# Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art

Krzysztof Socha, Michael Sampels, and Max Manfrin

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

**Abstract.** Two ant algorithms solving a simplified version of a typical university course timetabling problem are presented – Ant Colony System and $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. The algorithms are tested over a set of instances from three classes of the problem. Results are compared with recent results obtained with several metaheuristics using the same local search routine (or neighborhood definition), and a reference random restart local search algorithm. Further, both ant algorithms are compared on an additional set of instances. Conclusions are drawn about the performance of ant algorithms on timetabling problems in comparison to other metaheuristics. Also the design, implementation, and parameters of ant algorithms solving the university course timetabling problem are discussed. It is shown that the particular implementation of an ant algorithm has significant influence on the observed algorithm performance.

## 1 Introduction

The work presented here arises out of the Metaheuristics Network[1] (MN) – a European Commission project undertaken jointly by five European institutes – which seeks to compare metaheuristics on different combinatorial optimization problems. In the current phase of the four-year project, a university course timetabling problem is being considered.

The University Course Timetabling Problem (UCTP) is a typical problem faced periodically by every university of the world. The basic definition states that a number of courses must be placed within a given timetable, so that the timetable is feasible (may actually be carried out), and a number of additional preferences being satisfied is maximized. There are also other timetabling problems described in the literature that are similar to UCTP. They include examination timetabling [1], school timetabling [2], employee timetabling, and others [3]. They all share similar properties and are similarly difficult to solve. The general university course timetabling problem is known to be NP-hard, as are many of the subproblems associated with additional constraints [4, 5, 2]. Even when

[1] http://www.metaheuristics.org

restricting the interest to UCTP alone, it is difficult to provide a uniform and generic definition of the problem. Due to the fact that course organization as well as additional preferences may vary from case to case, the number and type of soft and hard constraints changes. Hence, the algorithmic solutions proposed for this problem usually concentrate on a particular subproblem.

Recently, in the course of the MN, five metaheuristics were evaluated and compared on instances of a certain reduction of UCTP [6]. The metaheuristics evaluated included: Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search (TA), Iterated Local Search (ILS), and Ant Colony Optimization (ACO). The $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [7, 8] algorithm for the UCTP was developed later as an approach alternative to the ACO (ACS in fact) developed for the metaheuristics comparison. The purpose of this paper is to present how ant algorithms are performing on such highly constrained problems as UCTP, and analyze the impact of choosing a particular type of ant algorithm.

The remaining part of the paper is organized as follows: Section 2 defines the reduction of the UCTP being solved. Section 3 presents the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and the Ant Colony System used for solving the UCTP. Also the major differences and similarities of the algorithms are highlighted. Section 4 presents the experiments that were performed in order to evaluate the algorithms' performance. The results obtained by ant algorithms are also compared to the results previously obtained by other metaheuristics [6]. Finally, Section 5 summarizes the findings and presents the conclusions drawn.

## 2 UCTP – Problem Definition

For the purpose of evaluating metaheuristics in the course of the MN, a reduction of UCTP has been defined [6, 8]. The problem consists of a set of $n$ events $E$ to be scheduled in a set of timeslots $T = \{t_1, \ldots, t_k\}$ ($k = 45$, 5 days of 9 hours each), a set of rooms $R$ in which events can take place (rooms are of a certain capacity), 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.

### 2.1 Problem Instances

Instances of the UCTP were constructed using a generator written by Paechter[2]. For the comparison being carried out by the MN, three classes of instance have been chosen, reflecting realistic timetabling problems of varying sizes: *small*, *medium*, *large*. They differ mostly by number of events being placed (100, 400, 400 respectively) and number of students attending these events (80, 100, 400 respectively). For details on the problem instance generator and exact parameters used to generate the three above classes of the UCTP problem see [6]. For further analysis of the performance of both ant algorithms, 10 additional problem instances were used. Those instances have been proposed as a part of the International Timetabling Competition[3] (*competition instances*). The complexity level of those competition instances may be considered to be between the *medium* and *large* instances. Note, that all the instances used for the tests are known to have a perfect solution, i.e. so that no hard or soft constraints are violated.

## 3 Ant Algorithms to Be Compared

Ant Colony Optimization (ACO) is a metaheuristic proposed by Dorigo et al. [9]. 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 quadratic assignment problem, satisfiability problems, scheduling problems etc.

There exist at least two basic variations of the ACO metaheuristic – the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System initially proposed in [7], and the ACS which is described in detail in [10, 11]. While the basic idea of operation is identical for both of those variations, there are some differences. The main difference lies in the way the pheromone is updated, which we will explain below.

Both ant algorithms used for solving the UCTP that are presented here, are based on the general ACO framework [10, 12, 7]. They both have been shown to be able to produce meaningful results for UCTP instances. The ACS has been compared with other recent metaheuristics in [6], and $\mathcal{MM}$AS has been shown to significantly outperform the random restart local search algorithm in [8]. However, the performance of the two ant algorithms has never been compared directly in order to assess the impact of the particular design and implementation. This paper provides more insight into the influence of the ant algorithm architecture and its implementation on the performance.

[2] `http://www.dcs.napier.ac.uk/~benp`
[3] `http://www.idsia.ch/Files/ttcomp2002/`

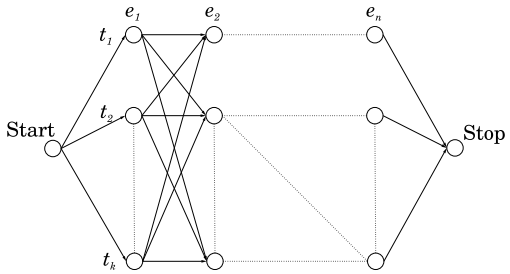


**Fig. 1.** The construction graph that the ants traverse when building an assignment of events into timeslots.

### 3.1 Differences and Similarities

The general mode of operation of both ant algorithms is very similar. At each iteration of the algorithm, each of the $m$ ants constructs a complete assignment of events to timeslots. Following a pre-ordered list of events, the ants choose the timeslot for the given event probabilistically, guided by two types of information: heuristic information and stigmergic information. The stigmergic information is in the form of a matrix of *pheromone* values $\tau : E \times T \rightarrow \mathbf{R}^+$, where $E$ is the set of events and $T$ is the set of timeslots. The pheromone values are an estimate of the utility of making the assignment, as judged by previous iterations of the algorithm. Fig. 1 presents the idea of the construction graph that the ants traverse.

After all the events have been assigned to the timeslots, a deterministic matching algorithm assigns the rooms and a candidate solution $C$ is generated. The local search routine [8] is then applied to $C$. The local search routine, which was provided separately in the course of the MN project, enables the specification of a maximal number of steps and/or maximum running time.

The algorithms compared, differ in the way they use the existing information (both stigmergic and heuristic), and the way they use local search. Also the rules of updating the pheromone matrix are different.

The $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System introduces upper and lower limits on the pheromone value. If the differences between some pheromone values were too large, all ants would almost always generate the same solutions, which would mean algorithm stagnation. The bounds on pheromone values prevent that. The maximal difference between the highest and the lowest level of pheromone may be controlled, and thus the level of search intensification versus diversification may be balanced. The pheromone update rule becomes then as follows (for the particular case of assigning events $e$ into timeslots $t$):

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

where $\rho \in [0,1]$ is the evaporation rate. Pheromone update is completed using the following:

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

The pheromone update value has been set to 1 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, $\tau_{max}$ was set to a fixed value $\frac{1}{\rho}$. The proper balance of the pheromone update and the evaporation was needed, and this was controlled by the scaling factor $g$. We observed that when $g$ was too small, the evaporation was faster than pheromone update, and pheromone levels even on the best paths finally reached $\tau_{min}$. When the values of $g$ were too large, the pheromone values on the best paths grew faster than they evaporated and finally reached $\tau_{max}$, where they were cut-off according to the $\mathcal{MAX}$-$\mathcal{MIN}$ rule. It became apparent that any value of the pheromone update that was close to $\tau_{max} \cdot \rho$ is just as good. Experimental results supported this claim. Hence, we decided that for pheromone update the constant was more efficient than the calculation of the exact value.

In ACS not only the global update rule is used, but also a special local update rule. After each construction step a local update rule is applied to the element of the pheromone matrix corresponding to the chosen timeslot $t_{chosen}$ for the given event $e_i$:

$$\tau_{(e_i,t_{chosen})} \leftarrow (1-\alpha) \cdot \tau_{(e_i,t_{chosen})} + \alpha \cdot \tau_0 \quad (3)$$

The parameter $\alpha \in [0,1]$ is the pheromone decay parameter, which controls the diversification of the construction process. The aim of the local update rule is to encourage the subsequent ants to choose different timeslots for the same given event $e_i$.

At the end of the iteration, the global update rule is applied to all the entries in the pheromone matrix:

$$\tau_{(e,t)} \leftarrow \begin{cases} (1-\rho) \cdot \tau_{(e,t)} + \rho \cdot \frac{g}{1+q(C_{global\_best})} & \text{if } (e,t) \text{ is in } C_{global\_best} \\ (1-\rho) \cdot \tau_{(e,t)} & \text{otherwise,} \end{cases} \quad (4)$$

where $g$ is a scaling factor, and the function $q$ has been described above. This global update rule is than very similar to the one used by $\mathcal{MM}$AS with the exception of not limiting the minimal and maximal pheromone level.

Another important difference between the implementations of the two algorithms, is the way that they use heuristic information. While $\mathcal{MM}$AS does not

use any heuristic information, the ACS attempts to compute it before making every move. In ACS the heuristic information is an evaluation of the constraint violations caused by making the assignment, given the assignments already made. Two parameters $\beta$ and $\gamma$ control the weight of the hard and soft constraint violations, respectively.

The last difference between the two ant algorithms concerns the use of the local search. In the case of $\mathcal{MM}$AS, only the solution that causes the fewest number of constraint violations is selected for improvement by the local search routine. Ties are broken randomly. The local search is run until reaching a local minimum or until assigned time for the trial is up – whichever happens first. The local search in case of ACS is run according to a two phase strategy: if the current iteration is lower than a parameter $j$ the routine runs for a number of steps $s_1$, otherwise it runs for a number of steps $s_2$. In case of ACS all candidate solutions generated by the ants are further optimized with the use of local search.

Tab. 1 summarizes the parameters used by the two algorithms.

**Table 1.** Parameters used by the algorithms.

| Parameter Name | $\mathcal{MM}$AS | ACS |
|---|---|---|
| $m$ | number of ants | |
| $\rho$ | pheromone evaporation | |
| $s(j)$ | number of steps of the local search | |
| $\tau_0$ | value with which the pheromone matrix is initialized | |
| $\tau_{max}$ | maximal pheromone level | - |
| $\tau_{min}$ | minimal pheromone level | - |
| $\alpha$ | - | local pheromone decay |
| $\beta$ | - | weight of the hard constraints |
| $\gamma$ | - | weight of the soft constraints |
| $g$ | - | scaling factor |

## 4 Performance of the Ant Algorithms

For each class of the problem, a time limit for producing a timetable has been determined. The time limits for the problem classes `small`, `medium`, and `large` are respectively 90, 900, and 9000 seconds. These limits were derived experimentally. All the experiments were conducted on the same computer (AMD Athlon 1100 MHz, 256 MB RAM) under a Linux operating system. The ant algorithms were compared against the best metaheuristics on those instances [6], which were the Iterated Local Search and Simulated Annealing; also against a reference random restart local search algorithm (RRLS) [8], which simply generated a random solution and then tried to improve it by running just the local search. Since all algorithms were run on the same computer, it was easy to compare their performance and a fair comparison could be achieved.

For the remaining `competition instances`, only the ant algorithms were compared against each other and against RRLS. The running time on the same computer was set to 672 seconds. The time limit has been calculated with the use of the benchmark program provided by the organizers of the International Timetabling Competition.

Tab. 2 presents the actual parameters used for running the ant algorithms. The same parameters were used for all runs of both ant algorithms.

**Table 2.** Parameter settings used by the algorithms.

| Parameter | $\mathcal{MM}$AS | ACS |
|---|---|---|
| $m$ | 10 | 10 |
| $\rho$ | 0.3 | 0.1 |
| $s(j)$ | 10 000 000 | $\begin{cases} 50\,000 & j \leq 10 \\ 20\,000 & j \geq 11 \end{cases}$ |
| $\tau_0$ | 3.3 | 10.0 |
| $\tau_{max}$ | 3.3 | - |
| $\tau_{min}$ | 0.019 | - |
| $\alpha$ | - | 0.1 |
| $\beta$ | - | 3.0 |
| $\gamma$ | - | 2.0 |
| $g$ | - | $10^{10}$ |

The ant algorithms were tested on a set of instances of the UCTP as described in Sec. 2.1. The files containing those instances as well as source code of the algorithms and summary of the results may be found on the Internet[4].

In case of the set of `medium` instances, the algorithms were run 40 times on each. For the `large` instances the algorithms were run 10 times, and for the `competition instances`, the algorithms were run for 20 independent trials.

Fig. 2 presents rank comparison of the results obtained for the set of five `medium` instances by the ant algorithms and reference algorithms: Simulated Annealing (SA) and Random Restart Local Search (RRLS). It is clear that the SA performs significantly better than any of the ant algorithms and the reference RRLS algorithm. It is however interesting to see that while $\mathcal{MM}$AS is performing better than RRLS, the ACS produced solutions significantly inferior to those of RRLS. These differences are significant at least at a $p$-value of 0.05 in a pairwise Wilcoxon test. Detailed results can also be found on [4].

Fig. 3 presents a similar comparison as Fig. 2, but for two `large` instances. The results are however rather different. It may be said with high statistical significance ($p < 0.01$) that $\mathcal{MM}$AS is performing best on these instances. ACS is performing worse, and comparably well to the Iterated Local Search (ILS) – the winner of the comparison in [6] on the problem. In this case both ant algorithms beat the performance of RRLS. SA, which was very efficient in case

[4] `http://iridia.ulb.ac.be/~ksocha/ttantcmp03.html`

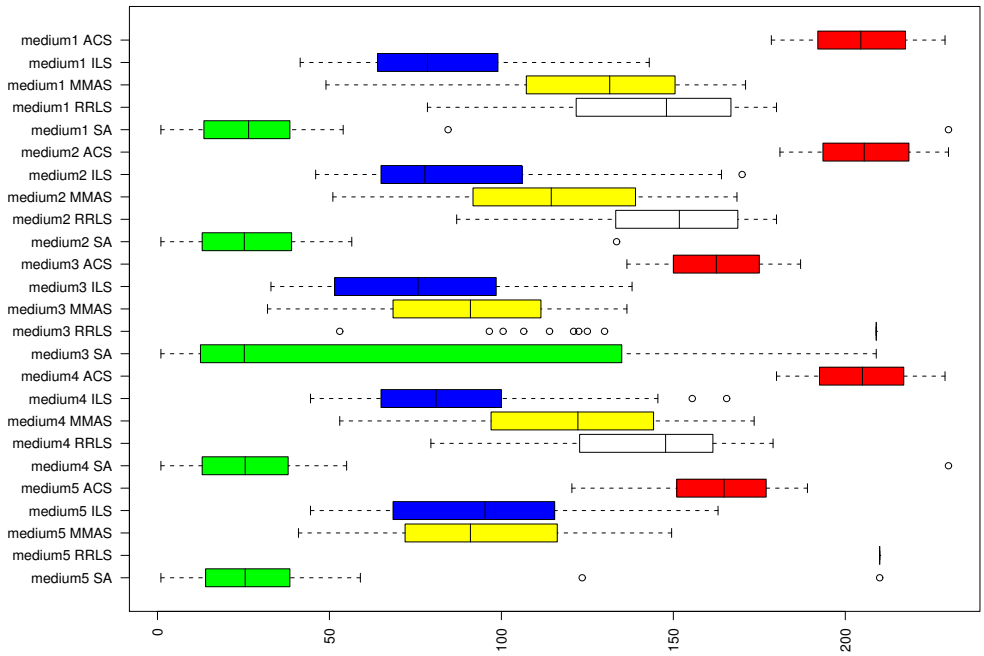


**Fig. 2.** Rank comparison of the results obtained by the two ant algorithms, leading other metaheuristics (ILS and SA), and also random local search algorithm (RRLS) on five `medium` instances of the problem. A non feasible solution is considered to be worse than any feasible solution. Hence, the rank distribution may sometimes appear as a one point distribution (single vertical line).

of `medium` instances failed to provide feasible timetables for the both `large` instances (similarly to RRLS).

Fig. 4 presents the comparison of the performance of ant algorithms on `competition instances`. There is no reference data available from other metaheuristics for these instances yet. We run the mentioned earlier RRLS algorithm on these instances, but as it did not provide feasible solutions for any of the instances, we did not include it in the comparison. Hence, the ant algorithms are compared only among themselves. The results show statistically significant better performance of $\mathcal{MM}$AS in comparison to ACS. Note that Fig. 4 contains also additional results obtained by the modified versions of the ACS and $\mathcal{MM}$AS algorithms, as described in Sec. 5.1.

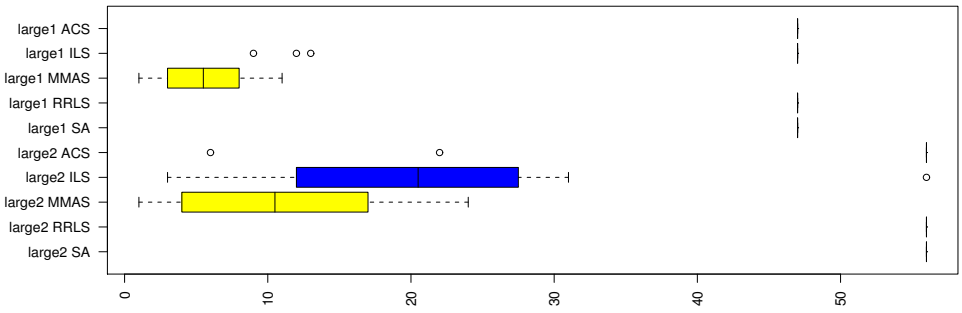


**Fig. 3.** Rank comparison of the results obtained by the two ant algorithms and leading other metaheuristics (ILS and SA), and also random local search algorithm (RRLS) on two `large` instances of the problem. A non feasible solution is considered to be worse than any feasible solution. Hence, the rank distribution may sometimes appear as a one point distribution (single vertical line). It is seen in case of SA, RRLS, and ACS for both presented instances, and also for ILS on the first `large` instance.

## 5 Conclusions

Based on the results of comparison, it is clear that the two ant algorithms perform differently. The $\mathcal{MM}$AS performs better than ACS on all instances tested. When comparing the better of the two ant algorithms to other reference algorithms, it becomes clear that for some classes of the UCTP problem, the ant algorithm proves to be very efficient. While on `medium` instances of the problem, the SA is significantly better than $\mathcal{MM}$AS, while on the `large` instances $\mathcal{MM}$AS beats any current competitor.

The difference in performance of the two ant algorithms may be due to one or more of the following factors:

- While $\mathcal{MM}$AS does not use the heuristic information, the ACS uses it extensively. The improvement provided by the heuristic information does not make up for the time lost on its calculation (which in case of the UCTP may be quite high).
- The ACS has a different strategy in using local search than $\mathcal{MM}$AS. While ACS runs the local search for a particular number of steps, the $\mathcal{MM}$AS tries always to reach the local optimum by specifying extensive number of steps.
- The $\mathcal{MM}$AS uses local search to improve only one of the solutions generated by the ants, while the ACS tries to improve all the solutions generated. While the approach of $\mathcal{MM}$AS may lead to discarding some good potential solutions, the approach of ACS may mean that two (or more) very similar solutions will be further optimized by local search, which may be an inefficient use of time.

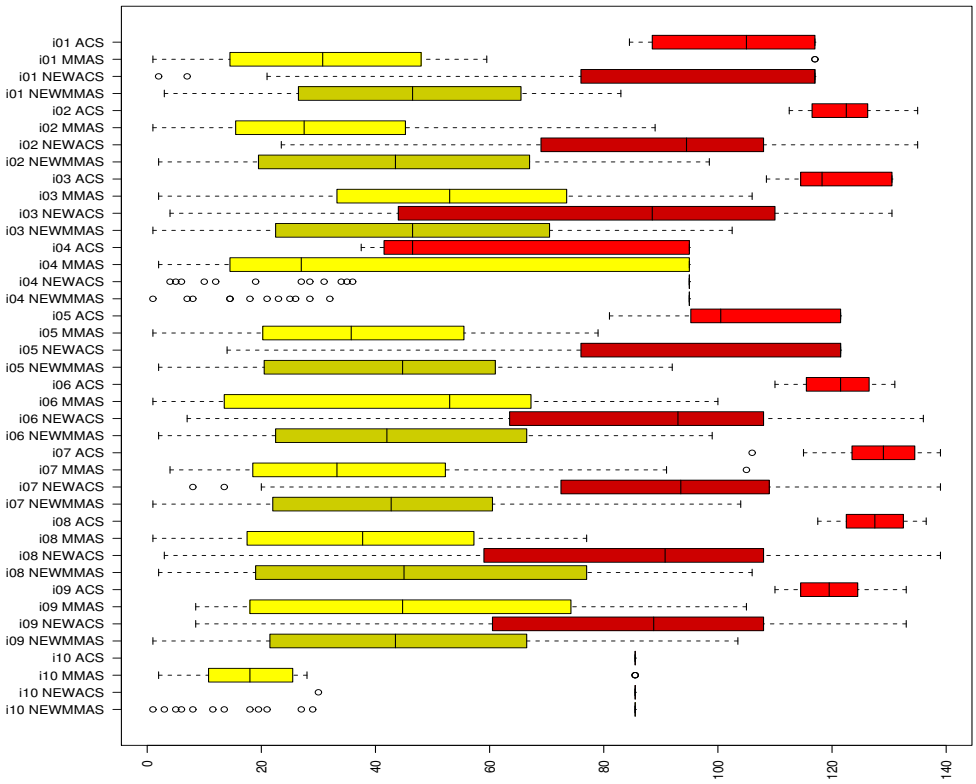


**Fig. 4.** Rank comparison of the results obtained by the two ant algorithms (ACS and $\mathcal{MM}$AS) on ten `competition instances` of the problem. Also the performance of the versions of those algorithms (NEWACS and NEW$\mathcal{MM}$AS) modified as described in Sec. 5.1 are presented.

### 5.1 Further Investigation

In order to check the hypothesis that due to the design choices made, the ACS actually takes longer to run one iteration, we calculated the number of iterations done by both algorithms. We counted the number of iterations of both ant algorithms for 5 runs on a single `competition instance`. While the $\mathcal{MM}$AS performed on average 45 iterations, in case of ACS it was only 21.6. This shows that in fact a single iteration of ACS takes more than twice the amount of time of a single $\mathcal{MM}$AS iteration. Thus, it is most probable that the first and third of the factors presented above influence the performance of the ant algorithm.

We found it interesting to investigate the topic further. Hence, we decided to run some additional experiments. This time, we tried to make the features

of both algorithms as similar as possible, to be able to see which of the factors presented above may be in fact the key issue. We modified the $\mathcal{MM}$AS so that it runs the local search on each solution generated by the ants. We also modified the ACS features so that they resembled more the features of the $\mathcal{MM}$AS. Hence, we removed the use of heuristic information, and introduced the same parameter for the use of local search as in case of $\mathcal{MM}$AS (10 000 000 steps). The results shown in Fig. 4 clearly indicate that the performance of ACS has improved significantly reaching almost the level of performance of $\mathcal{MM}$AS. Therefore, it is clear that the key factor causing differences in the original ant algorithms was the use of local search.

It is important to note that the new version of $\mathcal{MM}$AS performed best with only one ant (this was the value used to produce the results presented in Fig. 4), while the ACS obtained its best results with 10 ants. This discrepancy can be explained by the inherent properties of the two types of ant algorithms. In case of $\mathcal{MM}$AS the more ants are used in each iteration, the higher the probability that some ants will choose exactly the same path, thus not exploring the search space efficiently. In case of ACS – thanks to the local pheromone update rule – each subsequent ant in one iteration is encouraged to explore a different path. Thus, while adding more ants in case of ACS is theoretically advantageous, it is not quite the same in case of $\mathcal{MM}$AS.

The results presented indicate that there is a large dependency of the particular design decisions on ant algorithm performance. Similar algorithms using the same local search routine performed quite differently. The results also show that well designed ant algorithm may successfully compete with other metaheuristics in solving such highly constrained problems as the UCTP. Further analysis and testing is needed in order to establish in more detail the influence of all the parameters on ant algorithm performance.

**Acknowledgments.** We would like to thank Ben Paechter and Olivia Rossi-Doria for the implementation of data structures and routines for the local search. Also, we would like to thank Thomas Stützle and Marco Chiarandini for the implementation of the ILS and SA algorithms for UCTP. The information provided 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. Gaspero, L.D., Schaerf, A.: Tabu search techniques for examination timetabling. In: Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT 2000), LNCS 2079, Springer-Verlag (2001) 104–117
2. ten Eikelder, H.M.M., Willemen, R.J.: Some complexity aspects of secondary school timetabling problems. In: Proceedings of the 3rd International Conference

on Practice and Theory of Automated Timetabling (PATAT 2000), LNCS 2079, Springer-Verlag (2001) 18–29
3. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT 2000), LNCS 2079, Springer-Verlag (2001) 176–190
4. Cooper, T.B., Kingston, J.H.: The complexity of timetable construction problems. In: Proceedings of the 1st International Conference on Practice and Theory of Automated Timetabling (PATAT 1995), LNCS 1153, Springer-Verlag (1996) 283–295
5. de Werra, D.: The combinatorics of timetabling. European Journal of Operational Research **96** (1997) 504–513
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. Stützle, T., Hoos, H.H.: $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. Future Generation Computer Systems **16** (2000) 889–914
8. 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)
9. 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
10. Bonabeau, E., Dorigo, M., Theraulaz, G. Oxford University Press (1999)
11. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the travelling salesman problem. IEEE Transactions On Evolutionary Computation (1997) 53–66
12. Dorigo, M., Di Caro, G., Gambardella, L.M.: Ant algorithms for discrete optimization. Artificial Life **5** (1999) 137–172