Ann Oper Res (2012) 194:325–339
DOI 10.1007/s10479-012-1078-5

Solving the post enrolment course timetabling problem by ant colony optimization

Clemens Nothegger · Alfred Mayer · Andreas Chwatal · Günther R. Raidl

Published online: 9 February 2012

Abstract In this work we present a new approach to tackle the problem of *Post Enrolment Course Timetabling* as specified for the International Timetabling Competition 2007 (ITC2007), competition track 2. The heuristic procedure is based on Ant Colony Optimization (ACO) where artificial ants successively construct solutions based on *pheromones* (stigmergy) and local information. The key feature of our algorithm is the use of two distinct but simplified pheromone matrices in order to improve convergence but still provide enough flexibility for effectively guiding the solution construction process. We show that by parallelizing the algorithm we can improve the solution quality significantly. We applied our algorithm to the instances used for the ITC2007. The results document that our approach is among the leading algorithms for this problem; in all cases the optimal solution could be found. Furthermore we discuss the characteristics of the instances where the algorithm performs especially well.

Keywords Timetabling · Ant colony optimization · Ant system · Metaheuristics · Combinatorial optimization

C. Nothegger (✉)
Institute of Photogrammetry and Remote Sensing, Vienna University of Technology, Gusshausstr. 27-29, 1040 Vienna, Austria
e-mail: cn@ipf.tuwien.ac.at

A. Mayer · A. Chwatal · G.R. Raidl
Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstr. 9-11, Vienna, Austria

A. Mayer
e-mail: alfredmayer@gmx.at

A. Chwatal
e-mail: andy@ads.tuwien.ac.at

G.R. Raidl
e-mail: raidl@ads.tuwien.ac.at

1 Introduction

Course timetabling problems periodically arise at various universities and other educational institutions. The general course timetabling problem is known to be NP-hard and is also in practice a challenging computational task.

Here, we focus in particular on the *Post Enrolment Course Timetabling Problem* as specified for the International Timetabling Competition 2007 (ITC 2007), competition track 2. In this problem the challenge is to assign university courses to timeslots and rooms, where each assignment has to fulfill various constraints. According to McCollum et al. (2010) a problem instance for the ITC2007 (track 2) consists of

- a set of n events that are to be scheduled into 45 timeslots (5 days of 9 hours each),
- a set of r rooms, each of which has a specific seating capacity, in which the events take place,
- a set of f room-features that are satisfied by rooms and which are required by events,
- a set of s students who attend various different combinations of events,
- a set of *available* timeslots for each of the n events (i.e. not all events will be available in all timeslots), and
- a set of *precedence* requirements stating that certain events should occur before others.

The goal is to assign the n events to the time slots and rooms such that the following hard constraints are fulfilled:

1. No student should be required to attend more than one event at the same time.
2. In each case the room should be large enough for all the attending students and should satisfy all of the features required by the event.
3. Only one event is put into each room in any timeslot.
4. Events should only be assigned to timeslots that are pre-defined as "available" for those events.
5. Where specified, events should be scheduled to occur in the correct order.

It is required that all solutions fulfill all of the hard constraints. Eventually, some events may be left unassigned if they cannot be assigned without hard constraint violations. A solution is defined to be *feasible* if all events are assigned without hard constraint violation. Otherwise the quality of the solution is characterized by the *Distance to Feasibility* (DTF) which is the number of students of each of the unplaced events. Besides the hard constraints, solutions should fulfill the following soft constraints:

1. Students should not be scheduled to attend an event in the last timeslot of a day.
2. Students should not have to attend three (or more) events in successive timeslots occurring in the same day.
3. Students should not be required to attend only one event in a particular day.

The *Soft Constraint Penalty* (SCP) is determined as the sum of the following components:

- Count the number of students having just one class a day.
- Count the number of occurrences of a student having more than two classes consecutively (3 consecutively scores 1, 4 consecutively scores 2, 5 consecutively scores 3, etc.).
- Count the number of occurrences of a student having a class in the last timeslot of the day.

Due to the greater importance of the hard constraints, two solutions are first compared by their DTF, and only in the case of equal DTFs the SCP is considered for comparison.

2 Related work

A comprehensive review of timetabling problems can be found in Lewis (2008), Schaerf (1999), Burke and Petrovic (2002), MirHassani and Habibi (2011). Many metaheuristics have been applied successfully to diverse variants of this problem. For a particular problem, various Ant Colony Optimization (ACO) algorithms have been developed, e.g. a Max-Min Ant System is presented in Socha et al. (2002), and an ant colony system in Rossi-Doria and Paechter (2003). In Rossi-Doria and Paechter (2003) various other metaheuristics including evolutionary algorithms, ant colony optimization, iterated local search and simulated annealing as well as diverse neighborhood structures are compared. As these algorithms perform very differently depending on various properties of the problem instances, the authors conclude that hybrid algorithms may be a promising way to tackle the problem. The most successful approaches at the First International Timetabling Competition (TTComp 2002) are described in detail by Kostuch (2005), Burke et al. (2003), Gaspero and Schaerf (2003), Chiarandini et al. (2006) and Rossi-Doria and Paechter (2004). The algorithm proposed by Kostuch (2005) first constructs a feasible solution and then applies simulated annealing to minimize the soft constraint violations. Local search strategies are successfully applied by Burke et al. (2003) and Gaspero and Schaerf (2003). A hybrid algorithm, described by Chiarandini et al. (2006), is reported to find very good results for many of the TTComp2002 instances. A memetic algorithm is presented in Rossi-Doria and Paechter (2004).

A preliminary version of our work was published in Mayer et al. (2008) and here we extend this work by a parallelization and combination with Simulated Annealing. Furthermore extended experimental results are presented.

3 The algorithm

Our approach is based on ant colony optimization (ACO) and can be more specifically classified as Ant System (AS). For a comprehensive introduction to ant colony optimization see Dorigo and Stützle (2004). In ACO algorithms artificial ants successively construct solutions based on global information (pheromones) and local information (e.g. some greedy criterion). The pheromones hence act as a probabilistic model for solution construction and are perpetually amplified by ants that have constructed high quality solutions. Pheromone evaporation counteracts early convergence by allowing exploration of the search space for a greater duration. A generic ant colony optimization procedure consists of three steps which are iterated. The steps are solution construction by the artificial ants, pheromone update and optionally daemon actions like performing a local search.

More specifically, our AS can be described by Algorithm 1. The main task in designing an ACO algorithm is to devise the pheromone structures and update rules, and to find effective local methods able to drive the process towards promising regions of the search space. The respective parts of our algorithm are described in detail in the following sections.

3.1 Pheromone information

In our algorithm ants assign events to timeslots and rooms based on two kinds of pheromone denoted by τ^{s}_{ij} and τ^{r}_{ik}. τ^{s}_{ij} is an $n \times 45$ matrix representing the relative probabilities of assigning an event i to timeslot j. Likewise, τ^{r}_{ik} is an $n \times r$ matrix representing the relative probabilities of assigning an event i to room k. The decision to store pheromone information in this way is a key-feature of the algorithm, as it avoids the usage of a much larger data

Algorithm 1: TimeTabling-AS()

```
while time limit not yet reached do
    for each ant k = 1, ..., m do
        create random permutation π^e of the events
        for each event in order π^e do
            assign event based on pheromones
        end
    end
    locally improve each constructed solution
    for each solution with a better than average score do
        pheromone amplification for assignments appearing in solution
    end
    pheromone evaporation
end
```

structure implied by a more traditional encoding using individual pheromone values for all slot/room/event combinations (see e.g. Socha et al. 2002). On the other hand it contains more information than the exclusive use of event–timeslot pheromones (e.g. Rossi-Doria et al. 2003; Socha et al. 2003).

Obviously, distinct pheromone matrices τ^{s} and τ^{r} are not as expressive as a third order pheromone tensor (τ_{ijk}) covering all combinations of events i, timeslots j, and rooms k would be. On the other hand (τ_{ijk}) can be expected to be sparse, i.e. there are few elements different from zero. Thus, it is likely that the elements τ_{ijk} can be sufficiently well approximated by $\tau_{ijk} \approx \tau^{\mathrm{s}}_{ij} \times \tau^{\mathrm{r}}_{ik}$. Furthermore we do not expect particularly strong mutual dependencies of event–room and event–timeslot relations. Suppose some course E is required to be held in a subset of the rooms including room R_1 and the students attending this course as well as some precedence constraints require the event to be scheduled early in the first day, say at one of the slots $S_1, \ldots, S_m$. There is no obvious need to express the demand to assign E to exactly R_1 when using S_1 for instance, as all rooms satisfying the requirements of E will typically be equally good in this situation w.r.t. a current partial solution. These considerations directly lead to the conclusion that the assignment to a room is typically less critical than the assignment to a timeslot, which is also supported by our experiments. If there are mutual dependencies, they are handled implicitly by the solution construction procedure.

Table 1 Comparison of different pheromone representations. For the two representations and two time limits the table lists median DTF, median SCP, standard deviation for DTF and SCP respectively and the probability of finding a feasible solution. Results were obtained from 50 runs on each of the ITC2007 instances

Time	No. of matrices	DTF	SCP	σ_{DTF}	σ_{SCP}	P(DTF = 0)
5 min	1	54	516	121	600	82
	2	0	215	82	406	92
1 h	1	0	120	5.6	172	99
	2	0	38	0	58	100

Algorithm 2: getNextPermItem(j)

```
1  if j > pos then
2      for j' = pos, ..., j do
3          if σ > ε then
4              rnd ← σ · random number
5              q ← pos
6              ξ ← 0
7              while ξ < rnd and q < 45 do
8                  ξ ← ξ + w^s_q
9                  q ← q + 1
10             end
11             σ ← σ − w^s_{q−1}
12             swap(q − 1, j')
13         else
14             // the remaining weights are all zero
15             arbitrarily choose one of the remaining
16             indices rnd
17             swap(rnd, j')
18         end
19         pos ← pos + 1
20     end
21 end
22 return π^s_j
```

To test the hypothesis that the approximation $\tau_{ijk} \approx \tau^{\mathrm{s}}_{ij} \times \tau^{\mathrm{r}}_{ik}$ is sufficient, we implemented both pheromone representations: τ_{ijk} and $\tau_{ij} + \tau_{ik}$. If the approximation is sufficient then the solution quality for the approximated representation should not be worse than the full representation. Our experiments support this hypothesis, the results are listed in Table 1. The two matrix representation actually performs better than the full representation because the algorithm executed between two and five times more iterations per time unit. The reason for this is that the single matrix is much larger than the two matrices combined and thus the construction of candidate solutions and the pheromone evaporation are more expensive to compute. Thus the gain in efficiency outweighs the lack of expressiveness.

3.2 Solution construction

The solution construction considers the events in a uniform random order and assigns each event to a feasible room and a feasible time slot in a greedy randomized way (if possible) considering the pheromone information.

Typical ACO algorithms also consider local heuristic information in addition to the pheromone information. This information is sometimes called the visibility function, especially in the case of the traveling salesman problem. We do not use any local heuristic information because this increases the randomness of the algorithm, which we believe is important for the algorithm's behavior in the presence of a significant solution backbone (cf. Sect. 4).

For each event randomized weighted permutations of the available slots and rooms (π^{s} and π^{r}, respectively) are derived by copying the pheromone matrix rows τ^{s}_i τ^{r}_i into vectors

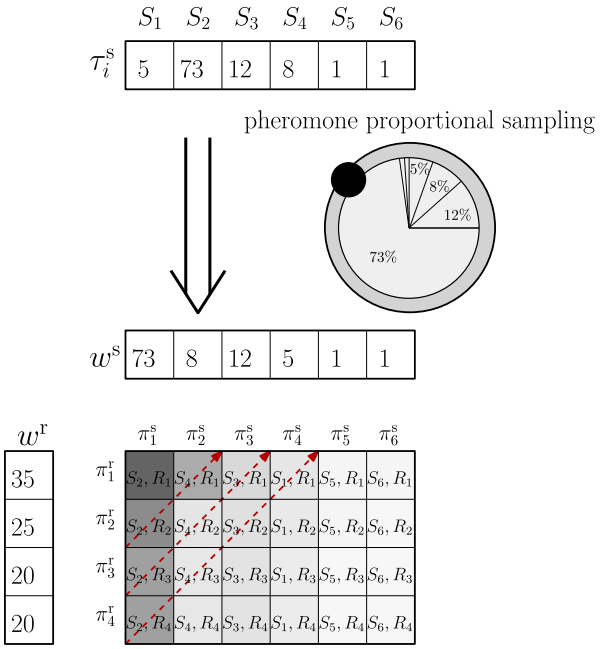

Fig. 1 Assignment of an event to a feasible room and timeslot: The pheromone matrix rows τ_i^{s} τ_i^{r} are copied into vectors w^{s} and w^{r} and sorted according to weighted random permutations π^{s} and π^{r} so that entries with higher values appear more likely at the beginning. Room/timeslot combinations are then checked in the indicated order, and the first feasible assignment is accepted

w^{s} and w^{r} and computing the weighted random permutations π^{s} and π^{r} so that entries with higher values appear more likely at the beginning. This way slots (rooms) with higher pheromone values for the current event are more likely to appear earlier than slots (rooms) with low pheromone values. This can be compared to the fitness proportional selection in genetic algorithms as for each position an item is selected (from the remaining ones) with a probability proportional to the respective pheromones.

The ant then tries to assign the current event to a slot/room combination based on their order in π^{s} and π^{r}, respectively. The first possible assignment not violating any hard constraints w.r.t. the current partial solution is accepted. To ensure that both kinds of pheromone are accounted for in a balanced way, the slot/room combinations are considered in the following order: $(\pi_1^{\mathrm{s}}, \pi_1^{\mathrm{r}}), (\pi_1^{\mathrm{s}}, \pi_2^{\mathrm{r}}), (\pi_2^{\mathrm{s}}, \pi_1^{\mathrm{r}}), (\pi_1^{\mathrm{s}}, \pi_3^{\mathrm{r}}), (\pi_2^{\mathrm{s}}, \pi_2^{\mathrm{r}}), (\pi_3^{\mathrm{s}}, \pi_1^{\mathrm{r}}), \ldots, (\pi_{45}^{\mathrm{s}}, \pi_r^{\mathrm{r}})$. See also Fig. 1.

To speed up this process the weighted random permutations are not entirely created in advance, but rather the requested elements are calculated on demand. Algorithm 2 performs this task for the slots; the room-pheromones are treated analogously. The algorithm returns the requested j-th element from the weighted random permutation π^{s}. The global array w^{s} is assumed to be filled with the respective rows of the pheromone matrices τ^{s} for the event i under consideration. Before the method is executed the first time, let $\sigma \leftarrow \sum_{l=1}^{45} w_l^{\mathrm{s}}$.

The integer variable *pos* stores the index to which the weighted permutation has already been created. If j is less than *pos* no further computation is required. Otherwise the remaining elements are calculated (lines 3–20). The function swap(i, j) exchanges the elements $\pi_i^{\mathrm{s}}, \pi_j^{\mathrm{s}}$ and $w_i^{\mathrm{s}}, w_j^{\mathrm{s}}$ respectively. In the special case of all remaining weights being zero, or close to zero by some small tolerance ϵ, some arbitrary element is chosen (line 13–17).

Note that to prepare for the next ant only *pos* needs to be reset. w^{s} and π^{s} do not need to be reinitialized for each ant. Moreover, for all but the first ants the method has a much better performance, as w^{s} and π^{s} are already roughly sorted in advance. The effect of this

presorting is that the inner-most loop will, on average, run fewer times since ξ accumulates faster. This effect is especially pronounced once the pheromone distribution has settled after a few dozens of iterations since then there are typically only a few large weights in w_q^s with the majority of weights being close to zero.

3.3 Pheromone update

After each iteration only the set of ants whose solutions are tied for the lowest DTF-score is considered for the pheromone update. For this set the average SCP-score is computed and all ants which are both members of the set and whose SCP-score is below the average score add an amount of pheromone proportional to the solution quality for the performed event/slot and event/room assignments.

In the early stage these are very few ants, typically only one, since the DTF-score will be different for most ants. In the later stages, as more and more ants produce feasible solutions, i.e. a DTF-score of zero, the number of ants depositing pheromone increases and can reach up to half the total number of ants.

The pheromone update is performed according to this formula:

$$\tau_{ij}^s \leftarrow \max(0, \tau_{ij}^s + \Delta\tau_{ij} - \Delta\tilde{\tau}_{ij}). \tag{1}$$

τ_{ij}^s is the pheromone which is stored in the pheromone matrix. $\Delta\tau_{ij}$ is the amount of pheromone added by the solution and $\Delta\tilde{\tau}_{ij}$ is a penalty term for solution components violating soft constraints. The maximum calculation avoids negative pheromone values which might otherwise appear due to the penalization w.r.t. SCP.

For the event/slot assignments, this is done in detail as follows:

$$\Delta\tau_{ij} = \begin{cases} f \cdot g & \text{if event } i \text{ and slot } j \text{ is part of the solution} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$f = \begin{cases} \frac{100}{\#\text{unplaced}} & \text{if there are unplaced events} \\ 200 & \text{otherwise} \end{cases} \tag{3}$$

$$g = \begin{cases} \frac{1000}{\text{SCP}} & \text{SCP} \neq 0 \\ 2000 & \text{otherwise} \end{cases} \tag{4}$$

The numeric constants have been determined by preliminary experiments. The numerators are chosen such that for most solutions f and g are greater than one, i.e. on the tested problem instances there are fewer than 100 unplaced events and SCP is typically below 1000. The constants used when the denominator becomes zero are chosen to be twice the denominator, which is designed to boost good assignments somewhat. Please note that the absolute values of the constants in the above formulae are not important, only their relative magnitude is important.

If an assignment is found to cause a violation of soft constraints, the involved assignments are punished accordingly

$$\Delta\tilde{\tau}_{ij} \leftarrow (1 - (1 - \gamma)^{\text{SCP}(i)}) \cdot f \cdot g, \tag{5}$$

where SCP(i) denotes the soft constraint penalty induced by event i and γ denotes the penalty factor. We used penalty factors between 0.05 (low penalty) and 0.5 (high penalty) with 0.3 being a good compromise. Note that while $\Delta\tau_{ij}$, i.e. the pheromone deposition,

is constant for the entire solution, the penalty $\Delta\tilde{\tau}_{ij}$ is assignment specific. Thus different solution components can receive different pheromone updates.

Finally, pheromone evaporation follows the standard AS method, which is

$$\tau^{\mathrm{s}}_{ij} \leftarrow (1-\rho)\tau^{\mathrm{s}}_{ij} \tag{6}$$

with ρ being the pheromone evaporation coefficient. We used values from 0.1 to 0.3 for ρ.

The pheromone update for the event–room pheromones τ^{r}_{ik} is performed analogously. In the case of stagnation, i.e. no new so-far-best solution has been found during the last 500 iterations, the pheromone values are normalized. The normalization procedure performs a linear scaling, such that the average value of the original pheromones remains the same and the deviation from this average is relatively small, e.g. approximately ten percent. Assuming that the minimum pheromone value is zero this scaling is achieved by setting

$$\tau^{\mathrm{n}}_{ij} = (\tau_{ij} - \bar{\tau}) \cdot \frac{\bar{\tau}\omega}{\max(\tau_{ij})} + \bar{\tau} \tag{7}$$

where τ^{n}_{ij} denotes the normalized pheromone value, $\bar{\tau}$ denotes the average pheromone value and ω being the desired spread around the average. Because of the linearity of the expectation operator, the mean of the scaled pheromone values is the same as the mean of the original values.

3.4 Improvement method

Often, ACO approaches benefit significantly by including a local search procedure for improving candidate solutions derived by the ants. In our algorithm we employ a best first ejection chain improvement heuristic (Glover 1992). A chain is started by moving an event which causes soft constraint violations to a different timeslot if this can be achieved without violating hard constraints and while removing at most one other event from the solution. If an event needed to be removed the chain is continued by trying to place this event using the rules described above. If there is more than one way to continue the chain, alternatives which reduce the soft constraint penalty most are tried first. If an event can be placed without ejecting another one, the chain is ended and accepted if it reduces the total soft constraint penalty. If no acceptable chains can be found within the maximum chain length, the search is aborted. We apply this procedure once an ant found a new best solution and used a maximum chain length of 16 moves.

3.5 Parallelization

ACO algorithms are easily parallelizable, since each ant constructs its solution independently. Only the solution evaluation and pheromone update after all ants have finished constructing their solution needs to be synchronous. We tested a parallel implementation of the algorithm where batches of ants are run in multiple threads. For this implementation we did not use the improvement heuristic described in Sect. 3.4, however. Instead we used a simulated annealing (SA) algorithm as described by Lewis (2010). The ejection chain based improvement heuristic is designed to deliver a best-effort "quick fix", and as such it quickly exhausts its rather restricted search space. It is also not capable of escaping local minima, and therefore not suitable for being run in parallel with the ACO algorithm. Thus the switch to SA.

The neighborhood operator randomly selects two combinations of timeslots and rooms, each determining a slot in the timetable. The timeslots are chosen such that the two selected timeslots are not identical. The content associated with these slots is then swapped. There are three cases, the first is that both slots are empty, this case is disregarded. The second case is that one of the slots is empty the other has an associated event. This results in the event being moved. And lastly the third case is that both slots have associated events, which results in these events being swapped. Moves which result in hard constraint violations are automatically rejected, moves which either improve the SCP or leave it unchanged are automatically accepted. Moves which decrease the SCP are accepted with a probability $e^{-|\Delta SCP|/T_i}$, where T_i is the current temperature. It is initially set to a fixed value T_0 and reduced every third iteration of the ACO according to the geometric cooling schedule $T_{i+1} = \delta T_i$.

The SA is run in a separate thread in parallel to the ACO. This thread is stopped after each iteration of the ACO, where the solutions from both algorithms are compared. If an improved solution is found by SA it is included in the pheromone update once, if an improved solution is found by ACO the SA is restarted with this improved solution.

4 Results and discussion

The algorithm is parameterized by α (importance of the pheromones), β (importance of local information), ρ (pheromone evaporation) and η (number of ants). Robust parameter values have been determined by preliminary experiments. It turned out that $\alpha \in [1.0, 1.1]$ yields good results, and the choices of γ and η are less critical. Lower values of γ and η reduce the time to obtain a feasible solution, but decrease the average solution quality as well. We used $\rho = 0.25$, $\gamma = 0.3$ and $\eta = 20$. As no local information is used in our algorithm, $\beta = 0$. The pheromone matrices τ^{s} and τ^{r} are initialized with the value 0.5.

The parameters we used for the SA were $T_0 = 15$ and a cooling rate $\delta = 0.995$. T_0 was estimated following the procedure outlined in Abramson et al. (1999). Since the cooling is essentially determined by the time the ACO takes for an iteration, which is dependent on instance size and the number of threads, the cooling schedule was chosen such that the number of moves per generation and the time to reach the final temperature is acceptable for all instances. Finally, in case of stagnation the SA was reheated to T_0. To detect stagnation we used the same criteria as for the ACO algorithm, i.e. no new best-so-far solution for 500 iterations.

We performed our computational experiments with the instances used for the Second International Timetabling Competition (ITC2007). The experiments with the single-threaded algorithm were performed on a Intel Xeon 5160 (3.0 GHz) using Sun Java SE 1.6.0_04 64-bit Server VM. With the benchmark-tools for the competition, we determined $t_{\max} = 299$ seconds maximum running time.

The multi-threaded algorithm was tested on three different machines, a dual Xeon 5160 (3.0 GHz, 4 cores), a dual Xeon X5355 (2.66 GHz, 8 cores) and a Core 2 Duo (2.4 GHz, 2 cores). We used the competition benchmark-tool to compensate for the different CPU speeds. For this reason the algorithm running times given in Table 4 cannot be directly compared.

Table 2 lists the results achieved for the ITC2007 instances using the single-threaded algorithm submitted to the competition. The algorithm was run 100 times for each instance to make the statistical summary data more accurate. The first two rows show that an optimum solution could be found within the given time limit in many cases. Furthermore the probability of finding a feasible solution is very high, and also the average SCP values are very promising. Within the given time limit, on average 5000 iterations have been performed.

Table 2 Results for 100 runs of the ITC2007 instances. The first two rows indicate the best results achieved within the given time limit. The distance to feasibility is denoted by DTF, soft constraint penalty by SCP. The subsequent rows list average solution values of 100 runs followed by the respective standard deviations and the probability to find a feasible solution. The last three rows show the influence of the local search improvement heuristic: percentage of solutions found by LS and average improvement by ACO and LS respectively

Instance	1	2	3	4	5	6	7	8
DTF_{best}	0	0	0	0	0	0	0	0
SCP_{best}	0	0	110	53	13	0	0	0
DTF_{avg}	237	274	0	0	0	2	0	0
SCP_{avg}	613	556	680	580	92	212	4	61
σ_{DTF}	290	369	0	0	0	9	0	0
σ_{SCP}	612	671	255	268	55	165	24	47
$P(DTF = 0)$	0.54	0.59	1.0	1.0	1.0	0.95	1.0	1.0
LS(%)	33	29	50	49	49	50	38	39
ΔACO	39	49	63	55	19	27	22	17
ΔLS	17	23	107	78	22	28	28	26
Instance	9	10	11	12	13	14	15	16
DTF_{best}	0	0	0	0	0	0	0	0
SCP_{best}	0	0	143	0	5	0	0	0
DTF_{avg}	109	0	0	20	2	0	0	0
SCP_{avg}	202	4	774	538	360	41	29	101
σ_{DTF}	294	0	6	56	9	0	0	0
σ_{SCP}	492	18	247	605	167	56	104	72
$P(DTF = 0)$	0.85	1.0	0.99	0.86	0.94	1.0	1.0	1.0
LS(%)	30	33	50	43	50	47	40	42
ΔACO	42	35	67	54	36	16	21	18
ΔLS	21	14	104	87	33	18	31	28

Table 2 also shows the influence of the local improvement method, i.e. the ejection chain algorithm. The table shows the average percentage of solution improvements found by local search and the average amount by which the solution was improved by ACO and LS respectively. Since local search is only run after an improved solution has been found by ACO, 50% is the maximum for this parameter. It can be seen that local search contributes significantly to solution quality, improving solutions most of the time, even in the difficult cases (1, 2, 9, 10). Also in most cases the amount by which solutions are improved is higher for local search, except in the difficult cases. It should be noted, however, that even without local search high quality solutions are found by ACO alone, though on average the solution quality is significantly lower.

In Table 3 the results of our algorithm are compared with results obtained with other algorithms (cf. Lewis 2010; Chiarandini et al. 2008; Cambazard et al. 2010). The numbers are taken from the results of ITC2007. The algorithms were tested under the same conditions, i.e. run on the same hardware and the same time limit, thus the results can be directly

Table 3 Comparison of the results obtained with our serial ACO algorithm with results obtained with other algorithms submitted to ITC2007. Listed are best and median solutions (DTF/SCP) for 10 runs. Algorithms other than serial ACO are local search based metaheuristic algorithms (cf. Lewis 2010; Chiarandini et al. 2008; Cambazard et al. 2010)

Inst.	Cambazard et al.		Chiarandini et al.		Lewis		Serial ACO	
	Best	Med	Best	Med	Best	Med	Best	Med
1	0/571	0/877	0/1482	0/1696	0/1294	17/1492	0/15	578/1376
2	0/993	0/1523	0/1635	0/1896	0/1599	46/1826	0/0	23 /708
3	0/164	0/236	0/288	0/374	0/278	0/457	0/391	0/709
4	0/310	0/372	0/385	0/463	0/388	0/589	0/239	0/724
5	0/5	0/7	0/559	0/681	0/22	0/193	0/34	0/114
6	0/0	0/0	0/851	0/985	0/369	0/696	0/87	0/288
7	0/6	0/8	0/10	0/403	0/74	0/421	0/0	0/0
8	0/0	0/0	0/0	0/1	0/0	0/206	0/4	0/77
9	0/1560	0/1823	0/1947	0/2081	0/1582	80/2312	0/0	779/1457
10	0/1650	0/1737	0/1741	0/2288	0/2380	126/2262	0/0	0/0
11	0/178	0/286	0/240	0/365	0/344	0/541	0/547	0/891
12	0/146	0/349	0/475	0/593	0/486	0/741	0/32	44/1455
13	0/0	0/160	0/675	0/926	0/365	0/631	0/166	0/404
14	0/1	0/2	0/864	0/958	0/222	0/660	0/0	0/52
15	0/0	0/0	0/0	0/320	0/266	0/344	0/0	0/1
16	0/2	0/11	0/1	0/6	0/99	0/194	0/41	0/129

compared. They cannot be directly compared to our results, which are on average significantly better. The most likely cause for this are platform differences (i.e. different operating system, Java VM, and CPU) which are not completely equalized by the applied benchmark tool. It can be seen that our algorithm often produces the best solution, especially in the cases of the instances 1, 2, 9 and 10. The average solution quality or our serial algorithm, however, is slightly worse than that of competing algorithms, except the algorithm by Lewis (2010) which was not part of the competition, and it fails to find a feasible solution in the given time slightly more often than the competing algorithms.

Table 4 lists the results achieved using the parallelized algorithm. It can be seen that the results with four or eight threads are significantly better than the results with just two threads, while the results with 8 threads are only slightly better than those with 4 threads, however, the solutions are often found in less time. For instance 10, for example, all found solutions were optimal and computed in only 18 seconds, on average.

The number of iterations shows that the algorithm scales fairly well. For the instances where it does not terminate prematurely the 8 threads achieve nearly twice as many iterations as the 4 threads. It should be noted, however, that once a feasible solution is found, one thread is reserved for the SA.

The last column of Table 4 shows how much the SA contributed to the final solution. This contribution is either very high or very low, with not much in between. To find out possible reasons for this result, we looked at the solution backbone, i.e. the elements of the solution which are constant for every optimal solution, of all optimal solutions we found. Table 5 shows how many of the optimal solutions were distinct and the percentage of elements which are constant over all solutions. This, of course, is only an upper bound of the size of the true backbone, since we do not know all optimal solutions. However, these results

Table 4 Results for 100 runs of the ITC2007 instances for the parallel algorithm. The first two columns list instance and number of threads, the following five columns list range and quartiles of the solution scores (DTF/SCP), then follows the median time taken (in seconds), then the median number of iterations and the last column shows the fraction of solutions found by local search, in percent

Inst.	Thr.	best	Q1	Q2 (med)	Q3	worst	time	iter	LS (%)
1	2	0/0	0/15	285/1102	631/1288	832/1468	377	3605	0
	4	0/0	0/0	0/0	0/5	646/1412	233	7266	0
	8	0/0	0/0	0/0	0/0	0/399	136	6542	0
2	2	0/0	0/0	0/12	690/1368	1214/1446	377	3453	7.14
	4	0/0	0/0	0/0	0/0	728/1343	107	3042	0
	8	0/0	0/0	0/0	0/0	0/0	73	3325	0
3	2	0/257	0/336	0/389	0/453	97/444	377	5769	98.61
	4	0/0	0/87	0/236	0/357	0/503	299	14287	98.36
	8	0/0	0/37	0/79	0/178	0/523	336	26025	97.2
4	2	0/12	0/203	0/454	0/516	0/821	377	5724	98.28
	4	0/0	0/27	0/51	0/138	0/583	299	14221	95.04
	8	0/2	0/15	0/37	0/65	0/564	336	25822	93.86
5	2	0/6	0/47	0/67	0/90	0/277	377	4795	0
	4	0/2	0/26	0/42	0/62	0/155	299	12187	0
	8	0/0	0/17	0/30	0/58	0/175	336	20297	0
6	2	0/1	0/88	0/157	0/246	169/851	377	4675	52
	4	0/0	0/25	0/85	0/153	0/462	299	11668	0
	8	0/0	0/13	0/54	0/138	0/299	336	19752	0
7	2	0/0	0/0	0/0	0/0	0/0	59	1472	10
	4	0/0	0/0	0/0	0/0	0/0	26	2102	7.14
	8	0/0	0/0	0/0	0/0	0/20	22	2806	6.67
8	2	0/0	0/0	0/0	0/41	0/161	270	6874	38.71
	4	0/0	0/0	0/0	0/1	0/130	101	8207	23.81
	8	0/0	0/0	0/0	0/0	0/133	70	8659	28.57
9	2	0/0	0/0	0/0	0/1	1239/1646	294	2974	0
	4	0/0	0/0	0/0	0/0	0/0	85	2635	0
	8	0/0	0/0	0/0	0/0	0/0	57	2729	0
10	2	0/0	0/0	0/0	0/0	0/130	87	795	0
	4	0/0	0/0	0/0	0/0	0/0	32	928	0
	8	0/0	0/0	0/0	0/0	0/0	18	872	0
11	2	0/80	0/374	0/424	0/486	75/430	377	5883	98.33
	4	0/0	0/208	0/331	0/415	0/553	299	14539	98.57
	8	0/0	0/166	0/248	0/389	88/355	336	26122	98

Table 4 (*Continued*)

Inst.	Thr.	best	Q1	Q2 (med)	Q3	worst	time	iter	LS (%)
12	2	0/0	0/0	0/27	0/569	488/2337	377	5693	89.58
	4	0/0	0/0	0/0	0/4	47/638	195	9377	82.61
	8	0/0	0/0	0/0	0/0	0/15	145	10670	83.13
13	2	0/8	0/220	0/255	0/301	59/848	377	4551	68.18
	4	0/0	0/143	0/204	0/255	19/471	299	11106	52.08
	8	0/0	0/95	0/175	0/223	0/310	336	18832	0
14	2	0/0	0/3	0/7	0/50	0/159	377	4804	0
	4	0/0	0/0	0/1	0/40	0/191	299	11806	0
	8	0/0	0/0	0/0	0/39	0/156	175	10381	0
15	2	0/0	0/0	0/0	0/0	0/378	59	1462	30.77
	4	0/0	0/0	0/0	0/0	0/388	26	2103	20
	8	0/0	0/0	0/0	0/0	0/7	36	4424	14.29
16	2	0/0	0/0	0/12	0/84	0/214	377	9653	67.5
	4	0/0	0/0	0/0	0/36	0/123	278	22949	60.87
	8	0/0	0/0	0/0	0/43	0/136	312	38765	53.85

Table 5 Solution backbone estimation. The table shows for each of the ITC2007 early and late instances the total number of optimal solutions found, how many of those are distinct and the fraction of elements of the solution which are identical in all solutions, expressed in percent

Instance	1	2	3	4	5	6	7	8
Solutions	249	386	16	40	29	65	556	307
Distinct	82	14	16	40	29	65	556	306
Backbone est.	94.5	97.0	61.5	74.5	54.25	69.25	79.0	82.5
Instance	9	10	11	12	13	14	15	16
Solutions	451	495	20	227	32	232	500	178
Distinct	15	10	19	227	32	232	500	178
Backbone est.	96.75	98.25	73.5	79.5	79.5	70.25	80.5	76.0

correlate well with experiences from running the algorithm. It can be seen, for example, that for instance 10, which our algorithm solves best, only 10 out of 495 optimal solutions are distinct and these have almost all (393 of 400 or 98.25%) of events in common. On the other hand, instances which our algorithm does not solve so well (i.e. those where it ran for the full time for most runs) have a smaller backbone, and these are also the ones where SA contributed most to the solution. That is not to say that a large solution backbone is a necessary condition for our algorithm to perform well, only that if there is a large backbone the algorithm will likely perform well.

It should be noted that algorithms may be biased towards one kind of solution, making it appear as if there were a backbone. In our ACO algorithm, however, we do not use any a priori heuristic information. The algorithm starts with a number of completely random so-

lutions. In each of the iterations each ant constructs a completely new solution from scratch guided only by the pheromone information, which is very faint in the early stages of the algorithm, resulting in a lot of randomness in the solution construction. It is thus very unlikely that the algorithm is biased towards a certain kind of solution.

Lewis (2010) also analyzed which instances are well suited for SA. His results also correlate well with ours. The instances which we identified to most likely have a large backbone (i.e. instances 1, 2, 9 and 10) are very full, i.e. most available slots must be filled. He found that for instance 10 his algorithm had the least chance of making a successful move while these moves also resulted in the lowest reduction in the SCP. The other instances with we found to have a large backbone are close by.

5 Conclusions

From our point of view the key feature of the presented algorithm is the use of two distinct but relatively compact pheromone matrices in combination with an effective procedure to exploit their information in the heuristic solution construction. The algorithm is able to produce high quality solutions even without the local improvement method, but better results could be achieved when including it.

The algorithm ranked 4th among all submitted algorithms, which demonstrates that the presented algorithm falls within the leading algorithms for this task. For 11 out of 24 instances our algorithm found the best solution of all 5 finalists, including ties. On the other hand, our algorithm also showed the largest variation in solution quality, for several instances it produced both the best and the worst solution.

Nevertheless, the large majority of the competition instances can with high probability be solved to optimality within a couple of minutes (i.e. the time limit for the ITC2007 competition). Parallelizing the algorithm further improves its performance, the parallel implementation solves these instances to optimality most of the time.

A very interesting aspect is that the algorithm performs especially well on instances where the algorithms based on local search (cf. Lewis 2010; Chiarandini et al. 2008; Cambazard et al. 2010) have the most difficulty with, see also Table 3. Our analysis suggests that the optimal solutions for these instances have a large backbone, i.e. these problems are highly constrained and for most events there is only one place where they can be placed without violating constraints. One reason why our algorithm performs so well in these situations may be that the penalty term introduced in the pheromone update step prevents most pheromone deposition, and thus convergence, unless the event is placed right. If it is placed right, however, this placement is greatly encouraged. This can also explain why it takes longer for the algorithm to achieve a feasible solution for these instances.

On the other hand, the instances which our algorithm solves the least well are the ones where local search is most successful. Thus a combination of both methods, as has been demonstrated in the parallel version of the algorithm, seems to be beneficial.

References

Abramson, D., Krishnamoorthy, M., & Dang, H. (1999). Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research*, *16*, 1–22.

Burke, E. K., Bykov, Y., Newall, J., & Petrovic, S. (2003). A time-predefined approach to course timetabling. *Yugoslav Journal of Operational Research*, *13*(2), 139–151.

Burke, E. K., & Petrovic, S. (2002). Recent research directions in automated timetabling. *European Journal of Operational Research*, *140*(2), 266–280.

Cambazard, H., Hebrard, E., O'Sullivan, B., & Papadopoulos, A. (2010). Local search and constraint programming for the post enrolment-based course timetabling problem. *Annals of Operations Research*, 1–25.

Chiarandini, M., Birattari, M., Socha, K., & Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, *9*(5), 403–432.

Chiarandini, M., Fawcett, C., & Hoos, H. H. (2008). A modular multiphase heuristic solver for post enrolment course timetabling. In *Proceedings of the 7th international conference on the practice and theory of automated timetabling (PATAT 2008)*.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge: MIT Press.

Gaspero, L. D., & Schaerf, A. (2003). Multi-neighbourhood local search with application to course timetabling. In *Lecture notes in computer science: Vol. 2740. Proc. of the 4th int. conf. on the practice and theory of automated timetabling (PATAT-2002)* (pp. 262–275). Berlin: Springer.

Glover, F. (1992). New ejection chain and alternating path methods for traveling salesman problems. In *Computer science and operations research: new developments in their interfaces* (pp. 449–509).

ITC2007 (2007). http://www.cs.qub.ac.uk/itc2007/ Second international timetabling competition.

Kostuch, P. (2005). The university course timetabling problem with a three-phase approach. In E. Burke, & M. Trick (Eds.), *Lecture notes in computer science: Vol. 3616. Practice and theory of automated timetabling V* (pp. 109–125). Berlin: Springer.

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR-Spektrum*, *30*, 167–190.

Lewis, R. (2010). A time-dependent metaheuristic algorithm for post enrolment-based course timetabling. *Annals of Operations Research*, 1–17.

Mayer, A., Nothegger, C., Chwatal, A., & Raidl, G. (2008). Solving the post enrolment course timetabling problem by ant colony optimization. In *Proceedings of the 7th international conference on the practice and theory of automated timetabling (PATAT 2008)*.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., Gaspero, L. D., Qu, R., & Burke, E. K. (2010). Setting the research agenda in automated timetabling: the second international timetabling competition. *INFORMS Journal on Computing*, *22*, 120–130.

MirHassani, S., & Habibi, F. (2011). Solution approaches to the course timetabling problem. *Artificial Intelligence Review*, 1–17.

Rossi-Doria, O., & Paechter, B. (2003). An hyperheuristic approach to course timetabling problem using an evolutionary algorithm. Technical Report CC-00970503, Napier University, Edinburgh, Scotland.

Rossi-Doria, O., & Paechter, B. (2004). A memetic algorithm for university course timetabling. In *Combinatorial optimisation 2004, Book of abstracts* (p. 56).

Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., & Stützle, T. (2003). A comparison of the performance of different metaheuristics on the timetabling problem. In E. Burke, & P. De Causmaecker (Eds.), *Lecture notes in computer science: Vol. 2740. Practice and theory of automated timetabling IV* (pp. 329–351). Berlin: Springer.

Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review*, *13*(2), 87–127.

Socha, K., Knowles, J., & Sampels, M. (2002). A Max-Min ant system for the university timetabling problem. In M. Dorigo, G. Di Caro, & M. Sampels (Eds.), *Lecture notes in computer science: Vol. 2463. Proceedings of the 3rd international workshop on ant algorithms, ANTS 2002* (pp. 1–13). Berlin: Springer.

Socha, K., Sampels, M., & Manfrin, M. (2003). Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In J. Gottlieb, & G. Raidl (Eds.), *Lecture notes in computer science: Vol. 2611. Proceedings of EvoCOP 2003—3rd European workshop on evolutionary computation in combinatorial optimization*. Berlin: Springer.

TTComp2002 (2002). http://www.idsia.ch/Files/ttcomp2002/ First international timetabling competition (2002).