Ant Systems for a Dynamic TSP

Ants caught in a traffic jam

Casper Joost Eyckelhof[1] and Marko Snoek[2]

[1] Multigator vof
email: casper@eyckelhof.nl
[2] Dept. of Computer Science, University of Twente
email: snoek@cs.utwente.nl

Abstract. In this paper we present a new Ants System approach to a dynamic Travelling Salesman Problem. Here the travel times between the cities are subject to change. To handle this dynamism several ways of adapting the pheromone matrix both locally and globally are considered. We show that the strategy of smoothing pheromone values only in the area containing a change leads to improved results.

1 Introduction

In nature, many systems consisting of very simple parts demonstrate a remarkable 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*. Ants' foraging behaviour shows that travel between nests and sources of food is highly optimised. Based on this phenomenon, Ant Colony Optimisation (ACO) algorithms have been developed [SH97, BDT99, DC99], of which Ant System (AS) was the first [DMC96]. These ACO algorithms have been successfully applied to a variety of combinatorial optimisation problems [BR01].

Like other metaheuristics ACO algorithms have proven their use for static problems, but less is known about their behaviour on dynamic problems. In dynamic problems the goal is not to find desirable solutions, but to track and/or find new desirable solutions.

After the initial emphasis on static problems, some of the focus is now shifting towards dynamic variants of combinatorial optimisation problems. Recently some research is being done on ACO for dynamic problems [GBMS00, GM01, GMS01].

This paper gives an outline of research done as part of a graduation project at the University of Twente [Eyc01]. The main goals of that research were to test the hypothesis that ant systems can be applied successfully to dynamic problems and to adapt the original ant system algorithm in order to increase its performance on a particular dynamic problem.

In this paper we will show that both goals were reached: ant systems can be applied successfully to dynamic problems and enhancements to the original

algorithm are proposed for a chosen dynamic problem. The problem we study in this research is based on the Travelling Salesman Problem (TSP). TSP is a well-known combinatorial optimisation problem and the original ant system algorithm was designed for TSP.

The paper is structured as follows. In §2 the TSP as well as the ant system for this problem are explained. In §3 we introduce our adaptation to TSP in order to create a dynamic problem and we will explain how ant systems can be modified to perform better on this dynamic problem. Next the test setup for the experiments is explained in §4. In §5 the experimental results are presented. We will finish with our conclusions in §6 and ideas for future work in §7.

2 The Ant System Approach to TSP

The *Travelling Salesman Problem* is a well-known problem among computer scientists and mathematicians. The task basically consists of finding the shortest tour through a number of cities, visiting every city exactly once. Formally, the symmetric TSP is defined as:

> Given a set of n nodes and costs associated with each pair of nodes, find a closed tour of minimal total costs that contains each node exactly once, the cost associated with the node pairs $\{i, j\}$ and $\{j, i\}$ being equal[3].

It is also possible to discard that last condition and allow the distance from city i to city j to be different from the distance between city j and city i. We refer to that case as the asymmetric travelling salesman problem. Our focus will be on the symmetric TSP.

The popularity of TSP probably comes from the fact that it is a very easy problem to understand and visualise, while it is very hard to solve. Many other problems in the $\mathcal{NP}$-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. Within this domain heuristics that find acceptable solutions using an acceptable amount of resources are necessary.

In their book [BDT99], Bonabeau et al. give a good explanation of *Ant System (AS)*, the algorithm we will use as a basis for our own algorithm. It has become common practice to name the algorithms with their field of application, so this version of AS is called AS-TSP.

In AS-TSP m ants individually construct candidate solutions in an incremental fashion. The choice of the next city is based on two main components: pheromone trails and a heuristic value, called visibility in TSP. At the start all possible roads are initialised with a certain amount of pheromone: τ_0. Then

[3] Using nomenclature corresponding to the metaphor of a travelling salesman we will use city for node, distance or travel time for cost, and road for node pair.

each ant constructs a solution by choosing the next city based on the observed pheromone levels (τ) and visibility (η) until it has visited all cities exactly once. The visibility is a proximity measure, defined as the inverse of the distance between two cities i and j: $\eta_{ij} = 1/d_{ij}$, where η_{ij} is the visibility associated with choosing city j when in city i and d_{ij} is the distance between these two cities.

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

Formally, if during the t^{th} iteration the k^{th} ant is located in city i, the next city j is chosen according to the probability distribution over the set of unvisited cities J^k defined by:

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

When each ant has completed a tour, the amounts of pheromones are updated. On every road some fraction of the pheromone evaporates, while on roads, that have been visited by at least one ant, new pheromones are deposited. The amount of pheromone that is deposited is based upon the number of ants that visited the road, the length of the tours the road is part of and a parameter Q[4]. The more a road is visited and the shorter the tour, the more pheromone is deposited. The fraction of pheromone that evaporates is based on the parameter ρ.

All roads on the best tour so far get an extra amount of pheromone. This is called an *elitist ant* approach: the best ants reinforce their tour even more. The balance between positive reinforcement through depositing pheromone by an (elite) ant and pheromone evaporation, which is a negative reinforcement, is crucial for finding good solutions.

This process goes on until a certain condition, such as a certain number of iterations, amount of CPU time, or solution quality, has been achieved.

3 Ant System Approaches to Dynamic TSP

The Dynamic Travelling Salesman Problem (DTSP) that was used in this project is a variation on the TSP in the sense that the original TSP metaphor is extended to include traffic jams. If we look at the distance between cities as travel times, they no longer need to be fixed. By introducing a traffic jam on a certain road

[4] Since the value of Q only weakly influences the result, it will not be discussed here.

the associated travel time is increased[5]. We mention that several other variants of DTSP, such as the DTSP resulting from the insertion or deletion of cities, are also possible [GBMS00, GM01, GMS01].

In our DTSP, ants regularly come across traffic jams. This is guaranteed since traffic jams will be introduced only on roads that are in the best tour of the AS at that moment. Moreover, these traffic jams have a somewhat natural way of increasing for a while and then decreasing until they are resolved. More details will be given in the next paragraph. Let us now discuss ways to adapt the AS to handle this kind of dynamics.

The combination of positive and negative reinforcement, as mentioned in §2, works well for static problems. In the beginning there is relatively much exploration. 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 the associated pheromones have evaporated over time. In dynamic situations however, solutions that are bad before a change in the environment, might be good afterwards. Now, if the ant system has converged to a state where those solutions are ignored, very promising roads will not be sampled and the result will be a suboptimal solution. During our research this was one of the first things we ran into. We have devised several ways to counter this effect, which we will now discuss.

We used a lower boundary on the amount of pheromone on every road in AS-DTSP. This prevents the chances of a road to be chosen by an ant to approach zero beyond a certain point [SH97]. Initial testing showed that using τ_0 as a lower boundary seemed to do what we wanted.

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 smooth all pheromone levels in a certain way. If the amount of pheromones on a road becomes much higher than on all other roads going out of a city, this road 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 on it.

Shaking changes the ratio between the amount of pheromone on all roads, while it ensures that the relative ordering is preserved: if $\tau_{ij}(t) > \tau_{i'j'}(t)$ holds before shaking, it also holds after shaking. The formula used for shaking is a logarithmic one:

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

See figure 1. This formula will cause pheromone values close to τ_0 to move a little towards τ_0 and higher values to move relatively more to τ_0. See figure 1. Note that τ_0 is the minimum value for τ, so the condition $\tau_{ij} \geq \tau_0$ is preserved.

A possible problem with this shake might be that it has a global character, i.e. it affects the pheromone values on all roads. When we have a big problem instance where one traffic jam occurs somewhere near the edge of the map there is

[5] Since we are not adapting distances *as if we are moving a city*, the resulting TSP structure is not Euclidian. For instance the Cauchy-Schwarz inequality does not always hold.

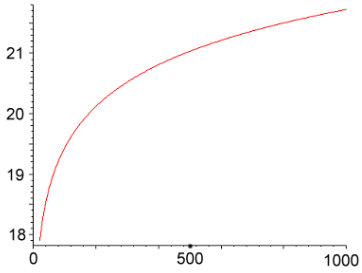


Fig. 1. Shaking function, pheromone level before and after shaking

a high probability that routes only have to change in the vicinity of the traffic jam [SH97]. So, too much information might be lost in the global shaking process. For this reason we define *local shaking*. Here, the same formula is used, but it is only applied to roads that are closer than $p \cdot MaxDist$ to one of the two cities where the traffic jam is formed between. MaxDist is the maximum distance between any two cities in the original problem (without traffic jams) and $0 < p < 1$. In pseudo code:

if the distance between cities (a, b) changed then
  for every road (i, j) **do**
    if $(d_{ai} < p \cdot MaxDist) \vee (d_{bi} < p \cdot MaxDist) \vee$
      $(d_{aj} < p \cdot MaxDist) \vee (d_{bj} < p \cdot MaxDist)$ **then**
      $\tau_{ij} = \tau_0 \cdot (1 + log(\tau_{ij}/t_0))$
    endif
  endfor
endif

Note we only introduce traffic jams on roads that are in the current best tour of the salesman. Therefore these two cities are generally close to each other and the effect of local shaking with the mentioned values for p and global shaking differ therefore significantly. Another observation is that for $p = 0$ no pheromone values are affected. Global shaking is equivalent with $p = 1$ since all pheromone values are now subject to shaking.

In this paragraph, we discussed our DTSP and presented the proposed changes to AS-TSP. Summarizing, these are:[6]

- After pheromone evaporation, we make sure that the amount of pheromone does not become less than τ_0: $\forall i, j, i \neq j : \tau_{ij} = max(\tau_{ij}, \tau_0)$.
- We introduce a shaking operator, that through smoothing of a part of the pheromone matrix increases the amount of exploration done after a change has occurred. Depending on the setting of the 'shaking-percentage' p, this operator acts globally with $p = 1$ or locally with $0 < p < 1$. Setting p to zero yields no shaking.

[6] A complete high level description of AS-DTSP is given in [Eyc01].

4 Test setup

A problem with 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. As a result we had to choose our own criteria to measure the performance of an algorithm.

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 better route that does not contain the congested road. Because all traffic jams in our tests are introduced in a few incremental steps, this is visible in the graphs as low up-peaks: before the traffic jam is at its highest point, the ants have already changed their route.

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 or low down-peaks, meaning that some more optimisations are being performed before the next change.

And finally the values for the length of the complete tour should be as low as possible. For 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.

We tested AS-DTSP with two different case studies, one with 25 cities, the other with 100 cities. For both case studies we compared five approaches:

- *Original* is the original AS-TSP with a lower bound on the pheromone values: here, no special measures are taken when a traffic jam occurs.
- *With Reset* is a copy of *original*, but when a traffic jams occurs, we reset the complete pheromone matrix to τ_0.
- *With Shake* is equal to *original* extended with shaking as explained in the previous paragraph: both global with $p = 1$ and local shaking with $p = 0.1$ and $p = 0.25$ are tested.

4.1 25 city problem

The problem instance used for this case study has 25 cities placed on a 100x100 grid. See [Eyc01] for the problem data. 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 in the best solution found. Using this technique we are sure that all traffic jams have influence on the best solution so far. We will introduce three traffic jams. In this case study there are

two interesting intervals: from iteration 100 to 150 one traffic jam is introduced on a certain road. From iteration 200 to 250 this traffic jam disappears, while simultaneously two others are introduced on other roads.

All traffic jams appear and disappear in increments and decrements of 10 units, one incremental step every 5 iterations. This way the algorithms get a chance to respond to small changes. If a traffic jam is introduced all at once, an algorithm cannot show that it is able to switch its route even while the traffic jam is still growing.

The AS-parameters were set to: $\alpha = 1, \beta = 6, m = n = 25, Q = 100, \tau_0 = 10^{-6}$.

4.2 100 city problem

To check how DTSP performs on a similar, but extended problem, we constructed a bigger and more dynamic test case. This case study has *one* static 100 city TSP problem at its foundation. See [Eyc01] for the problem data. This instance was constructed by placing 100 cities randomly in a 100x100 grid. 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 on a road in the currently best route. This orderly way of introducing jams is not realistic, but it has the advantage that we can clearly compare approaches.

Assume a solution to the problem without traffic jams, which is reached after some time of optimisation, has length x. Now we start introducing a traffic jam every 50 iterations. For every newly created traffic jam we simultaneously remove the previous one. The result is that without any further optimisation, the length of the original solution is x + *length of one full traffic jam*. Because all

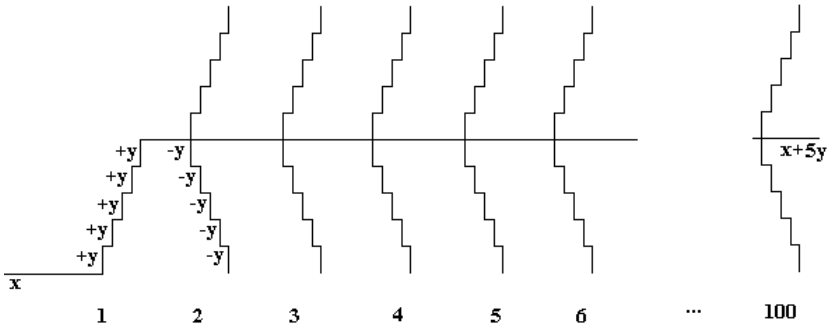


Fig. 2. Traffic jam appearance and disappearance in 100 city problem. The tour length of the fixed solution during the dynamic part of the simulation is $x + 5y$, except for a small time period in the beginning.

jams are created in 5 equal steps of y units in our test setup, there is always a $5 * y$ traffic jam (except for the first 20 iterations). This process of appearing and disappearing traffic jams is shown in figure 2. Using a pre-established arbitrary ordering of the cities, in each simulation the first traffic jam occurs on the outgoing road of city number 1, the second on the outgoing road of city number 2, and so on. The traffic jams disappear in the same order. This way we have problems that are somewhat similar each run. Every experiment runs for 5000 iterations after the start-up period, since exactly 100 traffic jams are created. Each experiment is done 10 times.

For this problem we will judge our strategies on the low and high peaks as well as the average tour length during the dynamic part of the simulation. Good scores will require good reactions to both kinds of change. For a road with an increased road length an alternative route becomes worthwhile during its steady growth. Contrarily, a road with a decreasing road length might become attractive to re-enter into the tour.

The structure of the dynamics of this problem has as an advantage that it is meaningful to compare our strategies with the strategy of ignoring all changes. This strategy simply means using the solution to the static problem and putting up with the traffic jams as they occur. Since the dynamics of the problem involves no more than two traffic jams at all times, the problem can be said to 'wobble around' the static instance. For this reason, the seemingly trivial strategy of ignoring all changes gives us a means of comparison.

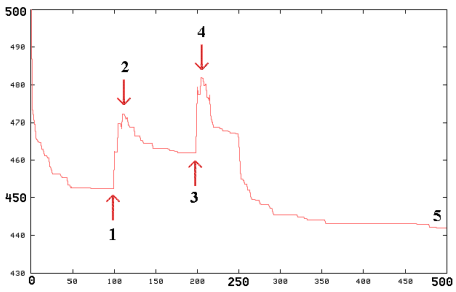
The stepsize y is set to 10. The AS-parameters were set to: $\alpha = 1, \beta = 6, m = n = 100, Q = 100, \tau_0 = 10^{-6}$.

5 Experimental results

5.1 25 city problem

The average tour length for *shake,* $p = 1$ as well as some points for which the scores will be measured are shown in figure 3. Until point 1 this problem instance is static. Two interesting points are located directly after 100 and 200 iterations: those are the dynamic parts of the problem. At both times, we see a peak in the graph. Low peaks suggest a quick recovery from a growing traffic jam. Two other interesting values that say something about the capability to recover from changes are the result right before the second peak and the result after the last iteration.

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 change has occurred, there are 250 iterations left to reach that value. The scores of three of the five strategies during the entire simulation are given in figure 4. The results on the four checkpoints are shown in figure 5.
The results are very close to each other, and it is difficult to point out a clear winner. Although *reset* has multiple best values, some others are very close, like



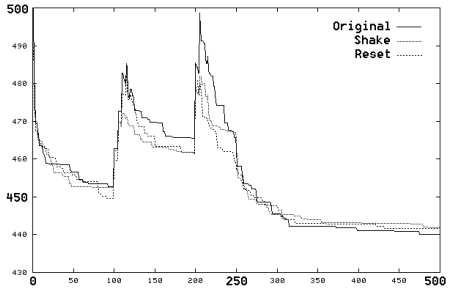


Fig. 3. Example of tour length during 500 iterations, averaged over 10 runs with locations of interesting checkpoints 2-5 marked. Results are from *Shake, ($p = 1$)* strategy.

Fig. 4. The results for *Orginal*, *Shake ($p = 1$)*, and *Reset* averaged over 10 runs.

Strategy	*2*	*3*	*4*	*5*
Original	485.2	465.4	498.7	**440.1**
With Shake, $p = 0.1$	477.7	463.0	489.0	443.4
With Shake, $p = 0.25$	479.5	465.0	485.0	440.6
With Shake, $p = 1$	**472.0**	461.8	481.8	441.9
With Reset	483.6	**461.2**	**480.0**	441.6

Fig. 5. Results at various checkpoints of the 25 city DTSP.

shake. As described in §4.1, the first change around point 2 is less dynamic than the second change around point 4. This can explain the fact that *reset* scores well at point 4, while it does not score well at point 2. For a major change it is better to reset the pheromone matrix. For point 2 the three shake algorithms perform best. For this change it is better to retain some of the pheromone matrix (compared with *reset*), but not all of it (compared with *original*). For the highly dynamic situation at point 4 the relative scores make sense when we consider the amount of exploration involved. *Reset* has to rely fully on exploration and achieves the best score. It is closely followed by *global shake*, that also uses a lot of exploration. In the middle are the *local shake* strategies. Last is *original* that does not increase exploration because of changes. An interesting observation is that all strategies score comparable at point 5. Their performance is also comparable to an AS-TSP that did not encounter any dynamics. During the traffic jams *original* performs worse than all other algorithms, but in the end it reaches the best score.

Based on this case study we see support for the claim made by [BDT99] and [GBMS00] that ant systems are very robust and even able to solve dynamic problems. Next to that, we have also shown that extending AS-TSP does improve performance in certain dynamic situations.

5.2 100 city problem

For the problem described in §4.2 the average results during 300 iterations are given in figure 6 for two strategies. Note, that the first 600 iterations were used as a start-up duration during which the five AS approaches could settle. No scores were recorded during this period.

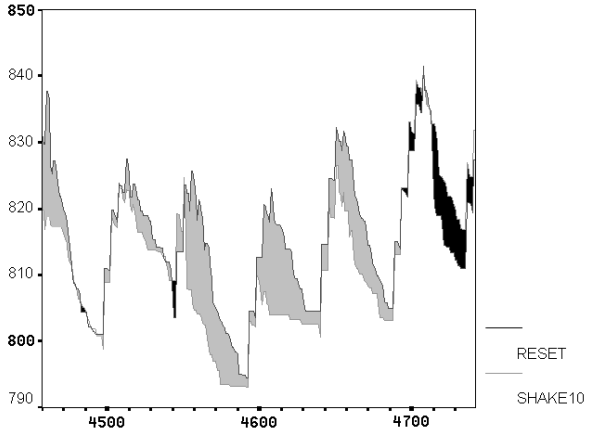


Fig. 6. Comparison of average tour length for *Reset* and *Shake ($p = 0.1$)* during an average part of the simulation. The grey area indicates the latter outperforming the first and the black vice versa.

As can be seen the consecutive growth and decrease of traffic jams lead to alternating up- and down-peaks. In comparison with the previously described test case, the traffic jams are fewer iterations apart. Usually the AS has not converged when the next traffic jam occurs. For each of the strategies the average tour length, average down-peak, and average up-peak are given.

Strategy	*Avg peak*$^-$	*Avg length*	*Extra length*	*Relative %*	*Avg peak*$^+$
Fixed without t.jam	N/A	781.6	0.0	0 %	N/A
Fixed with t.jam	N/A	831.5	49.9	100 %	N/A
Original	805.5*	813.9*	32.3	65 %	830.8
With Shake, $p = 0.1$	**803.0**	**811.8**	**30.2**	**60 %**	**830.2**
With Shake, $p = 0.25$	803.1	812.7	31.1	62 %	831.8*
With Shake, $p = 1$	805.3*	815.6*	34.0	68 %	834.9*
With Reset	804.7*	815.5*	33.9	68 %	835.5*

Fig. 7. Results on the 100 city DTSP. In the down-peak, average length and up-peak columns the scores that differ significantly from the best are marked with "*".

The fixed strategy has an optimised tour length of 781.6. With a traffic jam of length 50 the same solution obviously has a length of 831.6 (resulting in an average of 831.5). It is clear that all AS strategies are able to react to the dynamics and are able to significantly improve upon the fixed strategy. As can be seen, the local shake strategies achieve the best results, although they are closely followed by the other AS strategies. Interpreting these results in the context of our earlier findings, the level of dynamics of this problem seems to be intermediate. While it is beneficial to retain some of your pheromone matrix, exploration near the site of change needs to be encouraged.

Interestingly, *original* comes in third with respect to average performance. A likely explanation of this fact is that in this bigger problem it is very expensive to throw away information as *reset* and *global shake* do. Exploration done far away from the directly affected area will seldom lead to an improvement.

6 Conclusion

In our version of the Dynamic TSP, traffic jams lead to increased travel times on certain roads. By allowing the traffic jams to grow in a stepwise manner we can compare the reaction times of different strategies.

We have looked at various versions of AS applied to two different instances of the dynamic travelling salesman problem. The most striking effect that we have observed is that every strategy we tried performed reasonably well on both problems. This supports the claim we were investigating as stated in §1: Ant Systems are able to perform well on dynamic problems.

But when looking closer at the results, we also were able to achieve the second goal of our research. We created our AS-DTSP by extending AS-TSP through the addition of the shake routine. Especially the strategies with 'local shake' performed well. The higher the frequency of the dynamics gets, the more important it is to preserve some of the information that was collected earlier in the pheromone matrix. This is why resetting the pheromone matrix yields poor results for the 100 city problem: all information is lost and new changes occur before the algorithm is able to find a good solution. The *local shake* algorithm has as a strength that it combines exploitation of the developed pheromone matrix and biased exploration within the area around a change. With the shake parameter p the size of this area can be tweaked.

7 Recommendations for future research

A common problem when looking at Ant Systems is the large number of parameters in the algorithm. As we used settings from literature, we did not look at the influence of these regarding dynamic problems.

Several ideas on the introduced shaking routine present interesting leads for further research. First, the shake parameter p was changed a few times ($p = 0.1, 0.25, 1$) in this research, but requires more research in our opinion. We expect the optimal setting of this parameter to be related to problem attributes, such

as size and structure, and change attributes, such as change frequency, duration, magnitude, and location.

The ratio behind our shaking routine is to increase the exploration after the occurrence of a change, while at the same time limiting this exploration to the affected area. Alternative ways to implement this thought, such as a more smoothed manner of shaking (e.g. the η- and τ-strategy presented in [GMS01]) or the application of a local updating rule ([DG97]) near the site of change, might be worth considering for our DTSP.

A third option to deal with changes does not involve smart adaptation of the pheromone matrix but smart utilisation of it. An important characteristic of an Ant Colony System is its ability to directly balance the ratio of exploration versus exploitation [DG97]. Also for our DTSP it seems logical to increase this ratio around the region most affected by a change.

References

[BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence. From Natural to Artificial Systems*. Studies in the Sciences of Complexity. Oxford University Press, 1999.

[BR01] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. Technical Report TR/IRIDIA/2001-13, IRIDIA, 2001.

[DC99] Marco Dorigo and Gianni Di Caro. Ant algorithms for discrete optimization. *Artificial life*, 5(2):137–172, 1999.

[DG97] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 26(1):29–41, 1996.

[Eyc01] Casper Joost Eyckelhof. Ant systems for dynamic problems, the TSP case - ants caught in a traffic jam. Master's thesis, University of Twente, The Netherlands, August 2001. Available throug http://www.eyckelhof.nl.

[GBMS00] Michael Guntsch, Jurgen Branke, Martin Middendorf, and Hartmut Schmeck. ACO strategies for dynamic TSP. In Marco Dorigo et al., editor, *Abstract Proceedings of ANTS'2000*, pages 59–62, 2000.

[GM01] M. Guntsch and M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*. Springer Verlag, 2001.

[GMS01] Michael Guntsch, Martin Middendorf, and Hartmut Schmeck. An ant colony optimization approach to dynamic TSP. In Lee Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 860–867, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[SH97] T. Stützle and H. Hoos. Improvements on the ant system: Introducing MAX(MIN) ant system. In G.D. Smith, N.C. Steele, and R.F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 245–249. Springer-Verlag, 1997.