

MAX-MIN Ant System
for
International Timetabling Competition

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>

March 31, 2003

1 Notation

Here is the notation that is used throughout this document:

- T - a set of timeslots; $T = \{0, \dots, |T| - 1\}$,
- $|T|$ - number of timeslots in the set T ; please note¹ that $|T| \geq 45$,
- t - a timeslot from the set T ; $t \in T, t < |T|$,
- R - a set of rooms; $R = \{0, \dots, |R| - 1\}$,
- $|R|$ - a number of rooms in the set R (provided in the input file for each instance),
- r - a room from the set R ; $r \in R, r < |R|$,
- P - a set of places – a *place* is a unique timeslot/room combination; $P = \{0, \dots, |P| - 1\}$,
- $|P|$ - a number of rooms in the set P ; $|P| = |T| \cdot |R|$,
- p - a room from the set P ; $p \in P, p < |P|, p = t \cdot |R| + r$,
- E - a set of events; $E = \{0, \dots, |E| - 1\}$,
- $|E|$ - a number of events in the set E (provided in the input file for each instance),
- e - an event from the set E ; $e \in E, e < |E|$,
- S - a set of students; $S = 0, \dots, |S| - 1$,
- $|S|$ - a number of students in the set S (provided in the input file for each instance),
- s - a student from the set S ; $s \in S, s < |S|$,
- F - a set of features; $F = \{0, \dots, |F| - 1\}$,
- $|F|$ - a number of features in the set F (provided in the input file for each instance),
- f - a feature from the set F ; $f \in F, f < |F|$,
- τ - the pheromone matrix, $\tau : E \times T \times R \rightarrow \mathbf{R}^+, \tau : E \times P \rightarrow \mathbf{R}^+$,
- τ_{ep} - the pheromone value between the event e and place p ,
- τ_{etr} - the pheromone value between the event e and place represented by the timeslot t and the room r ,

¹Our timetable may actually have more than the 45 timeslots, contrary to what has been stated in the problem definition – see Sec. 2.2 for explanation.

- ρ - pheromone evaporation rate, $\rho \in [0, 1]$,
- γ - exploration rate, $\gamma \in [0, 1]$,
- C - complete assignment of events into places, $C : E \rightarrow P$,
- $q(C)$ - fitness of the solution represented by the assignment C .

2 Algorithm Description

The algorithm that we developed for submitting for International Timetabling Competition is based on Ant Colony Optimization (ACO). 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 [3], quadratic assignment problem [5], scheduling problems [2], and others. We implemented the *MAX-MIN* Ant System [4] – a version of ACO.

Before describing the actual algorithm, there is number of issues that should be clearly defined:

- Preprocessing Problem Data - how the problem data is preprocessed before the algorithm actually tackles the problem, and
- Representation - how the problem data is represented.

2.1 Preprocessing Problem Data

Before the problem is tackled by the algorithm, there is number of steps that are done in the preprocessing phase. Apart from the obvious actions such as reading the problem file and command line parameters, the following actions are performed:

- The number of students per each event is calculated and stored.
- A matrix indicating student clashes between events is created.
- For each event a list of possible rooms is created, (this is done by analyzing room sizes, features provided by rooms, and also number of students attending each event, and features required by each event).
- For each room it is established how many events may be placed in it (based on lists of possible rooms for each event).
- Based on dependencies between possible rooms for events and number of events that may go into a given room, the lists of possible rooms for each event are further restricted, if possible:

If there are exactly 40 events² that may *only* be placed into room r , this means that all *other* events *must* be placed into other rooms even though they theoretically may fit into room r .

If the optimal timetable has to use fully all the 40 optimal timeslots³, and if there are exactly 40 events that may be placed into room r , they *must* be placed into this room and not any other room that they may perhaps also fit into.

- A list of rooms sorted (ascending) based on the number of events that may go into them is created.
- A list of events is created, sorted (ascending) based on the number of rooms that they may go into, with ties broken by number of students that events have in common with all other events (descending)⁴.

2.2 Representation

The timetable is represented in the form of a integer matrix tt with $|T|$ rows and $|R|$ columns. Each position in the matrix – or *place* – is then described by a timeslot-room pair (t, r) – timeslot t and room r . The value of element $e = tt[t][r]$ is the event that has been placed in the timeslot t , room r . The value of $e = -1$ at any position in the matrix indicates that there is no event placed at that position. Since place $p = t \cdot |R| + r$, each position in the matrix may be also described in terms of p :

$$\begin{cases} t = p \div |R| \\ r = p \bmod |R| \end{cases} \quad (1)$$

It is important to note that such a representation does not allow to encode all possible assignments of events into places. In particular, it does not allow to encode any assignment such that any two (or more) events share the same place. However, such assignment is by definition not feasible, and hence should be anyway avoided. The representation chosen is able to encode any feasible assignment, though.

Further, we have decided not to allow any assignment that could cause the timetable to be infeasible. This means that an event may be placed in the timetable only in such a way that it does not violate *any* hard constraints. Hence, not only we do not allow to put two events in the same room in the same timeslot, but also an event may only be placed in a room that satisfies all the requirements in terms of size and required features. Also an event may be placed in a given timeslot only if there is not already other event in that timeslot that has any students in common with the newly placed event.

²Note that optimal timetable *must* use *not more* than 40 timeslots.

³It is the case if $\frac{|E|}{|R|} = 40$

⁴This way of sorting the events is based on the experiments of different event sorting done by Olivia Rossi-Doria from Napier University. We have also experimented with other ways of sorting the events, but this one appeared to be the best.

As number of possible infeasible assignments exceeds by far the number of feasible ones for the problem tackled, it is not trivial to find a feasible assignment. In order to make it easier, we allowed the timetable to actually use more than 45 timeslots. If in fact more than 45 timeslots are used, the timetable may not be of course considered feasible as it does not fulfil the requirements specified in problem definition. The timetable becomes feasible only when it uses at most 45 timeslots.

2.2.1 Fitness Function

Due to the specific way of representing the solution, the fitness function had to be defined accordingly. The calculation of the fitness of the solution depends whether the solution is feasible or not.

If the solution is feasible, the fitness of the solution is expressed by the number of soft constraint violations. The smaller the number, the better the solution. However, for an infeasible solution it does not make sense to calculate number of soft constraint violations. Due to the specific representation chosen, also calculating number of hard constraint violations is not possible, as none hard constraints are violated. Hence, for infeasible solutions the fitness is calculated based on number of timeslots that are used by the timetable over the allowed 45. This number is then multiplied by a large constant (10 000), so that any infeasible solution is much worse than any feasible solution.

Such approach has an advantage that calculating the fitness function for an infeasible solution is very fast. It is much easier to establish how many timeslots are being used, than it would be to establish the exact number of hard constraint violations, if an infeasible timetable was fit into 45 timeslots.

2.3 The Algorithm

The basic mode of operation of the *MAX-MIN* Ant System is as follows. At each iteration of the algorithm, each of the ants constructs a complete assignment C of events into timeslots and rooms (or places). Following a pre-ordered list of events (see Sec. 2.1), 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 τ . Alg. 1 presents the algorithm operation in greater detail.

Some problem specific knowledge (heuristic information) is also used by the algorithm. The place for an event (i.e. the timeslot and 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 one⁵, and the event is placed in one of the rooms of this additional timeslot. This of course results in an infeasible solution as number

⁵Initially $|T| = 45$

Algorithm 1 *MAX-MIN* Ant System

```
while time limit not reached do
  for  $a = 0$  to  $m - 1$  do
    {construction process of ant  $a$ }
     $C_0 \leftarrow \emptyset$ 
    for  $e = 0$  to  $|E| - 1$  do
      choose place  $p$  randomly from set  $P'$  places suitable for event  $e$ , according to probabilities  $prob_{ep}$  for event  $e$  and place  $p$ 
       $C_e \leftarrow C_{e-1} \cup \{ep\}$ 
    end for
     $C \leftarrow$  solution after applying local search algorithm to  $C_{|E|-1}$ 
     $C_{iteration\ best} \leftarrow$  best of  $C$  and  $C_{iteration\ best}$ 
  end for
   $C_{global\ best} \leftarrow$  best of  $C_{iteration\ best}$  and  $C_{global\ best}$ 
  global best or local best pheromone update (according to  $\gamma$ ) for  $\tau$  using  $C_{global\ best}$ ,  $\tau_{min}$ , and  $\tau_{max}$ 
end while
```

of timeslots used from now on exceeds 45⁶. This also means that pheromone matrix has to be extended as well.

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. 3. Finally the best solution of each iteration is compared to the global best solution found so far. Only the better of the two is kept as the new global best.

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 *MAX-MIN* Ant System introduces upper and lower limits on the pheromone values – τ_{max} and τ_{min} respectively [4] – 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 table τ is updated either by the best assignment of a given iteration (i.e. *local best*), or by the global best assignment. We probabilistically choose which one to use. The local best update is chosen with probability $prob$ proportional to its quality compared to the quality of the global best solution, and also the exploration rate γ :

$$prob = \gamma \cdot \left[\frac{q(C_{global\ best})}{q(C_{local\ best})} \right]^\alpha \quad (2)$$

where α is an additional scaling parameter. The pheromone update rule is as follows (for the particular case of assigning event e into place p):

$$\tau_{ep} \leftarrow \begin{cases} (1 - \rho) \cdot (\tau_{ep} + \Delta\tau_{ep}) & \text{if } ep \text{ is in } C_{best} \\ (1 - \rho) \cdot \tau_{ep} & \text{otherwise,} \end{cases} \quad (3)$$

⁶Only for this particular ant, and only in this iteration.

where C_{best} is either local or global best, and $\Delta\tau_{ep}$ is the pheromone update value, which is calculated based on the level of usage of rooms in given timeslot:

$$\Delta\tau_{ep} = \tau_{max} \cdot \frac{n}{|R|} \quad (4)$$

where $1 \leq n \leq |R|$ is the number of rooms used in timeslot $t = p / |R|$. Pheromone update is completed using the following:

$$\tau_{ep} \leftarrow \begin{cases} \tau_{min} & \text{if } \tau_{ep} < \tau_{min}, \\ \tau_{max} & \text{if } \tau_{ep} > \tau_{max}, \\ \tau_{ep} & \text{otherwise.} \end{cases} \quad (5)$$

3 Local Search

The LS used here by the *MMAS* solving the UCTP consists of two major modules. First module tries to improve an infeasible solution, so that it becomes feasible. Since its main purpose is to produce a solution that does not contain any hard constraints violations, 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 the **HardLS**). This second module tries to increase the quality of the solution by reducing number of soft constraint violations ($\#scv$), and hence is called the **SoftLS**.

3.1 Hard Constraints

As described in section Sec. 2.2, the possible infeasibility of the solution generated by an ant, may only lay in the fact that more timeslots 45 are actually used by the timetable. The **HardLS** tries to reduce the number of timeslots used by:

- moving **single** events from their places to other *suitable* places,
- swapping **pairs** of events (so they still endup in *suitable* places).

By a *suitable* place for an event we understand a place such that placing that event there will not violate any hard constraints. Note that a suitable place may still be in a timeslot $t > 45$.

The **HardLS** starts improving the timetable at a randomly selected place, and then loops through all the places trying to reduce number of timeslots used. It exits, when a feasible solution has been found, or when in last $|P|$ iterations no improvement has been made. The **HardLS** is fairly fast, since the only measure it uses to judge the improvement of the solution is the number of timeslots used.

Table 1: Parameters used by the algorithm.

Parameter	Value	Description
m	3	number of ants used
ρ	0.15	pheromone evaporation rate
τ_{min}	2.5E-05	minimal pheromone level
τ_{max}	6.67	maximal pheromone level
γ	0.65	exploration rate
α	1.0	scaling parameter

3.2 Soft Constraints

The **SoftLS** also rearranges the events. It however aims at increasing the quality of the already feasible solution, without introducing infeasibility. In case of the **SoftLS**, an event may only be placed in timeslot $t < 45$. This part of the LS routine is much slower than **HardLS**, as the improvement may only be measured through evaluating the number of soft constraints violations. Even though a fast delta evaluation is used, it is a computationally expensive operation.

The **SoftLS** performs two basic types of operations just like the **HardLS** module. It however accepts only moves that do not make the quality of the solution worse, and that do not introduce any infeasibility. In the initial stage of the search (for first 100 iterations of the algorithm) only the first operation is performed (moving events). Later both are executed in each iteration, until none is making any improvement.

The **SoftLS** is run only if the solution is already feasible. It is much slower than the **HardLS** module, as the evaluation of the fitness function is slower and more complicated.

4 Parameters

The algorithm accepts number of command line parameters that are used to tune its performance. Table 1 presents the parameters used together with short description and the values used for obtaining the results submitted to the Iterational Timetabling Competition.

References

- [1] M. Dorigo, V. Maniezzo, and A. Coloni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26:29–41, 1996.
- [2] D. Merkle, M. Middendorf, and H. Schneck. Ant colony optimization for resource-constrained project scheduling. In *Proceedings of the Genetic and*

Evolutionary Computation Conference (GECCO 2000), pages 893–900. Morgan Kaufmann Publishers, 2000.

- [3] T. Stützle and M. Dorigo. Aco algorithms for the traveling salesman problem. In M. Makela, K. Miettinen, P. Neittaanmäki, and J. Périaux, editors, *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, June 1999.
- [4] T. Stützle and H. H. Hoos. *MAX-MIN* Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.
- [5] Thomas Stützle and Marco Dorigo. *ACO Algorithms for the Quadratic Assignment Problem*. McGraw-Hill, 1999.