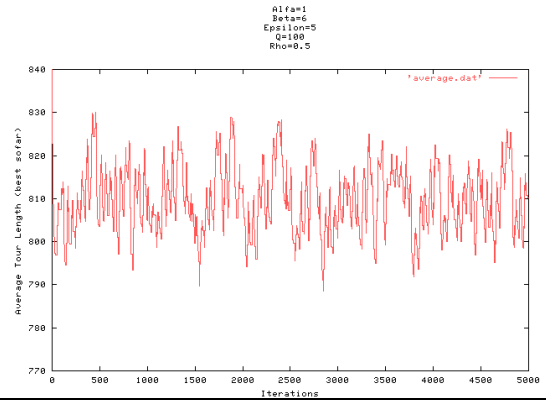
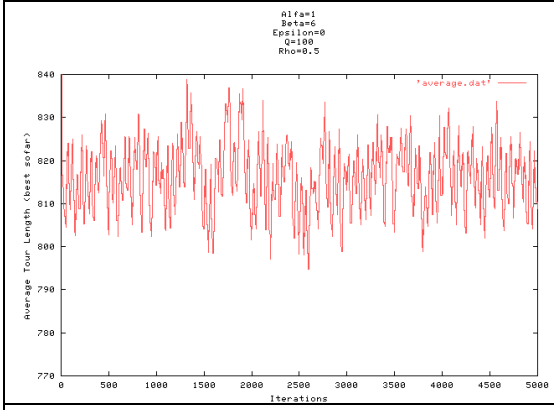
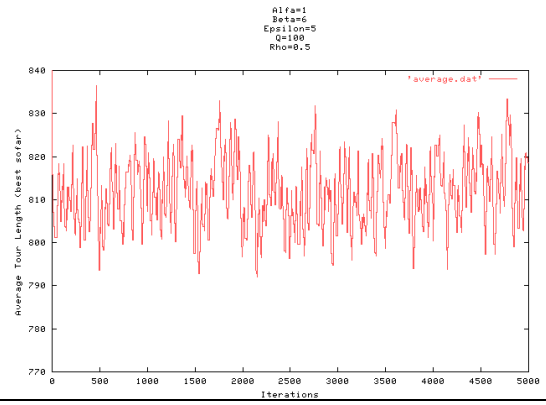
The average graph for every set of experiments. All 5600 iterations shown.

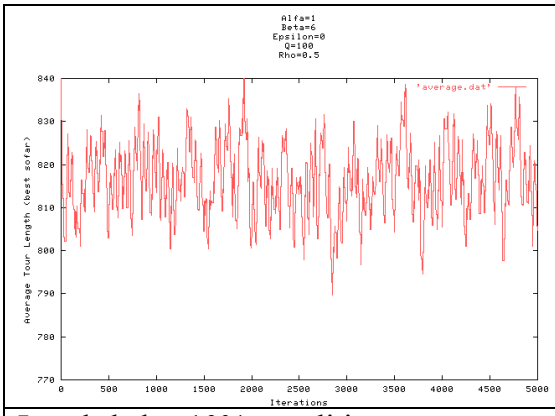
Nop, elitist	Reset, elitist
	
Shake, elitist	Shake 10%, elitist
	
Shake 25%, elitist	



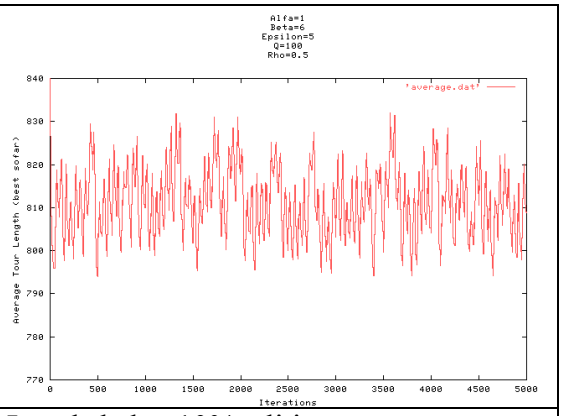
8.7.3 25 units of traffic jams

The average graph for every set of experiments. All 5000 iterations shown. Note: the scale on the y-axis is not the same as the graphs in 8.7.2

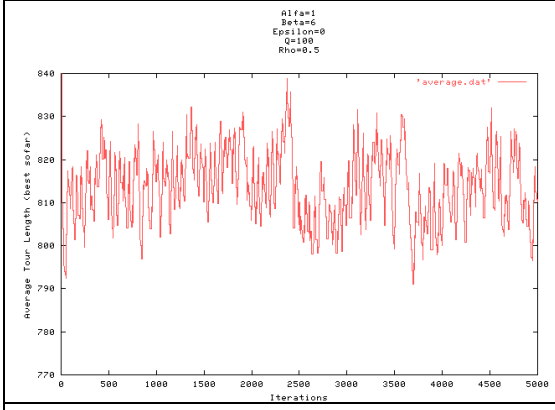
Nop, no elitist ant	Nop, elitist ant
	
Reset, no elitist ant	Reset, elitist ant
	
Global shake, no elitist ant	Global shake, elitist ant



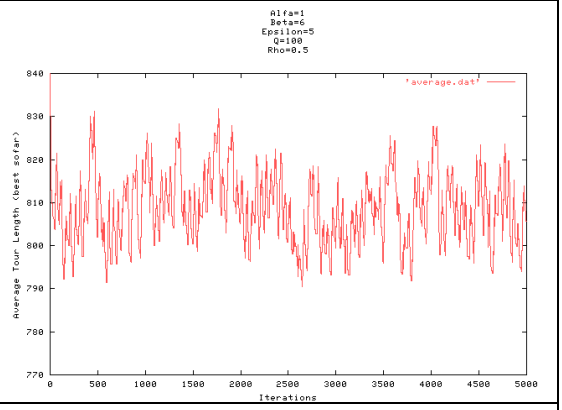
Local shake, 10%, no elitist ant



Local shake, 10%, elitist ant



Local shake, 25%, no elitist ant



Local shake, 25%, elitist ant