The Influence of Run-Time Limits on Choosing Ant System Parameters

Krzysztof Socha

IRIDIA, Université Libre de Bruxelles, CP 194/6,
Av. Franklin D. Roosevelt 50, 1050 Bruxelles, Belgium
`ksocha@ulb.ac.be`
`http://iridia.ulb.ac.be`

Abstract. The influence of the allowed running time on the choice of the parameters of an ant system is investigated. It is shown that different parameter values appear to be optimal depending on the algorithm run-time. The performance of the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) on the University Course Timetabling Problem (UCTP) – a type of constraint satisfaction problem – is used as an example. The parameters taken into consideration include the type of the local search used, and some typical parameters for $\mathcal{MM}$AS – the τ_{min} and ρ. It is shown that the optimal parameters depend significantly on the time limits set. Conclusions summarizing the influence of time limits on parameter choice, and possible methods of making the parameter choice more independent from the time limits, are presented.

1 Introduction

Ant Colony Optimization (ACO) is a metaheuristic proposed by Dorigo et al. [1]. The inspiration of ACO is the foraging behavior of real ants. The basic ingredient of ACO is the use of a probabilistic solution construction mechanism based on stigmergy. ACO has been applied successfully to numerous combinatorial optimization problems including the traveling salesman problem [2], quadratic assignment problem [3], scheduling problems [4], and others. In this paper we focus on the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System [5] – a version of ACO.

ACO, similarly to any other metaheuristic, may be parameterized. This means that in order to be able to deliver optimal performance, the ant algorithm uses a number of parameters that precisely define its operation. The parameters are usually chosen specifically for the class of problems the ant algorithm is to tackle. Usually, the optimal (or near optimal) parameters are chosen by the trial-and-error procedure. As the number of trials necessary to fine-tune the parameters is usually quite high, it is often the case that limited time is used for each algorithm run. The best parameters found are then used for tackling actual problems. In turn, the actual problem solving runs tend to be much longer.

We do not focus on the actual results obtained by the algorithm, but only on choosing the best set of parameters optimizing algorithm performance within allowed run-time limit. This paper attempts to show that optimal parameters for the ant algorithm depend significantly on the time given it to run. Hence, the parameter fine-tuning done with run-times significantly shorter than actual problem solving runs may cause choosing suboptimal parameters. It is assumed that for the time limits investigated, the algorithm does not reach optimality – it does not find the optimal solution.

The remaining part of the paper is organized as follows: Section 2 briefly presents the example problem used for evaluating the performance of the ant algorithm. Section 3 presents the ant system used to solve the problem – the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. The main design considerations and parameters used are highlighted. Section 4 presents the local search used by the algorithm, and discusses how time limits imposed determine the choice of the local search type. Section 5 discusses some other parameters used by the algorithm, and presents the relationship between their optimal values and the time limits imposed. Finally, Section 6 summarizes the findings and presents the conclusions drawn.

2 Example Problem

The problem used to illustrate the thesis of this paper, is the University Course Timetabling Problem (UCTP) [6–8]. It is a type of constraint satisfaction problem. It consists of a set of n events $E = \{e_1, \ldots, e_n\}$ to be scheduled in a set of i timeslots $T = \{t_1, \ldots, t_i\}$, and a set of j rooms $R = \{r_1, \ldots, r_j\}$ in which events can take place. Additionally, there is defined a set of students S who attend the events, and a set of features F satisfied by rooms and required by events. Each student is already preassigned to a subset of events. A feasible timetable is one in which all events have been assigned a timeslot and a room, so that the following hard constraints are satisfied:

– no student attends more than one event at the same time;
– the room is big enough for all the attending students and satisfies all the features required by the event;
– only one event is taking place in each room at a given time.

In addition, a feasible candidate timetable is penalized equally for each occurrence of the following soft constraint violations:

– a student has a class in the last slot of the day;
– a student has more than two classes in a row (one penalty for each class above the first two);
– a student has exactly one class during a day.

The infeasible timetables are worthless and are considered equally bad regardless of the actual level of infeasibility. The objective is to minimize the number of soft constraint violations (#scv) in a feasible timetable. The solution to the UCTP is a mapping of events into particular timeslots and rooms. Fig. 1 shows an example of a timetable.

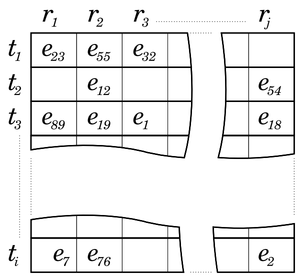

Fig. 1. Timetable of i timeslots and j rooms ($k = i \cdot j$ places). Some events from the set E have already been placed.

Two instances of the UCTP are used for illustrating the performance of the ant algorithm in this paper – `competition04` and `competition07`. These instances have been proposed as a part of the International Timetabling Competition[1]. Note that these instances are known to have a perfect solution, i.e. a solution where no hard or soft constraints are violated.

3 Algorithm Description

The basic mode of operation of the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System used for the experiments is as follows. At each iteration of the algorithm, each of the m ants constructs a complete assignment C of events into timeslots and rooms. Following a pre-ordered list of events, the ants choose the timeslot and room for the given event probabilistically, guided by stigmergic information. This information is in the form of a matrix of *pheromone* values $\tau : E \times T \times R \rightarrow \mathbf{R}^+$, where E is the set of events, T is the set of timeslots, and R is a set of rooms. In order to maintain simplicity of the notation, let us call the timeslot-room combination a *place*. The pheromone matrix becomes then of form: $\tau : E \times P \rightarrow \mathbf{R}^+$, where P is the set of k places, and $k = |P| = |T| \cdot |R| = i \cdot j$.

Also, some problem specific knowledge (heuristic information) is used by the algorithm. The place for an event (i.e. the timeslot-room combination) is chosen only from the ones that are suitable for the given event - placing the event there will not violate any hard constraint. If at some point of time during the construction of the assignment there is no such a place available, a list of timeslots is extended by one, and the event is placed in one of the rooms of this additional timeslot. This of course results in an infeasible solution[2] as the number of timeslots used from now on exceeds i. This also means that the pheromone matrix has to be extended as well. It is done by creating an extended set T' of i' timeslots, and consequently a new extended set P' of k' places. The new pheromone matrix is defined as $\tau : E \times P' \rightarrow \mathbf{R}^+$. Note that initially it is assumed that $i' = i$ and $k' = k$.

[1] `http://www.idsia.ch/Files/ttcomp2002/`

[2] Only for this particular ant, and only in this iteration.

Once all the ants have constructed their assignment of events into places, a local search routine is used to further improve the solutions. More details about local search routine are provided in Sec. 4. Finally the best solution of each iteration is compared to the global best solution found so far. If the iteration best solution is better than the global best, it is replaced. Only the global best solution is used for the pheromone update.

If the differences between extreme pheromone values were too large, all ants would almost always generate the same solutions, which would mean algorithm stagnation. The $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System introduces upper and lower limits on the pheromone values – τ_{max} and τ_{min} respectively [5] – that prevent this. The maximal difference between the extreme levels of pheromone may be controlled, and thus the search intensification versus diversification may be balanced. The pheromone update rule is as follows (for the particular case of assigning events e into places p):

$$\tau_{(e,p)} \leftarrow \begin{cases} (1-\rho)\cdot\tau_{(e,p)} + \tau_{fixed} & \text{if } (e,p) \text{ is in } C_{global\_best} \\ (1-\rho)\cdot\tau_{(e,p)} & \text{otherwise,} \end{cases} \tag{1}$$

where $\rho \in [0,1]$ is the evaporation rate, and τ_{fixed} is the pheromone update value. Pheromone update is completed using the following:

$$\tau_{(e,p)} \leftarrow \begin{cases} \tau_{min} & \text{if } \tau_{(e,p)} < \tau_{min}, \\ \tau_{max} & \text{if } \tau_{(e,p)} > \tau_{max}, \\ \tau_{(e,p)} & \text{otherwise.} \end{cases} \tag{2}$$

The pheromone update value τ_{fixed} is a constant that has been established after some experiments with the values calculated based on the actual quality of the solution. The function q measures the quality of a candidate solution C by counting the number of constraint violations. According to the definition of $\mathcal{MM}$AS, $\tau_{max} = \frac{1}{\rho}\cdot\frac{g}{1+q(C_{optimal})}$, where g is a scaling factor. Since it is known that $q(C_{optimal}) = 0$ for the considered test instances, we set τ_{max} to a fixed value $\tau_{max} = \frac{1}{\rho}$. We observed that the proper balance of the pheromone update and the evaporation rate was achieved with a constant value of $\tau_{fixed} = 1.0$, which was also more efficient than the calculation of exact value based on quality of the solution.

4 Influence of Local Search

It has been shown in the literature that ant algorithms perform particularly well, when supported by a local search (LS) routine [2, 9, 10]. There were also attempts to design the local search for the particular problem tackled here (the UCTP) [11]. Here, we try to show that although adding an LS to an algorithm improves the results obtained, it is important to carefully choose the type of such LS routine, especially with regard to algorithm running time limits imposed.

The LS used here by the $\mathcal{MM}$AS solving the UCTP consists of two major modules. The first module tries to improve an infeasible solution (i.e. a solution that uses more than i timeslots), so that it becomes feasible. Since its main purpose is to produce a solution that does not contain any hard constraint violations and that fits into i timeslots, we call it `HardLS`. The second module of the LS is run only if a feasible solution is available (either generated by an ant directly, or obtained after running `HardLS`). This module tries to increase the quality of the solution by reducing number of the soft constraint violations (#scv), and hence is called `SoftLS`. It does so by rearranging the events in the timetable, but any such rearrangement must never produce an infeasible solution.

The `HardLS` module is always called before calling the `SoftLS` module, if the solution found by an ant is infeasible. Also, it is not parameterized in any way, so in this paper we will not go into details of its operation.

`SoftLS` rearranges the events aiming at increasing the quality of the already feasible solution, without introducing infeasibility. This means that an event may only be placed in timeslot $t_{l:l\leq i}$. In the process of finding the most efficient LS, we developed the following three types of `SoftLS`:

- **type 0** – The simplest and the fastest version. It tries to move one event at a time to an empty place that is suitable for this event, so that after such a move the quality of the solution is improved. The starting place is chosen randomly, and then the algorithm loops through all the places trying to put the events in empty places until a perfect solution is found, or until in the last $k = |P|$ iterations there was no improvement.
- **type 1** – Version similar to the `SoftLS type 0`, but also enhanced by the ability to swap two events in one step. The algorithm not only checks, if an event may be moved to another empty suitable place to improve the solution, but also checks, if this event could perhaps be swapped with any other event. Only moves (or swaps) that do not violate any hard constraints and improve the overall solution are accepted. This version of `SoftLS` usually provides a greater solution improvement than the `SoftLS type 0`, but also a single run takes significantly more time.
- **type 2** – The most complex version. In this case, as a first step, the `SoftLS type 1` is run. After that, the second step is executed: the algorithm tries to further improve the solution by changing the order of timeslots. It attempts to swap any two timeslots (i.e. move all the events from one timeslot to the other without changing the room assignment), so the solution is improved. The operation continues until no swaps of any two timeslots may further improve the solution. The two steps are repeated until a perfect solution is found, or neither of them has produced any improvement. This version of `SoftLS` is the most time consuming.

4.1 Experimental Results

We ran several experiments in order to establish, which of the presented `SoftLS` types is best suited for the problem being solved. Fig. 2 presents the performance of our ant algorithm with different versions of `SoftLS`, as a function of time limit

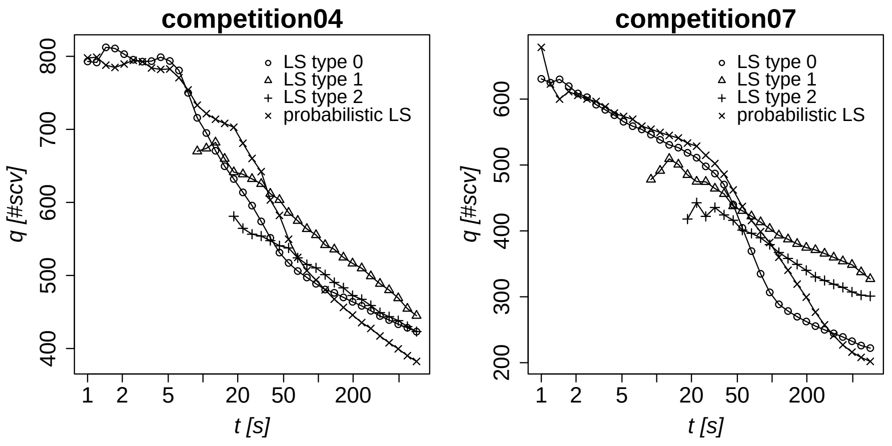

Fig. 2. Mean value of the quality of the solutions (#scv) generated by the $\mathcal{MM}$AS using different versions of local search on two instances of the UCTP – `competition04` and `competition07`.

imposed on the algorithm run-time. Note that we initially focus here on the three basic types of `SoftLS`. The additional `SoftLS` type – probabilistic LS – that is also presented on this figure, is described in more detail in Sec. 4.2.

We ran 100 trials for each of the `SoftLS` types. The time limit imposed on each run was 672 seconds (chosen with the use of benchmark program supplied by Ben Peachter as part of the International Timetabling Competition). We measured the quality of the solution throughout the duration of each run. All the experiments were conducted on the same computer (AMD Athlon 1100 MHz, 256 MB RAM) under a Linux operating system.

Fig. 2 clearly indicates the differences in performance of the $\mathcal{MM}$AS, when using different types of `SoftLS`. While the `SoftLS type 0` produces first results already within the first second of the run, the other two types of `SoftLS` produce first results only after 10-20 seconds. However, the first results produced by either the `SoftLS type 1` or `type 2` are significantly better than the results obtained by the `SoftLS type 0` within the same time. With the increase of allowed algorithm run-time, the `SoftLS type 0` quickly outperforms `SoftLS type 1`, and then `type 2`. While in case of `competition07`, the `SoftLS type 0` remains the best within the imposed time limit (i.e. 672 seconds), in case of `competition04`, the `SoftLS type 2` apparently eventually catches up. This may indicate that if more time was allowed for each version of the algorithm to run, the best results may be obtained by `SoftLS type 2`, rather than `type 0`. It is also visible that towards the end of the search process, the `SofLS type 1` appears to converge faster than `type 0` or `type 2` for both test instances. Again, this may indicate that – if longer run-time was allowed – the best `SoftLS` type may be different yet again.

It is hence very clear that the best of the three presented types of local search for the UCTP may only be chosen after defining the time limit for a single algorithm run. The examples of time limits and appropriate best LS type are summarized in Tab. 1.

Table 1. Best type of the `SoftLS` depending on example time limits.

Time Limit	Best `SoftLS` Type	
$[s]$	`competition04`	`competition07`
5	`type 0`	`type 0`
10	`type 1`	`type 1`
20	`type 2`	`type 2`
50	`type 0`	`type 2`
200	`type 0`	`type 0`
672	`type 0/2`	`type 0`

4.2 Probabilistic Local Search

After experimenting with the basic types of `SoftLS` presented in Sec. 4, we realized that apparently different types of `SoftLS` work best during different stages of the search process. We wanted to find a way to take advantage of all of the types of `SoftLS`.

First, we thought of using a particular type of `SoftLS` depending on the time spent by the algorithm on searching. However this approach, apart from having an obvious disadvantage of the necessity of measuring time and being dependent on the hardware used, had some additional problems. We found that the solution (however good it was) generated with the use of any basic type of `SoftLS`, was not always easy to be further optimized by another type of `SoftLS`. When the type of `SoftLS` used changed, the algorithm spent some time recovering from the previously found local optimum. Also, the sheer necessity of defining the right moments, when the `SoftLS` type was to be changed was a problem. It had to be done for each problem instance separately, as those times differed significantly from instance to instance.

In order to overcome these difficulties, we came up with the idea of *probabilistic local search*. Such local search would probabilistically choose the basic type of the `SoftLS` to be used. Its behavior may be controlled by proper adjustment of the probabilities of running the different basic types of `SoftLS`. After some initial tests, we found that rather small probability of running the `SoftLS type 1` and `type 2` comparing to the probability of running the `SoftLS type 0`, produced best results within the time limit defined. Fig. 2 also presents the mean values obtained by 100 runs of this probabilistic local search. The probabilities of running each type of the basic `SoftLS` types that were used to obtain these results, are listed in Tab. 2.

Table 2. Probabilities of running different types of the `SoftLS`.

`SoftLS` Type	Probabilities	
	`competition04`	`competition07`
`type 0`	0.90	0.94
`type 1`	0.05	0.03
`type 2`	0.05	0.03

The performance of the probabilistic `SoftLS` is apparently the worst for around first 50 seconds of the run-time for both test problem instances. After that, it improves faster than the performance of any other type of `SoftLS`, and eventually becomes the best. In case of the `competition04` problem instance, it becomes the best already after around 100 seconds of the run-time, and in case of the `competition07` problem instance, after around 300 seconds.

It is important to note that the probabilities of running the basic types of `SoftLS` have been chosen in such a way that this probabilistic `SoftLS` is in fact very close to the `SoftLS type 0`. Hence, its characteristics are also similar. However, by appropriately modifying the probability parameters, the behavior of this probabilistic `SoftLS` may be adjusted, and hence provide good results for any given time limits. In particular, the probabilistic `SoftLS` may be reduced to any of the basic versions of `SoftLS`.

5 ACO Specific Parameters

Having shown in Sec. 4 that choice of the best type of local search very much depends on the time the algorithm is run, we wanted to see if this also applies to other algorithm parameters. Another aspect of the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System that we investigated with regard to the imposed time limits, was a subset of the typical $\mathcal{MM}$AS parameters: evaporation rate ρ and pheromone lower bound τ_{min}. We chose these two parameters among others, as they have been shown in the literature [12, 10, 5] to have significant impact on the results obtained by a $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System.

We generated 110 different sets of these two parameters. We chose the evaporation rate $\rho \in [0.05, 0.50]$ with the step of 0.05, and the pheromone lower bound $\tau_{min} \in [6.25 \cdot 10^5, 6.4 \cdot 10^3]$ with the logarithmic step of 2. This gave 10 different values of ρ and 11 different values of τ_{min} – 110 possible pairs of values. For each such pair, we ran the algorithm 10 times with the time limit set to 672 seconds. We measured the quality of the solution throughout the duration of each run for all the 110 cases. Fig. 3 presents the gray-shade-coded grid of ranks of mean solution values obtained by the algorithm with different sets of the parameters for four different run-times allowed (respectively 8, 32, 128, and 672 seconds)[3]. The results presented, were obtained for the `competition04` instance.

[3] The ranks were calculated independently for each time limit studied.

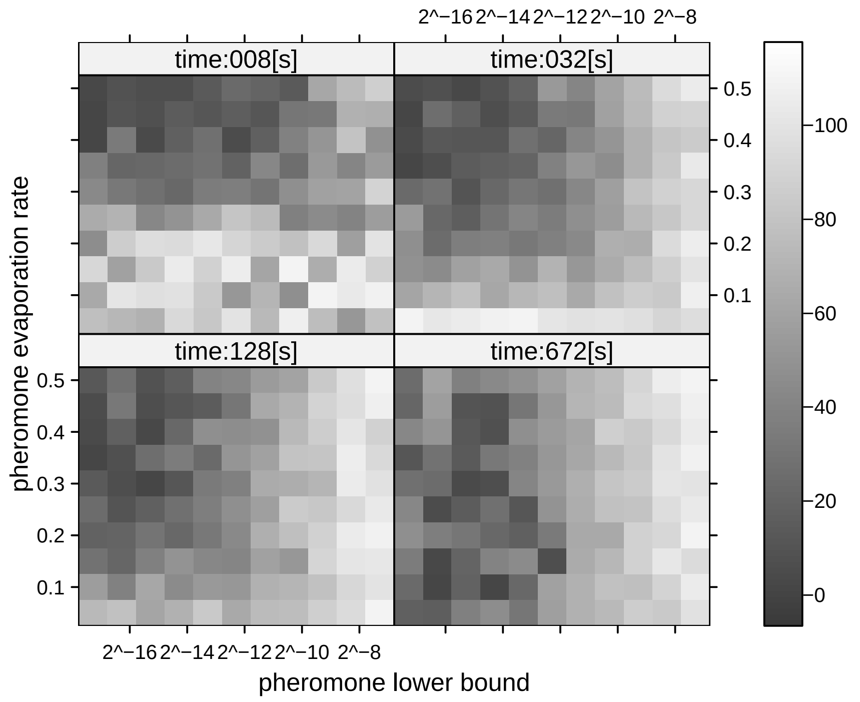

Fig. 3. The ranks of the solution means for the `competition04` instance with regard to the algorithm run-time. The ranks of the solutions are depicted (gray-shade-coded) as function of the pheromone lower bound τ_{min}, and pheromone evaporation rate ρ.

The results indicate that the best solutions – those with higher ranks (darker) – are found for different sets of parameters, depending on the allowed run-time limit. In order to be able to analyse the relationship between the best solutions obtained and the algorithm run-time more closely, we calculated the mean value of the results for 16 best pairs of parameters, for several time limits between 1 and 672 seconds. The outcome of that analysis is presented on Fig. 4. The figure presents respectively: the average best evaporation rate as a function of algorithm run-time: $\rho(t)$, the average best pheromone lower bound as a function of run-time: $\tau_{min}(t)$, and also how the pair of the best average ρ and τ_{min}, changes with run-time. Additionally, it shows how the average best solution obtained with the current best parameters change with algorithm run-time: $q(t)$.

It is clearly visible that the average best parameters change with the change of run-time allowed. Hence, similarly as in case of the local search, the choice of parameters should be done with close attention to the imposed time limits. At the same time, it is important to mention that the probabilistic method of choosing the configuration that worked well in the case of the `SoftLS`, is rather difficult to implement in case of the $\mathcal{MM}$AS specific parameters. Here,

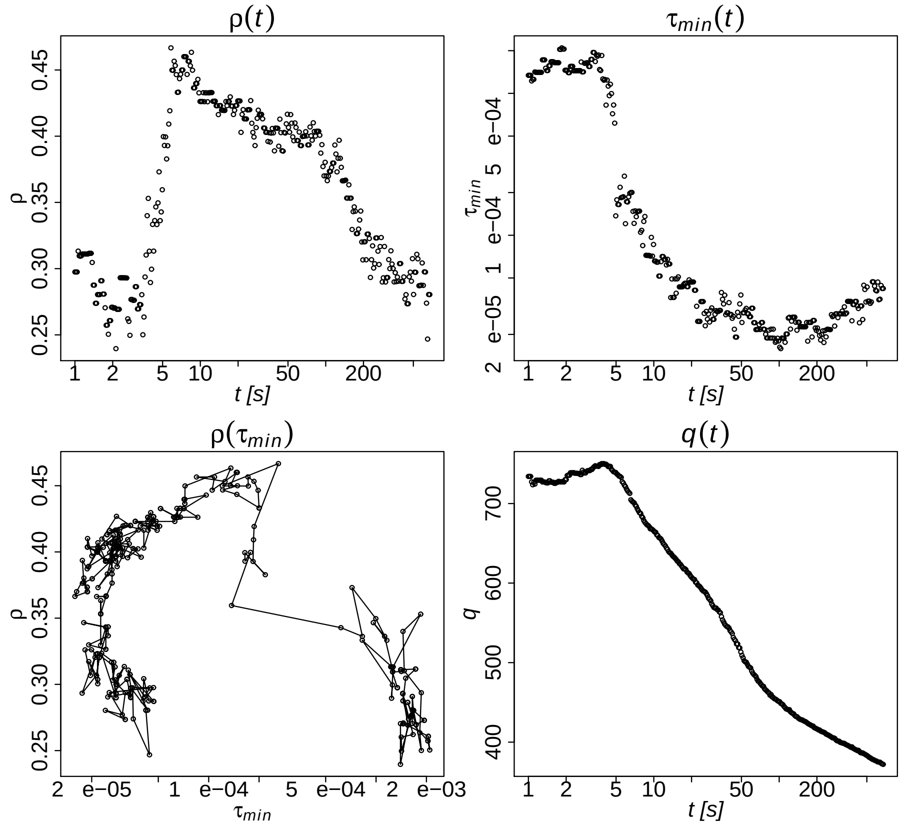

Fig. 4. Analysis of average best ρ and τ_{min} parameters as a function of time assigned for the algorithm run (the upper charts). Also, the relation between best values of ρ and τ_{min}, as changing with running time, and the average quality of the solutions obtained with the current best parameters as a function of run-time (lower charts).

the change of parameters' values has its effect on algorithm behavior only after several iterations, rather than immediately as in case of LS. Hence, rapid changes of these parameters may only result in algorithm behavior that would be similar to simply using the average values of the probabilistically chosen ones.

More details about the experiments conducted, as well as the source code of the algorithm used, and also results for other test instances that could not be included in the text due to the limited length of this paper, may be found on the Internet[4].

[4] `http://iridia.ulb.ac.be/~ksocha/antparam03.html`

6 Conclusions and Future Work

Based on the examples presented, it is clear that the optimal parameters of the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System may only be chosen with close attention to the run-time limits. Hence, the time-limits have to be clearly defined before attempting to fine-tune the parameters. Also, the test runs used to adjust the parameter values should be conducted under the same conditions as the actual problem solving runs.

In case of some parameters, such as the type of the local search to be used, a probabilistic method may be used to obtain very good results. For some other types of parameters (τ_{min} and ρ in our example) such a method is not so good, and some other approach is needed. The possible solution is to make the parameter values variable throughout the run of the algorithm. The variable parameters may change according to a predefined sequence of values, or they may be adaptive – the changes may be a derivative of a certain algorithm state.

This last idea seems especially promising. The problem however is to define exactly how the state of the algorithm should influence the parameters. To make the performance of the algorithm independent from the time limits imposed on the run-time, several runs are needed. During those runs, the algorithm (or at least algorithm designer) may *learn* what is the relation between the algorithm state, and the optimal parameter values. It remains an open question how difficult it would be to design such a *self-fine-tuning* algorithm, or how much time such an algorithm would need in order to learn.

6.1 Future Work

In the future, we plan to investigate further the relationship between different ACO parameters and run-time limits. This should include the investigation of other test instances, and also other example problems. We will try to define a mechanism that would allow a dynamic adaptation of the parameters. Also, it is very interesting to see if the parameter-runtime relation is similar (or the same) regardless of the instance or problem studied (at least for some ACO parameters). If so, this could permit proposing a general framework of ACO parameter adaptation, rather than a case by case approach.

We believe that the results presented in this paper may also be applicable to other combinatorial optimization problems solved by ant algorithms. In fact it is very likely that they are also applicable to other metaheuristics as well[5]. The results presented in this paper do not yet allow to simply jump to such conclusions however. We plan to continue the research to show that it is in fact the case.

Acknowledgments. Our work was supported by the *Metaheuristics Network,* a Research Training Network funded by the Improving Human Potential Programme of the CEC, grant HPRN-CT-1999-00106. The information provided

[5] Of course with regard to their specific parameters.

is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics **26** (1996) 29–41
2. Stützle, T., Dorigo, M.: Aco algorithms for the traveling salesman problem. In Makela, M., Miettinen, K., Neittaanmäki, P., Périaux, J., eds.: Proceedings of Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications (EUROGEN 1999), John Wiley & Sons (1999)
3. Stützle, T., Dorigo, M. In: ACO Algorithms for the Quadratic Assignment Problem. McGraw-Hill (1999)
4. Merkle, D., Middendorf, M., Schmeck, H.: Ant colony optimization for resource-constrained project scheduling. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000), Morgan Kaufmann Publishers (2000) 893–900
5. Stützle, T., Hoos, H.H.: $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. Future Generation Computer Systems **16** (2000) 889–914
6. Rossi-Doria, O., Sampels, M., Chiarandini, M., Knowles, J., Manfrin, M., Mastrolilli, M., Paquete, L., Paechter, B.: A comparison of the performance of different metaheuristics on the timetabling problem. In: Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT 2002) (to appear). (2002)
7. Socha, K., Knowles, J., Sampels, M.: A $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the University Timetabling Problem. In Dorigo, M., Di Caro, G., Sampels, M., eds.: Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms. Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany (2002)
8. Socha, K., Sampels, M., Manfrin, M.: Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In: Proceedings of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization, LNCS 2611. Volume 2611 of Lecture Notes in Computer Science., Springer, Berlin, Germany (2003)
9. Maniezzo, V., Carbonaro, A.: Ant Colony Optimization: an Overview. In Ribeiro, C., ed.: Essays and Surveys in Metaheuristics, Kluwer Academic Publishers (2001)
10. Stützle, T., Hoos, H. In: The MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Combinatorial Global Optimisation. Kluwer Academic Publishers (1998) 313–329
11. Burke, E.K., Newall, J.P., Weare, R.F.: A memetic algorithm for university exam timetabling. In: Proceedings of the 1st International Conference on Practice and Theory of Automated Timetabling (PATAT 1995), LNCS 1153, Springer-Verlag (1996) 241–251
12. Stützle, T., Hoos, H.: Improvements on the ant system: A detailed report on max-min ant system. Technical Report AIDA-96-12 – Revised version, Darmstadt University of Technology, Computer Science Department, Intellectics Group (1996)