Metaheuristics for the Timetabling Problem

Krzysztof SOCHA

Technical Report No.

TR/IRIDIA/2003-29
September 2003

DEA thesis

Metaheuristics
for the
Timetabling Problem

by

Krzysztof Socha
Iridia, Université Libre de Bruxelles
50, av. F.Roosevelt, CP 194/6, 1050 - Brussels, Belgium
e-mail: krzysztof.socha@ulb.ac.be

Supervised by:

Marco Dorigo
Ph.D., Maitre de Recherches du FNRS
IRIDIA, Université Libre de Bruxelles

50, av. F.Roosevelt, CP 194/6, 1050 - Brussels, Belgium
A thesis submited in partial fulfilment of the requirements
of the ULB, Faculté des Sciences Appliques
for the Diplome d'Etudies Approfondies (DEA).

Abstract

Timetabling is an interesting combinatorial optimization problem. Its definition comes from real world situations, where timetables have to be created – such as in schools, universities, hospitals, etc. There exist several different variants of this problem.

We shortly present the general idea of the timetabling problem and then a few distinctive variants of this problem. We identify the general characteristics of these variants and highlight the differences and similarities. We show that complexity of the timetabling problems in general is NP-complete due to several different factors. Eventually we choose one of the presented variants of a timetabling problem, and perform a bit more in-depth analysis of it. We clearly define the variant that we would tackle and present the test instances that we would use for evaluating the algorithms for solving this problem.

We then present a set of five metaheuristics that may be used for tackling the timetabling problem. They include: simulated annealing, iterated local search, tabu search, evolutionary algorithm, and ant colony optimization. We briefly present the key ideas, their origins and applications. For each of them we provide a short algorithmic overview of their operation.

We later focus on one of the metaheuristics – Ant Colony Optimization – to see how it may be applied to the timetabling problems. As it is one of the first attempts to use Ant Colony Optimization metaheuristic for these kind of problems, we discuss several design considerations. We implement the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the University Course Timetabling Problem. Our algorithm makes use of a local search procedure, but we show that our algorithm is significantly better than the local search alone by comparing it to a random restart local search algorithm. Using another ant algorithm – the Ant Colony System – developed by some other researchers[1] we are able to show how different ant algorithms perform on the same timetabling problem. Also, based on the results of other people developing other metaheuristics for the timetabling problems[1] we are able to show how our $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System compares to them.

After the initial experiments with the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System we gather enough experience to propose a second, improved version of this algorithm – $\mathcal{MM}$AS2. We present the changes introduced with regard to the initial algorithm and discuss their importance. We present some considerations of choosing proper parameters for this new algorithm, which we support by experimental results. Eventually, we show how much improvement in the performance has been archived with the new algorithm comparing to the previous one. We discuss the results obtained.

[1]See acknowledgment section for details.

During the research we have published several papers. Some of them concerned multiobjective agent systems, and other dealt clearly with the topics covered by this work. These were the papers on timetabling problems and the ant colony optimization metaheuristic. The published papers are listed towards the end of this work. The full versions of the papers are attached as Appendix A.

Contents

Chapter 1

The Timetabling Problem

The Timetabling Problem (TTP) [39, 14, 15] is a combinatorial optimization problem. The task is to assign a set of participants into particular events, and/or the events into places and timeslots. Depending on the variant of the problem there may be different participants, events, places, and timeslots. The TTP is a constraint satisfaction type of problem. It usually defines several hard and soft constraints. The hard constraints determine the *feasibility* of the timetable, and the soft constraints determine further its *quality*. Solving the TTP usually means finding such an assignment, so that no hard constraints are violated (timetable is feasible), and the number of violations of soft constraints is as low as possible (so that the feasible timetable is as good as possible). Alternatively, the aim may also be to state that a feasible solution does not exist to a given problem. The hard constraints usually include:

- no participant can be in two different places at the same time;
- only one event can take place in one place at given timeslot;
- assigned place has to fulfil certain requirements of the event (size, features, etc).

The soft constraints are very much problem specific. They depend on the additional qualities (features) of the places and events, additional requirements for the participants schedule, personal preferences, organizational issues, physical / geographical location of the places, etc.

There are many purely academic combinatorial optimization problems tackled by a number of researchers (and algorithms) in the world. Those pure academic problems are usually simple to define in a quite abstract way, but difficult to solve. They may not be always clearly related to real life situations. The TTP is another kind of problem. Its definition comes from real life applications. Initially people manually created timetables and only later they started using computers for this purpose. There existed (and still exists) a number of *interactive* approaches to timetabling, where manual actions were coupled with some automated timetabling. Here, we focus on the *automated* timetabling, i.e. the timetabling done fully by algorithms without any human interaction in the process.

1.1 Variants of the Timetabling Problem

Since the timetabling problem has not been defined in an utterly artificial way, but clearly came from real life situations, there are several types and variations of the TTP. They mostly differ by the types and the number of participants, timeslots, and places. Each problem also usually specifies different constraints. Some of the more popular types include:

- High School Timetabling,
- University Timetabling,
- Employee Timetabling.

However, many more exist. The following subsections present the basic characteristics of those more popular variations of the timetabling problem.

1.1.1 High School Timetabling

High School Timetabling usually refers to the problem of constructing timetables (or schedules) for high schools. The typical characteristics of such problem include:

- the participants are divided into teachers and (whole) classes;
- the places are the available classrooms;
- the timeslots are the available periods;
- additional information is provided on the given class *curriculum* (frequency of teachers having courses in the given class)

The example of the hard constraints for this problem may include [12, 45]:

- every teacher and every class must be present in the timetable in a predefined number of hours;
- there may not be more than one teacher in the same class on the same hour;
- no teacher can be in two classes on the same hour;
- there can be no "uncovered hours" (that is, hours when no teacher has been assigned to a class).

Additionally there may be a number of soft constraints defined. They may refer to particular organizational issues (teachers prefer to have a day off, classes should not have too many hours during one day, everyone should have a lunch break, etc.).

1.1.2 University Timetabling

The University Timetabling Problem (UTP) is a problem that is periodically faced by any university in the world. Depending on the definition, it involves scheduling number of courses or exams into rooms and timeslots. Hence, it is sometimes called the Examination Timetabling, or University Course Timetabling. The typical hard constraints for this problem are:

- no student should have two courses/exams at the same time;
- there may be only one course/exam in a given room at a given time;
- the room has to fulfill the requirements of the course/exam (size, features, etc.).

This is probably the most common type of the TTP found in the literature [10, 39, 16, 8]. There is a large number of types of this variant of the problem. These types differ mostly by the number and type of soft constraints. In fact, it seems that each university has a different opinion how a good timetable should look like. Some want to optimize the room use, some want to include student preferences, and yet some have very specific other requirements.

1.1.3 Employee Timetabling

The Employee Timetabling Problem (ETP) is often defined as the problem of assigning a set of employees to tasks and work-shifts [32]. Employees have a set of skills, while the tasks have a set of recommendations. The work-shifts are assumed to be fixed in time. The examples of the ETP may be:

- assignment of nurses to shifts in a hospital,
- assignment of workers to cash registers at a large store,
- assignment of phone operators to shifts and stations in a service-oriented call-center.

The ETP usually involves an institution with a set of tasks to be completed, a set of employees that have a certain skills, a time schedule for completion of the tasks, and availability of the employees. There also may be a number of hard and soft constraints influencing possible assignments of employees to tasks and work-shifts.

The goal is usually to fulfill the constraints defined, or achieve some general objectives, such as minimal tasks completion time, minimization of the number of employees, or equitable division of work.

1.2 Common Framework

There have been numerous attempts to develop a common framework for tackling all types of timetable problems. However, none seems to be widely adopted by the people involved in the timetabling research. The efforts in this area focus on developing a common specification of timetable problems. There have been some generic structures and languages developed. Examples include STTL [25], UniLang [37], and others [9, 11].

The major difficulty however with such a generalized approach to timetabling is that the more general is the description of the problem tackled, the less efficient becomes the algorithm to solve it. Another words, in order to have an efficient algorithm for a given type of the timetabling problem, it has to be as narrowly defined as possible.

1.3 Complexity of Timetabling Problems

It has been shown that timetabling problems are difficult combinatorial optimization problems. The timetabling is sometimes compared to a more pure academic problem – the *graph coloring* problem [35]. Usually however, the timetabling problem appears to be much harder to solve than just graph coloring, and in order to present it as a graph coloring problem requires some simplifications [15, 8]. In particular it has been shown that they may be transformed (through reduction) to graph coloring problems [14]. In turn, it has been shown previously that graph coloring is a NP-complete type of problem [24].

Also, [14] shows that usually the timetabling problems are in fact NP-complete due to number of factors. It shows that there may be a set of subproblems defined on the basis of a timetabling problem, each of which is NP-complete on its own. It is hence quite clear that the timetabling problems are some of the most difficult problems to solve.

1.4 Problem Tackled: University Course Timetabling

We present in this section the actual variant of the problem that we tackled in our research – the University Course Timetabling Problem (UCTP). The problem involves scheduling number of courses into set of timeslots and rooms. We present here a clear definition of the version of the problem tackled, and also the problem instances that were used for testing and evaluating the algorithms.

1.4.1 UCTP Definition

The University Course Timetabling Problem [38][5, 6] consists of a set of events $E = \{e_1, \ldots, e_{|E|}\}$ to be scheduled in a set of timeslots $T = \{t_1, \ldots, t_{|T|}\}$, and a set of rooms $R = \{r_1, \ldots, r_{|R|}\}$ in which events can take place. Two additional

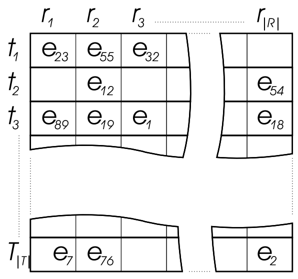

Figure 1.1: Timetable of $|T|$ timeslots and $|R|$ rooms ($|P| = |T| \cdot |R|$ places). Some events from the set E have already been placed.

sets are 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. The solution to the UCTP is a mapping of events into particular timeslots and rooms. Fig. 1.1 shows an example of a timetable. The problem may be defined in a more formal way. Let:

a indicate students attendance:

$$a_{(s_i,e_j)} = \begin{cases} 1, & \text{if student } s_i \text{ attends event } e_j, \\ 0, & \text{otherwise,} \end{cases} \tag{1.1}$$

g indicate which rooms are suitable for which events (i.e. the room provides all the features that the event requires and has the right size):

$$g_{(r_l,e_j)} = \begin{cases} 1, & \text{if room } r_l \text{ is suitable for the event } e_j, \\ 0, & \text{otherwise,} \end{cases} \tag{1.2}$$

p indicate the placement of events:

$$p_{(e_j,t_k,r_l)} = \begin{cases} 1, & \text{if event } e_j \text{ is placed in timeslot } t_k \text{ and room } r_l, \\ 0, & \text{otherwise,} \end{cases} \tag{1.3}$$

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:

- hc_1 : no student attends more than one event at the same time;
- hc_2 : the room is big enough for all the attending students and satisfies all the features required by the event;
- hc_3 : only one event is taking place in each room at a given time.

The hard constraints may be then presented in the from of the following conditions, respectively:

$$hc_1 \quad : \quad \forall_{s\in S}\forall_{t\in T}\sum_{j=0}^{|E|}\sum_{l=0}^{|R|} a_{(s,e_j)}\cdot p_{(e_j,t,r_l)} \leq 1 \tag{1.4}$$

$$hc_2 \quad : \quad \forall_{e\in E}\sum_{k=0}^{|T|}\sum_{l=0}^{|R|} p_{(e,t_k,r_l)}\cdot g_{(r_l,e)} = 1 \tag{1.5}$$

$$hc_3 \quad : \quad \forall_{t\in t}\forall_{r\in R}\sum_{j=0}^{|E|} p_{(e_j,t,r)} \leq 1 \tag{1.6}$$

The infeasible timetables are worthless and are considered equally bad regardless of the actual level of infeasibility. In addition, a feasible candidate timetable is penalized equally for each occurrence of the following soft constraint violations:

- sc_1 : a student has a class in the last slot of the day;
- sc_2 : a student has more than two classes in a row (one penalty for each class above the first two);
- sc_3 : a student has exactly one class during a day.

Also the soft constraints may be defined in more formal way. They are respectively:

$$sc_1 \quad = \quad \sum_{d=0}^{4}\sum_{l=0}^{|R|}\sum_{j=0}^{|E|}\sum_{i=0}^{|S|} p_{(e_j,t_{9d+8},r_l)}\cdot a_{(s_i,e_j)} \tag{1.7}$$

$$sc_2 \quad = \quad \sum_{d=0}^{4}\sum_{i=0}^{|S|}\sum_{k=9d}^{9d+7} h_{()}, \tag{1.8}$$

$$h_{()} \quad = \quad \begin{cases} 1, & \text{if } \sum_{n=0}^{2}\sum_{l=0}^{|R|}\sum_{j=0}^{|E|} p_{(e_j,t_{k+n},r_l)}\cdot a_{(s_i,e_j)} = 3 \\ 0, & \text{otherwise,} \end{cases}$$

where d indicates the day of the week and $h_{()}$ indicates whether the student s_i has three consecutive classes starting with timeslot t_k.

$$sc_3 \quad = \quad \sum_{d=0}^{4}\sum_{i=0}^{|S|} q_{()}, \tag{1.9}$$

$$q_{()} \quad = \quad \begin{cases} 1, & \text{if } \sum_{k=9d}^{9d+9}\sum_{l=0}^{|R|}\sum_{j=0}^{|E|} p_{(e_j,t_k,r_l)}\cdot a_{(s_i,e_j)} = 1 \\ 0, & \text{otherwise,} \end{cases}$$

where $q_{()}$ indicates whether the student s_i has exactly one class during the day d.

Table 1.1: Parameter values for the three UCTP classes.

Class	`small`	`medium`	`large`
Num_events	100	400	400
Num_rooms	5	10	10
Num_features	5	5	10
Approx_features_per_room	3	3	5
Percent_feature_use	70	80	90
Num_students	80	200	400
Max_events_per_student	20	20	20
Max_students_per_event	20	50	100

The goal of solving the problem formulated in such a way is to minimize the number of soft constraint violations:

$$\min \#scv = sc_1 + sc_2 + sc_3 \tag{1.10}$$

1.4.2 UCTP Instances

There were two general types of instances used for evaluation of our algorithms for UCTP:

- *Metaheuristics Network (MN) instances* – a set of instances chosen by the Metaheuristic Network[1] for evaluation of the metaheuristics developed in the course of the Metaheuristics Network project, and
- *competition instances* – a set of instances provided by the organizers of the International Timetabling Competition[2].

Instances of the UCTP of both types were constructed using a generator written by Paechter[3]. The generator makes instances for which a perfect solution exists, that is, a timetable having no hard or soft constraint violations. The generator is called with eight command line parameters that allow various aspects of the instance to be specified, plus a random seed.

Three classes of the MN instances have been chosen, reflecting realistic timetabling problems of varying sizes. These classes are defined by the values of the input parameters to the generator, and different instances of the class can be generated by changing the random seed value. The parameter values defining the classes are given in Tab. 1.1. There were several instances of each class generated.

The second type of instances used for algorithm evaluation, were the instances that have been proposed as a part of the International Timetabling

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

Competition. They were also generated with the same generator, however the parameters used to generate them were not made available. There were 20 instances created.

Chapter 2

Metaheuristics

The timetabling problems being difficult and often encountered in the real world, have been tackled by many algorithms. Also metaheuristics have been used to solve such problems.

In this chapter we present some general information about the metaheuristics that have been used in the past for tackling the timetabling problems. The metaheuristics covered include:

- Simulated Annealing,
- Iterated Local Search,
- Tabu Search,
- Evolutionary Algorithms,
- Ant Colony Optimization.

In this chapter we provide general description of these metaheuristics. Chapter 3 provides a more focused view on one of these metaheuristics (the Ant Colony Optimization) as used for UCTP, as well as results obtained by other metaheuristics as a reference.

2.1 Simulated Annealing

The Simulated Annealing (SA) is considered to be the oldest metaheuristic. It was first proposed by Kirkpatrick *et al.* [26]. It is a Monte Carlo approach to optimization and its origins are in statistical mechanics (Metropolis algorithm).

The term *simulated annealing* derives from the roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure. In simulation, a minima of the cost function corresponds to this ground state of the substance. The simulated annealing process lowers the temperature by slow stages until the system *freezes* and no further changes

Algorithm 1 Simulated Annealing

```
initialize $T$
$s \leftarrow$ initial solution
while termination condition not met do
  $s' \leftarrow$ random solution from neighborhood $N(s)$
  if $f(s') < f(s)$ then
    $s \leftarrow s'$
  else
    $s \leftarrow s'$ with probability $e^{\frac{-(f(s')-f(s))}{T}}$
  end if
  update $T$
end while
```

occur. At each temperature the simulation must proceed long enough for the system to reach a steady state or equilibrium. This is known as *thermalization.* The time required for thermalization is the decorrelation time; correlated microstates are eliminated. The sequence of temperatures and the number of iterations applied to thermalize the system at each temperature comprise an annealing schedule.

To apply simulated annealing, the system is initialized with a particular configuration, and the so-called temperature parameter T is initialized. A new configuration is constructed by imposing a random displacement. If the energy of this new state is lower than that of the previous one, the change is accepted unconditionally and the system is updated. If the energy is greater, the new configuration is accepted probabilistically: a solution s' from the neighborhood $N(s)$ of the solution s is accepted as new current solution depending on $f(s)$, $f(s')$ and T. The probability is generally computed following the Boltzmann distribution $e^{\frac{-(f(s')-f(s))}{T}}$. This is the Metropolis step, the fundamental procedure of simulated annealing. This procedure allows the system to move consistently towards lower energy states, yet still *jump* out of local minima due to the probabilistic acceptance of some upward moves. If the temperature is decreased algorithmically, simulated annealing guarantees an optimal solution.

The temperature T is decreased during the search process, thus at the beginning of the search the probability of accepting uphill moves is high and it gradually decreases, converging to a simple iterative improvement algorithm. Regarding the search process, this means that the algorithm is the result of two combined strategies: random walk and iterative improvement. In the first phase of the search, the bias toward improvements is low and it permits the exploration of the search space; this erratic component is slowly decreased thus leading the search to converge to a (local) optimum. The probability of accepting uphill moves is controlled by two factors: the difference of the objective functions and the temperature. On the one hand, at fixed temperature, the higher the difference $f(s\prime) - f(s)$, the lower the probability to accept a move from s to s'. On the other hand, the higher T, the higher the probability of

Algorithm 2 Iterated Local Search

```
$s \leftarrow$ initial solution
$s^* \leftarrow$ LocalSearch($s$)
while termination condition not met do
  $s' \leftarrow$ Perturbation($s^*$, history)
  $s^{*\prime} \leftarrow$ LocalSearch($s'$)
  if AcceptanceCriteria($s^*$, $s^{*\prime}$, history) then
    $s^* \leftarrow s^{*\prime}$
  end if
end while
```

uphill moves.

The basic mode of operation of a simulated annealing algorithm is shown in Alg. 1. The simulated annealing may be further improved by introducing local search heuristic [31].

2.2 Iterated Local Search

Iterated Local Search (ILS) is a simple idea, but has a long history. Its rediscovery by many authors has lead to many different names for ILS like iterated descent [7], large-step Markov chains [30], iterated Lin- Kernighan [23], chained local optimization [31], and combinations of these [28].

In short, the idea of the iterated local search metaheuristic relies on the multiple runs of a given local search algorithm (that being either a heuristic or an exact algorithm) iteratively. The important difference between the ILS and the RRLS (Random Restart Local Search) is the choice of the subsequent starting points. While in case of the RRLS each starting point is chosen at random, in the ILS the subsequent starting points are chosen based on the previous best solution found. A *perturbation* is applied to the previous best solution that yields a new solution to be improved again with the use of local search procedure. The performance of the ILS algorithm depends very significantly on the perturbation used.

The ILS essentially transforms the search space from a set of all possible solutions S to a set of local optima S^*. The power of the ILS relies on the idea of this reduction of the search space and biased sampling of this reduced search space. Also, often the closer the search progresses to the global optimum, the more local optima are in the proximity. This is for instance true for the Traveling Salesman Problem (TSP).

The general mode of operation of the ILS is the following. A random solution $s \in S$ is chosen. This solution s is improved by the local search procedure to obtain a local optimum $s^* \in S^*$. A perturbation is applied to s^* in order to obtain $s' \in S$. Then the local search routine is used to improve s' and obtain $s^{*\prime} \in S^*$. Some acceptance criteria is than used whether $s^{*\prime}$ should become the s^*. Then another perturbation is performed. Often, some history

Algorithm 3 Tabu Search

```
$k \leftarrow 1$
$s \leftarrow$ initial solution
$s^* \leftarrow s$
while termination condition not met do
    identify neighborhood set $N(s)$
    identify tabu set $T(s,k)$
    identify aspiration set $A(s,k)$
    $s \leftarrow$ best of $N(s,k) = N(s) - T(s,k) + A(s,k)$
    if $f(s) < f(s^*)$ then
        $s^* \leftarrow s$
    end if
    $k \leftarrow k + 1$
end while
```

of perturbations is used in order to further enhance the guidance on the search process. Alg. 2 presents the overview of a typical ILS.

There exist numerous variants of the ILS algorithm. They vary by the acceptance criteria used, the history, and also the neighborhood used. ILS is currently the state-of-the-art metaheuristic for number of combinatorial optimization problems.

2.3 Tabu Search

The Tabu Search (TS) metaheuristic was first proposed by Glover [20, 21]. Similarly to the ILS described earlier, TS also employs the idea of local search. However it does not restart the local search in order to recover from local optimum, but employs a special technique of accepting worse solutions. It allows the search to explore solutions that do not decrease the objective function value only in those cases where these solutions are not forbidden. This is usually obtained by keeping track of the last solutions in term of the action used to transform one solution to the next. When an action is performed it is considered *tabu* (i.e. forbidden) for the next t iterations, where t is the tabu list length. A solution is forbidden if it is obtained by applying a tabu action to the current solution.

Tabu search assumes that a given solution s may be improved by making small changes. Those solutions s' obtained by modifying solution s are called neighbors of s. Hence the notion of the neighborhood of s denoted as $N(s)$. The local search algorithm starts with some initial solution and moves from neighbor to neighbor as long as possible while decreasing the objective function value. The main problem with this strategy is to escape from local minima where the search cannot find any further neighborhood solution that decreases the objective function value. Different strategies have been proposed to solve this problem. Tabu search is one of the most efficient of these strategies.

At each step k the tabu search identifies the neighborhood $N(s)$ of current solution s, current tabu list $T(s,k)$, and current aspiration set $A(s,k)$. The aspiration set identifies the possible solutions that are on the tabu list $T(s,k)$ but should nevertheless be accepted. This may be for instance the case if a solution $s' \in T(s,k)$ is better than current best solution s^*. Alg. 3 presents the overview of the TS operation.

2.4 Evolutionary Algorithms

The term Evolutionary Algorithms (EA) refers to the study of the foundations and applications of certain heuristic techniques based on the principles of natural evolution. In spite of the fact that these techniques can be classified into three main categories, this classification is based on some details and historical development facts rather than on major functioning differences. In fact, their biological basis is essentially the same.

Essentially the evolutionary algorithms may be divided into the following heuristic techniques: *genetic algorithms* [22], *evolution strategy* [36], *evolutionary programming* [19], and *genetic programming* [27].

Originally the evolutionary algorithms attempted to mimic some of the processes taking place in natural evolution. Although the details of biological evolution are not completely understood (even nowadays), there were some points strongly supported by experimental evidence:

- Evolution is a process operating over chromosomes rather than over organisms. The former are organic tools encoding the structure of a living being, i.e, a creature is *built* by decoding a set of chromosomes.
- Natural selection is the mechanism that relates chromosomes with the efficiency of the entity they represent, thus allowing those efficient organisms which are well-adapted to the environment to reproduce more often than those which are not.
- The evolutionary process takes place during the reproduction stage. There exists a large number of reproductive mechanisms in *Nature*. Most common ones are mutation (that causes the chromosomes of offspring to be different to those of the parents) and recombination (that combines the chromosomes of the parents to produce the offspring).

All types of evolutionary algorithms have some qualities in common. They operate on a *population* of individuals. Each individual represents a potential solution to the problem being solved. This solution is obtained by means of a encoding/decoding mechanism. Initially, the population is randomly generated (perhaps with the help of a construction heuristic). Every individual in the population is assigned, by means of a fitness function, a measure of its goodness with respect to the problem under consideration. This value is the quantitative information the algorithm uses to guide the search. An Evolutionary Algorithm (EA) is an iterative and stochastic process that operates on this set of individuals (population). Depending on the type of the EA, many different reproduction

Algorithm 4 Evolutionary Computation

```
generate initial population $P(0)$
$t \leftarrow 0$
while termination condition not met do
  evaluate $P(t)$
  $P'(t) \leftarrow$ Select($P(t)$)
  $P''(t) \leftarrow$ ApplyReproductionOperators($P'(t)$)
  $P(t+1) \leftarrow$ Replace($P(t)$,$P''(t)$)
  $t \leftarrow t+1$
end while
```

operators may be used, but they all are usually some form of mutation or recombination operators. All variants of the EAs also use some type of selection mechanism (which individuals form the population take part in the reproduction process), as well as replacement mechanism (which individuals stay in the population and which are discarded). Alg. 4 presents a general skeleton of an EA.

It can be seen that the algorithm comprises three major stages: selection, reproduction and replacement. During the selection stage, a temporary population is created in which the fittest individuals (those corresponding to the best solutions contained in the population) have a higher number of instances than those less fit (natural selection). The reproductive operators are applied to the individuals in this population yielding a new population. Finally, individuals of the original population are substituted by the new created individuals. This replacement usually tries to keep the best individuals deleting the worst ones. The whole process is repeated until a certain termination criterion is achieved (usually after a given number of iterations or certain computation time).

2.5 Ant Colony Optimization

ACO is a metaheuristic proposed by Dorigo et al. [18]. 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 [41], quadratic assignment problem [40], scheduling problems [33], and others.

ACO algorithms are based on a parameterized probabilistic model – *the pheromone model* – that is used to model the chemical pheromone trails. Artificial ants incrementally construct solutions by adding solution components to a partial solution under consideration. In order to accomplish that, artificial ants perform randomized walks on a completely connected graph $G = (C, L)$, whose vertices are the solution components C and the set L are the connections. This graph is commonly called the *construction graph.* When a constrained combinatorial optimization problem is considered, the problem constraints are built

Algorithm 5 Ant Colony Optimization

```
initialize the pheromone table $\tau$
$s \leftarrow$ random initial solution
while termination condition not met do
   ants construct the set of solutions $S'$
   update and evaporate the pheromone table
   $s \leftarrow$ best of $s' \in S'$ and $s$
end while
```

into the ants' constructive procedure in such a way that in every step of the construction process only feasible solution components can be added to the current partial solution. In most applications, ants are implemented to build feasible solutions, but sometimes it is desirable to also let them build unfeasible solution that later may be *repaired*.

The construction process performed by the ants is influenced by the *pheromone trail parameter*. The value of such a parameter is usually denoted by τ_i. The set of all pheromone trail parameters is denoted by τ and is often referred to as the *pheromone matrix*.

The ants traverse the construction graph making at each node of the graph a probabilistic decision which path to choose. This probabilistic decision is based on two types of information:

- heuristic information,
- pheromone information.

The probability is calculated by each ant using the following equation:

$$p_{(i,j)} = \frac{\tau^{\alpha}_{(i,j)}\eta^{\beta}_{(i,j)}}{\sum_{k=1}^{n}\tau^{\alpha}_{(i,k)}\eta^{\beta}_{(i,k)}}$$

where $\tau_{(i,j)}$ is the pheromone level associated with trail going from node i to node j, and $\eta_{(i,j)}$ is the heuristic information associated with the same trail (i.e. this could be the distance in case of TSP, or some other heuristic measure in case of other combinatorial optimization problems). α and β are scaling coefficients.

Once all the ants have constructed their solutions, some of them (sometimes all of them) return along the same path and update the pheromone trails. Also at each iteration of the algorithm the pheromone trails evaporate a bit. Alg. 5 presents the operation of ACO in greater detail. Some recent results in the literature [41, 29, 43] show that ACO performs especially well when coupled with a local search routine.

Chapter 3

Ant Colony Optimization for the UCTP

Main focus of our work concerned the development and evaluation of ant algorithms that could solve the UCTP. In this section we present some design considerations that we investigated as well as the choices eventually made. Also we present the results obtained by our algorithms, and we try to annalize the properties of the algorithms solving the UCTP to gain their better understanding.

3.1 Task Definition

One of the very first things to be decided on upon tackling any specific problem, is the very definition of a task that we wanted to accomplish. As the development of the ant algorithms was a part of the effort of the Metaheuristic Network project, the task was clearly defined and some modules have been supplied. The supplied modules included in particular:

- *a deterministic matching algorithm* that performed a proper matching of events into rooms for any given timeslot, and
- *a local search* allowing to improve the solutions found by the and algorithm.

Because of these ready-to-use modules provided, the task associated with developing the ant algorithm for UCTP was a bit reduced and restricted. Ants were to assign the the events into timeslots only. The proper assignment of events into the rooms is later done by the matching algorithm, and eventually improved with the local search.

3.1.1 Matching Algorithm

The deterministic matching algorithm based on a network flow algorithm was designed to assign events to particular rooms within each timeslot. This algorithm was provided by the Metaheuristic Network. It deterministically tried to fit a set of events into a set of rooms for a given timeslot. If it was not possible to properly fit all the events, the maximal possible number of events was assigned to rooms, and the rest was simply assigned randomly to any of the (appropriate) rooms.

3.1.2 Local Search

Local search – also provided by the Metaheuristic Network – aimed at improving the solutions found by the ants. The local search used three possible neighborhood structures:

N_1 – neighborhood defined by moving a single event from one timeslot to another;

N_2 – neighborhood defined by swapping two events from different timeslots;

N_3 – neighborhood defined by performing a 3-opt move.

The local search had two distinctive phases of operation. Initially it was guided by the number of hard constraints – it aimed at decreasing their number until the solution was feasible. Once that happened, only the moves that decreased the number of soft constraints and did not introduce any hard constraints, were approved.

It was possible to specify the number of possible iterations of this local search, probability of making a move from certain neighborhood (p_1, p_2, and p_3 respectively), and also the upper limit of the CPU time spent inside the local search.

3.2 Representation

Before any ant algorithm may tackle a combinatorial optimization problem, the problem has to be properly represented. This includes the definition of the construction graph that the ants may traverse constructing the solutions, the definition of the pheromone matrix, i.e. how the solution is represented by the pheromone, and also what type of heuristic information is used and how. In this section we present the representation possibilities that we investigated, and the decisions we finally made.

3.2.1 Construction Graph

One of the cardinal elements of the ACO metaheuristic is the mapping of the problem onto a *construction graph* [17, 18], so that a path through the graph

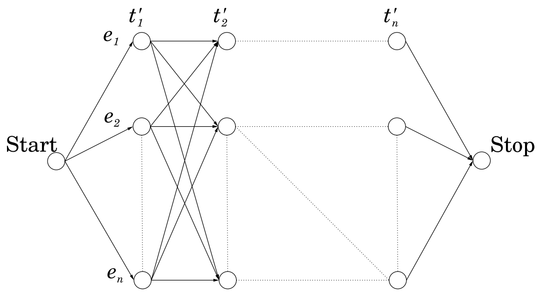

Figure 3.1: Each ant follows a list of *virtual* timeslots, and for each such timeslot $t' \in T'$, it chooses an event $e \in E$ to be placed in this timeslot. At each step an ant can choose any possible transition

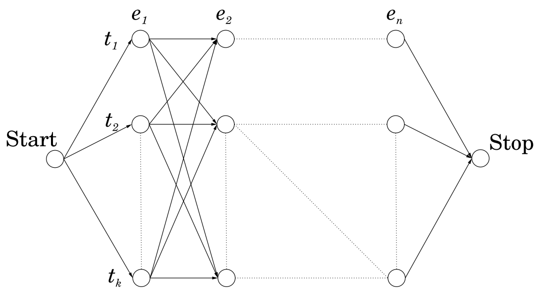

Figure 3.2: Each ant follows a list of events, and for each event $e \in E$, an ant chooses a timeslot $t \in T$. Each event has to be put exactly once into a timeslot, and there may be more than one event in a timeslot, so at each step an ant can choose any possible transition

represents a solution to the problem. In our formulation of the UCTP we are required to assign each of $|E|$ events to one of $|T|$ timeslots. In the most direct representation the construction graph is given by $E \times T$; given this graph we can then decide whether the ants move along a list of the timeslots, and choose events to be placed in them, or move along a list of the events and place them in the timeslots. Fig. 3.1 and Fig. 3.2 depict, respectively, these construction graphs.

As shown in Fig. 3.1, the first construction graph must use a set of virtual timeslots $T' = \{t'_1, \ldots, t'_{|E|}\}$, because exactly $|E|$ assignments must be made in the construction of a timetable, but in general $|T| \ll |E|$. Each of the virtual timeslots maps to one of the actual timeslots. To use this representation then, requires us to define an injection $\iota : T' \to T$, designating how the virtual timeslots relate to the actual ones. One could use for example the injection $\iota : t'_g \mapsto t_h$ with $h = \left\lceil \frac{g \cdot |T|}{|E|} \right\rceil$. In this way, the timetable would be constructed sequentially through the week. However, for certain problems, giving equal numbers of events to each timeslot may be a long way from optimal. Other injection functions are also possible but may contain similar implicit biases.

The simpler representation of the construction graph (Fig. 3.2), where ants walk along a list of events, choosing a timeslot for each, does not require the additional complication of using virtual timeslots and does not seem to have any obvious disadvantages. In fact, it allows us the opportunity of using a heuristically ordered list of events. By carrying out some preprocessing we should be able to order the events so that the most 'difficult' events are placed into the timetable first, when there are still many timeslots with few or no occupied rooms. For these reasons we choose initially to use this representation.

3.2.2 Pheromone Matrix

In a first representation, we let pheromone indicate the absolute position where events should be placed. With this representation the pheromone matrix is given by $\tau(A_i) = \tau, i = 1, \ldots, |E|$, i.e., the pheromone does not depend on the partial assignments A_i. Note that in this case the pheromone will be associated with *nodes* in the construction graph rather than *edges* between the nodes.

A disadvantage of this direct pheromone representation is that the absolute position of events in the timeslots does not matter very much in producing a good timetable. It is the relative placement of events which is important. For example, given a perfect timetable, it is usually possible to permute many groups of timeslots without affecting the quality of the timetable. As a result, this choice of representation can cause slower learning because during construction of solutions, an early assignment of an event to an 'undesirable' timeslot may cause conflicts with many supposedly desirable assignments downstream, leading to a poor timetable. This leads to a very noisy positive feedback signal.

In a second representation the pheromone values are indirectly defined. To do this we use an auxiliary matrix $\mu \in \mathbf{R}_+^{E \times E}$ to indicate which events should (or should not) be put together with other events in the same timeslot. Now, the values $\tau_{(e,t)}(A_i)$ can be expressed in terms of μ and A_i by

$$\tau_{(e,t)}(A_i) = \begin{cases} \tau_{max} & \text{if } A_i^{-1}(t) = \emptyset, \\ \min_{e' \in A_i^{-1}(t)} \mu(e, e') & \text{otherwise.} \end{cases}$$

Giving feedback to these values μ, the algorithm is able to learn which events should *not* go together in the same timeslot. This information can be learned without relation to the particular timeslot numbers. This representation looks promising because it allows the ants to learn something more directly useful to the construction of feasible timetables. However, it also has some disadvantages. For solving the soft constraints certain inter- timeslot relations between events matter, in addition to the intra-timeslot relations. This pheromone representation does not encode this extra information at all.

Some experimentation with the two different pheromone matrices indicated that the first one performed significantly better when the local search procedure was also used. Even though it is not ideal for the reasons stated above, it is capable of guiding the ants to construct timetables which meet the soft constraints as well as the hard ones. The problem of noisy feedback from this representation is also somewhat reduced when using the local search.

Clearly, other pheromone representations are possible, but with the variety of constraints which must be satisfied in the UCTP, it is difficult to design one that encodes all the relevant information in a simple manner. For the moment, the direct coding is the best compromise we have found.

3.2.3 Heuristic Information

We now consider possible methods for computing the heuristic information $\eta_{(e,t)}(A_{i-1})$. A simple method is the following:

$$\eta_{(e,t)}(A_{i-1}) = \frac{1.0}{1.0 + V_{(e,t)}(A_{i-1})}$$

where $V_{(e,t)}(A_{i-1})$ counts the additional number of violations caused by adding (e,t) to the partial assignment A_{i-1}. The function V may be a weighted sum of several or all of the soft and hard constraints. However, due to the nature of the UCTP, the computational cost of calculating some types of constraint violations can be rather high. We can choose to take advantage of significant heuristic information to guide the construction but only at the cost of being able to make fewer iterations of the algorithm in the given time limit. We conducted some investigations to assess the balance of this tradeoff and found that the use of heuristic information did not improve the quality of timetables constructed by the algorithm with local search. Without the use of LS, heuristic information does improve solution quality, but not to the same degree as LS.

3.3 $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System

3.3.1 Approach

As there are quite a few existing types of ant systems, we had to make a decision which one to choose for tackling the UCTP. As there was already some work being done on using Ant Colony System for UCTP at that time [38], we decided to take a closer look at the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS), as it is known to perform well on number of problems. Hence, we attempted to develop a $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the UCTP.

3.3.2 Algorithm Description

Our $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the UCTP is shown in Alg. 6. A colony of m ants is used and at each iteration, each ant constructs a complete event- timeslot assignment by placing events, one by one, into the timeslots. The events are taken in a prescribed order which is used by all ants. The order is calculated before the run based on edge constraints between the events. The choice of

Algorithm 6 $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the UCTP

```
input: A problem instance $I$
$\tau_{max} \leftarrow \frac{1}{\rho}$
$\tau(e,t) \leftarrow \tau_{max} \; \forall \; (e,t) \in E \times T$
calculate $c(e,e') \; \forall \; (e,e') \in E^2$
calculate $d(e)$
sort $E$ according to $\prec$, resulting in $e_1 \prec e_2 \prec \cdots \prec e_n$
while time limit not reached do
  for $a = 1$ to $m$ do
    {construction process of ant $a$}
    $A_0 \leftarrow \emptyset$
    for $i = 1$ to $|E|$ do
      choose timeslot $t$ randomly according to probabilities $p_{e_i,t}$ for event $e_i$
      $A_i \leftarrow A_{i-1} \cup \{(e_i,t)\}$
    end for
    $C \leftarrow$ solution after applying matching algorithm to $A_n$
    $C_{iteration\ best} \leftarrow$ best of $C$ and $C_{iteration\ best}$
  end for
  $C_{iteration\ best} \leftarrow$ solution after applying local search to $C_{iteration\ best}$
  $C_{global\ best} \leftarrow$ best of $C_{iteration\ best}$ and $C_{global\ best}$
  global pheromone update for $\tau$ using $C_{global\ best}$, $\tau_{min}$, and $\tau_{max}$
end while
output: An optimized candidate solution $C_{global\ best}$ for $I$
```

which timeslot to assign to each event is a biased random choice influenced by the pheromone level $\tau_{(e,t)}(A_i)$. The pheromone values are initialized to a parameter τ_{max}, and then updated by a global pheromone update rule.

At the end of the iterative construction, an event-timeslot assignment is converted into a candidate solution (timetable) using the matching algorithm. After all m ants have generated their candidate solution, one solution is chosen based on a fitness function. This candidate solution is further improved by the local search routine. If the solution found is better than the previous global best solution, it is replaced by the new solution. Then the global update on the pheromone values is performed using the global best solution. The values of the pheromone corresponding to the global best solution are increased and then all the pheromone levels in the matrix are reduced according to the evaporation coefficient. Finally, some pheromone values are adjusted so that they all lie within the bounds defined by τ_{max} and τ_{min}. The whole process is repeated, until the time limit is reached.

Some parts of Alg. 6 are now described in more detail. In a pre-calculation for events $e, e' \in E$ the following parameters are determined:

$$\begin{aligned} c(e,e') &:= 1 \text{ if there are students following both } e \text{ and } e',\ 0 \text{ otherwise, and} \\ d(e) &:= |\{e' \in E \setminus \{e\} \mid c(e,e') \neq 0\}| \ . \end{aligned}$$

We define a total order $\prec$ on the events by

$$\begin{aligned} e \prec e' \quad :\Leftrightarrow \quad & d(e) > d(e') \vee \\ & d(e) = d(e') \wedge l(e) < l(e') \ . \end{aligned}$$

Here, $l : E \to \mathbf{N}$ is an injective function that is only used to handle ties. We define $E_i := \{e_1, \ldots, e_i\}$ for the totally ordered events denoted as $e_1 \prec e_2 \prec \ldots \prec e_n$.

$$s = \begin{cases} \text{parameter m} & \text{if } j = 0, \\ 100 & \text{otherwise,} \end{cases} \tag{3.1}$$

Only the solution that causes the fewest number of hard constraint violations is selected for improvement by the LS. Ties are broken randomly. The pheromone matrix is updated only once per iteration, and the global best solution is used for update. Let $A_{global\ best}$ be the assignment of the best candidate solution $C_{global\ best}$ found since the beginning. The following update rule is used:

$$\tau_{(e,t)} = \begin{cases} (1-\rho) \cdot \tau_{(e,t)} + 1 & \text{if } A_{global\ best}(e) = t, \\ (1-\rho) \cdot \tau_{(e,t)} & \text{otherwise,} \end{cases}$$

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

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

3.3.3 Algorithm Performance

Following the development of the $\mathcal{MM}$AS, we wanted to evaluate its performance. One of the first things that we wanted to test, was the comparison of our $\mathcal{MM}$AS with a Random Restart Local Search (RRLS). Such a comparison provides a clear evidence whether a metaheuristic is actually doing something useful, or is it only the local search that helps to solve the problem. Before any comparison could be made however, proper parameters for the $\mathcal{MM}$AS had to be chosen.

Parameters

The development of an effective $\mathcal{MM}$AS for an optimization problem also requires that appropriate parameters be chosen for typical problem instances. We tested several configurations of our $\mathcal{MM}$AS on problem instances from the classes listed in Tab. 1.1. The best results were obtained using the parameters listed in Tab. 3.1.

The values of τ_{min} were calculated so that at convergence (when one 'best' path exists with a pheromone value of τ_{max} on each of its constituent elements, and all other elements in the pheromone matrix have the value τ_{min}) a path

Table 3.1: Parameter configurations used in the comparison.

Parameter	`small`	`medium`	`large`
ρ	0.30	0.30	0.30
$\tau_{max} = \frac{1}{\rho}$	3.3	3.3	3.3
τ_{min}	0.0078	0.0019	0.0019
α	1.0	1.0	1.0
β	0.0	0.0	0.0
m	10	10	10

Algorithm 7 Random Restart Local Search

```
input: A problem instance $I$
while time limit not reached do
  for $a = 1$ to $m$ do
    {random solution creation number $a$}
    $A_0 \leftarrow \emptyset$
    for $i = 1$ to $|E|$ do
      choose timeslot $t$ randomly
      $A_i \leftarrow A_{i-1} \cup \{(e_i, t)\}$
    end for
    $C \leftarrow$ solution after applying matching algorithm to $A_n$
    $C_{iteration\ best} \leftarrow$ best of $C$ and $C_{iteration\ best}$
  end for
  $C_{iteration\ best} \leftarrow$ solution after applying local search to $C_{iteration\ best}$
  $C_{global\ best} \leftarrow$ best of $C_{iteration\ best}$ and $C_{global\ best}$
end while
output: An optimized candidate solution $C_{global\ best}$ for $I$
```

constructed by an ant will be expected to differ from the best path in 20 % of its elements. The value 20 % was chosen to reflect the fact that a fairly large 'mutation' is needed to push the solution into a different basin of attraction for the local search.

Random Restart Local Search

To assess the developed $\mathcal{MM}$AS, we consider whether the ants genuinely learn to build better timetables, as compared to a random restart local search (RRLS). This RRLS iterates the same LS as used by $\mathcal{MM}$AS from random starting solutions and stores the best solution found. Alg. 7 presents briefly the RRLS used.

Table 3.2: Median of the number of soft constraint violations observed in independent trials of $\mathcal{MM}$AS and RRLS on different problem instances, together with the p-value for the null hypothesis that the distributions are equal. In the cases where greater than 50 % of runs resulted in no feasible solution the median cannot be calculated. Here, the fraction of unsuccessful runs is given. (In all other cases 100 % of the runs resulted in feasible solutions.) All infeasible solutions are given the symbolic value ∞. This is correctly handled by the Mann-Whitney test.

Instance	Median of #scv		p-value
	$\mathcal{MM}$AS	RRLS	
`small1`	1	8	$< 2 \cdot 10^{-16}$
`small2`	3	11	$< 2 \cdot 10^{-16}$
`small3`	1	8	$< 2 \cdot 10^{-16}$
`small4`	1	7	$< 2 \cdot 10^{-16}$
`small5`	0	5	$< 2 \cdot 10^{-16}$
`medium1`	195	199	0.017
`medium2`	184	202.5	$4.3 \cdot 10^{-6}$
`medium3`	248	(77.5 %)	$8.1 \cdot 10^{-12}$
`medium4`	164.5	177.5	0.017
`medium5`	219.5	(100 %)	$2.2 \cdot 10^{-16}$
`large`	851.5	(100 %)	$6.4 \cdot 10^{-5}$

Results

We tested both the developed $\mathcal{MM}$AS and the RRLS on previously unseen problem instances made by the generator mentioned in Sect. 1.4.2. For this test study, we generated eleven test instances: five small, five medium, and one large. For each of them, we ran our algorithms for 50, 40, and 10 independent trials, giving each trial a time limit of 90, 900, and 9000 seconds, respectively. All the tests were run on a PC with an AMD Athlon 1100 MHz CPU under Linux using the GNU C++ compiler gcc version 2.95.3. As random number generator we used `ran0` from the Numerical Recipes [34]. For the reproducibility of the results on another architecture, we observed that on our architecture one step of the local search has an average running time of 0.45, 1.4, and 1.1 milliseconds, respectively.

Boxplots showing the distributions of the ranks of the obtained results are shown in Fig. 3.3. The Mann-Whitney test (see [13]) was used to test the hypothesis H_0 that the distribution functions of the solutions found by $\mathcal{MM}$AS and RRLS were the same. The p-values for this test are given in Tab. 3.2, along with the median number of soft constraint violations obtained.

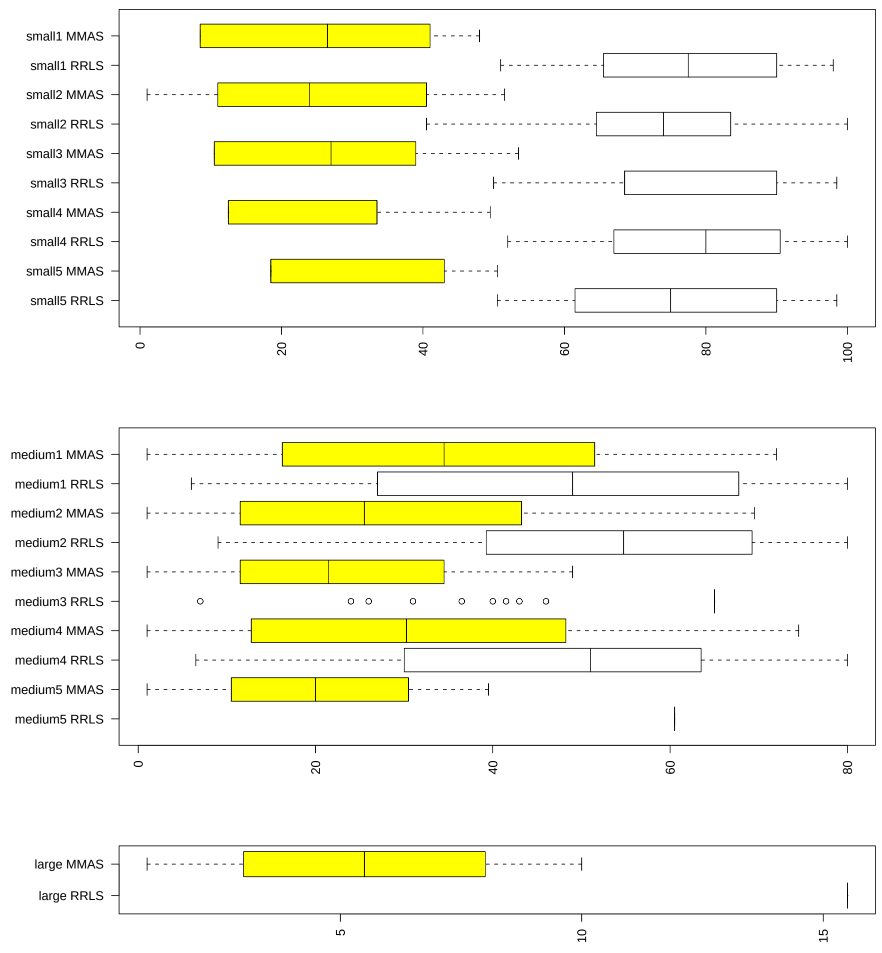

Figure 3.3: Boxplots showing the relative distribution of the number of soft constraint violations for $\mathcal{MM}$AS (shadowed) and RRLS (white) on all test instances. This is the distribution of the ranks of the absolute values in an ordered list, where equal values are assigned to the mean of the covered ranks. A box shows the range between the 25 % and the 75 % quantile of the data. The median of the data is indicated by a bar. The whiskers extend to the most extreme data point which is no more than 1.5 times the interquantile range from the box. Outliers are indicated as circles

For each of the tested problem instances we got with very high statistical significance the result that $\mathcal{MM}$AS performs better than RRLS. For some test instances of medium and large size some runs of RRLS resulted in infeasible solutions. In particular, the RRLS was unable to produce any feasible solution for the large problem instance.

3.4 $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System vs Ant Colony System

Following the encouriging results obtained by the $\mathcal{MM}$AS, when compared to the RRLS, we wanted to investigate how the $\mathcal{MM}$AS compares to other types of ant algorithms. Our choice became the Ant Colony System (ACS), as such an algorithm has been developed in parallel by some other researchers in our group.

3.4.1 Ant Colony System

The algorithms compared – the $\mathcal{MM}$AS and the ACS – 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 $\mathcal{MM}$AS has been presented in detail in Sec. 3.3.2. Here, we will focus on the description of the ACS that was used for comparison.

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:

$$\tau_{(e_i,t_{chosen})} \leftarrow (1-\alpha)\cdot\tau_{(e_i,t_{chosen})} + \alpha\cdot\tau_0 \tag{3.2}$$

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:

$$\tau_{(e,t)} \leftarrow \begin{cases} (1-\rho)\cdot\tau_{(e,t)} + \rho\cdot\frac{g}{1+q(C_{global\_best})} & \text{if } (e,t) \text{ is in } C_{global\_best} \\ (1-\rho)\cdot\tau_{(e,t)} & \text{otherwise,} \end{cases} \tag{3.3}$$

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

Table 3.3: Parameters used by the algorithms.

Parameter Name	$\mathcal{MM}$AS	ACS
m	number of ants	
ρ	pheromone evaporation	
$s(j)$	number of steps of the local search	
τ_0	value with which the pheromone matrix is initialized	
τ_{max}	maximal pheromone level	-
τ_{min}	minimal pheromone level	-
α	-	local pheromone decay
β	-	weight of the hard constraints
γ	-	weight of the soft constraints
g	-	scaling factor

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 β and γ 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. 3.3 summarizes the parameters used by the two algorithms.

3.4.2 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 [38], which were the Iterated Local Search and Simulated Annealing; also against a reference random restart local search algorithm (RRLS) [5], 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.

Table 3.4: Parameter settings used by the algorithms.

Parameter	$\mathcal{MM}$AS	ACS
m	10	10
ρ	0.3	0.1
$s(j)$	10 000 000	$\begin{cases} 50\,000 & j \leq 10 \\ 20\,000 & j \geq 11 \end{cases}$
τ_0	3.3	10.0
τ_{max}	3.3	-
τ_{min}	0.019	-
α	-	0.1
β	-	3.0
γ	-	2.0
g	-	10^{10}

A time limit for producing a timetable has been determined. The comparison of both ant algorithms was performed on the `competition instances`. 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. 3.4 presents the actual parameters used for running the ant algorithms. The same parameters were used for all runs of both ant algorithms.

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. 3.4 presents the comparison of the performance of ant algorithms on `competition instances`. 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. 3.4 contains also additional results obtained by the modified versions of the ACS and $\mathcal{MM}$AS algorithms, as described in Sec. 3.4.4.

3.4.3 Comparison – 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. 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).

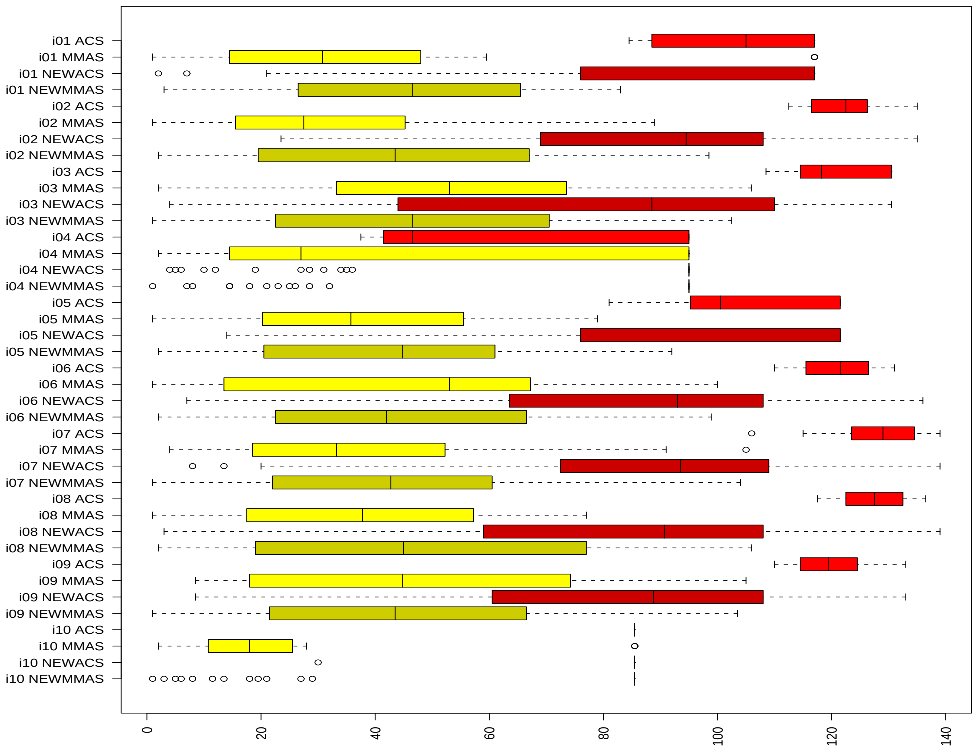

Figure 3.4: Rank comparison of the results obtained by the two ant algorithms (ACS and $\mathcal{MM}$AS) on ten `competition instances` of the problem. Also the performance of the versions of those algorithms (NEWACS and NEW$\mathcal{MM}$AS) modified as described in Sec. 3.4.4 are presented.

- 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.

3.4.4 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. 3.4 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. 3.4), 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.

3.5 $\mathcal{MM}$AS vs Other Metaheuristics

After the comparison of the $\mathcal{MM}$AS to Random Restart Local Search algorithm and also another ant algorithm, we wanted to see, how the performance of the $\mathcal{MM}$AS compares to some other metaheuristics. In order to achieve this, we used the results of comparison of several different metaheuristics presented in [38].

The metaheuristics compared in [38] included:

- Ant Colony Optimization[1],
- Genetic Algorithm,
- Simulated Annealing,
- Iterated Local Search,
- Tabu Search.

As we have shown in Sec. 3.4 that $\mathcal{MM}$AS outperforms the ACS used in this comparison, we only show here how the $\mathcal{MM}$AS compares to the other four metaheuristics.

It is important to mention that the local search routine that the $\mathcal{MM}$AS used, was exactly the same one as the one used by the other metaheuristics. Our algorithm was run exactly the same number of times on the same problem instances as the other metaheuristics. The test were performed on the same computer system with the same time limits. Our $\mathcal{MM}$AS has been trained on separate training set of instances – different from the ones used for performance evaluation.

We present here the results of the comparison of the $\mathcal{MM}$AS with the results obtained by other metaheuristics for only selected instances that have been tested for two out of three problem classes. We decided not to focus on the `small` class of problems, as most metaheuristics (including $\mathcal{MM}$AS) managed to obtain equally good (optimal) results. Instead we focused on `medium` and `large` problem classes.

We chose for illustration one instance from the `medium` problem class and one instance from the `large` problem class. Results for other instances of these classes of the problem indicate that the results presented are representative. Fig. 3.5 presents the comparison of results obtained by the reference mataheuristics as well as $\mathcal{MM}$AS on a `medium` problem instance. The figure shows the actual solutions found, rank comparison, and also the percentage of infeasible solutions found. Fig. 3.6 presents the similar data for one of the `large` instances.

In order to evaluate the performance of the $\mathcal{MM}$AS comparing to other metaheuristics, we used pairwise Wilcoxon rank sum test. The obtained *p-values* were then adjusted using the Holm method. Tab. 3.5 presents the *p-values* obtained.

[1]More particular: Ant Colony System – the very same one that we used for comparison with $\mathcal{MM}$AS presented in Sec. 3.4

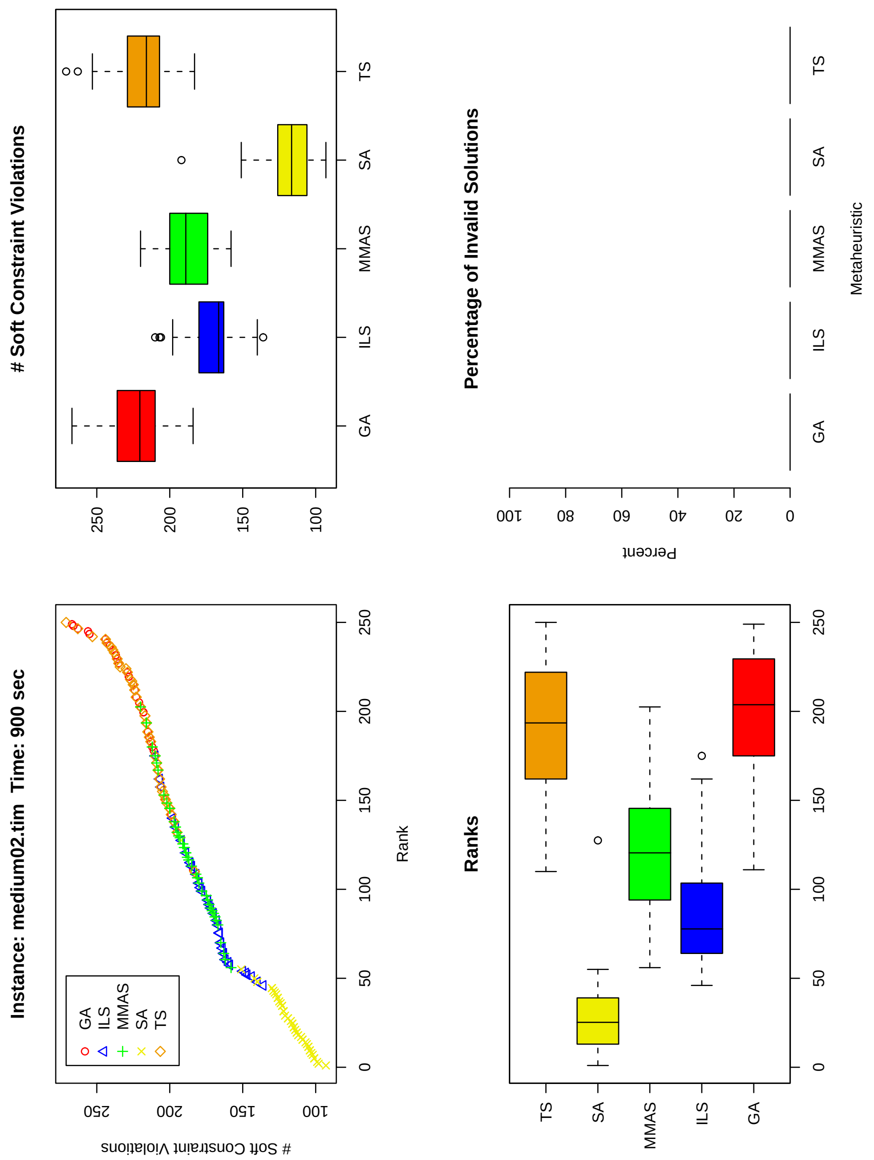

Figure 3.5: Charts showing the relative performance of $\mathcal{MM}$AS compared to other reference metaheuristics on `medium02` problem instance. All algorithms used the same neighborhood structure and local search (when applicable).

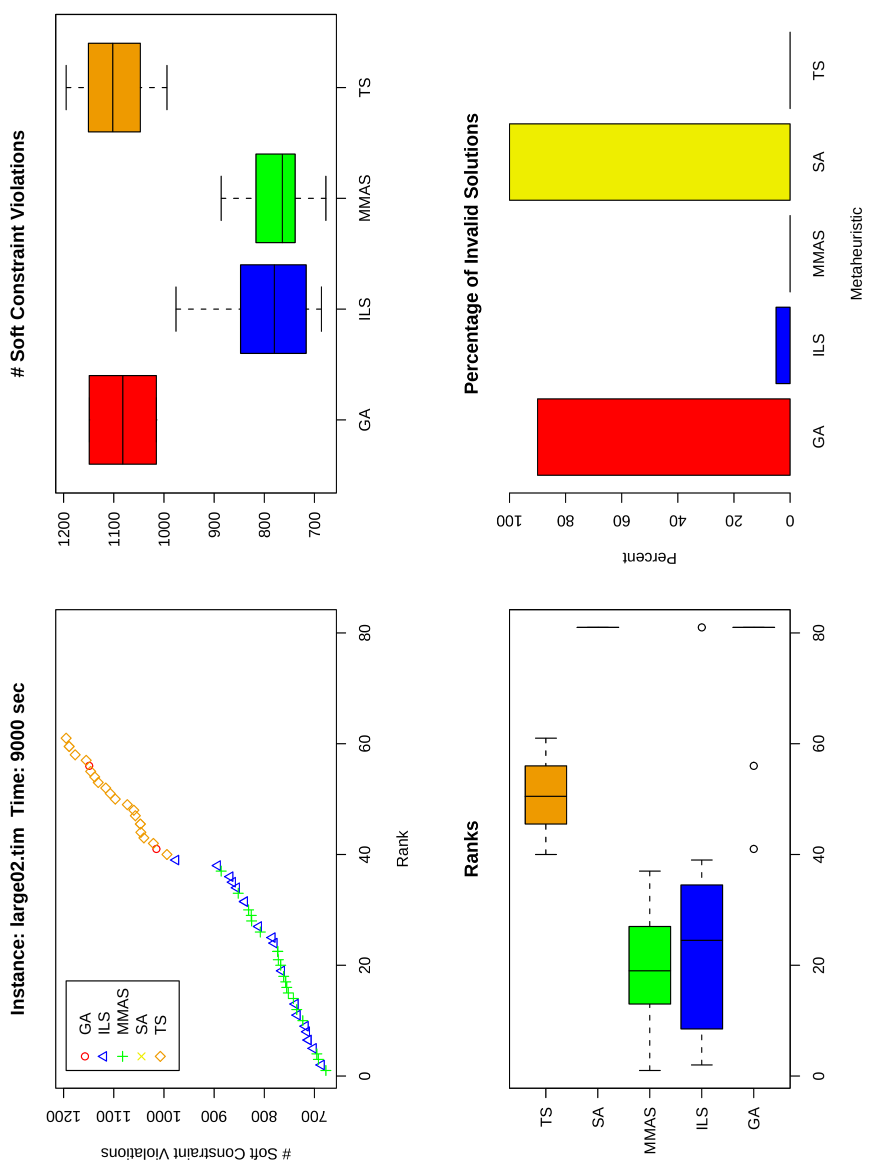

Figure 3.6: Charts showing the relative performance of $\mathcal{MM}$AS compared to other reference metaheuristics on `large02` problem instance. All algorithms used the same neighborhood structure and local search (when applicable).

Table 3.5: Pairwise comparisons using Wilcoxon rank sum test (p-value adjustment method: holm)

Algorithm	`medium02`	`large02`
GA	9.5e-13	1.2e-07
ILS	3.7e-06	0.41
SA	3.5e-16	8.0e-08
TS	5.6e-12	4.1e-07

3.6 $\mathcal{MM}$AS2

Following the development of the $\mathcal{MM}$AS algorithm, we thought of ways to further improve its performance. We noticed in case of the $\mathcal{MM}$AS that reaching feasibility took quite a long time (especially for `large` problem instances). It was especially obvious that the local search takes very long time before reaching local optimum for those types of problems. Hence, the first idea was to substitute the local search that was supplied by the Metaheuristic Network with the one written by us.

Apart from changing the local search we decided also to add some preprocessing stage that could help to reduce the problem a bit. Also, we decided to change slightly the representation and redefine the job for the ants – now they not only should choose the timeslot for the events, but also the room. Hence, we resigned from using the matching algorithm, which we found quite time consuming.

The sections below present those new design considerations in more detail.

3.6.1 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 events[2] 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 timeslots[3], 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].

3.6.2 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:

$$\begin{cases} t = p \div |R| \\ r = p \mod |R| \end{cases} \tag{3.4}$$

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.

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

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

[4]This 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.

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.

3.6.3 The Algorithm

The basic mode of operation of the new $\mathcal{MAX}$-$\mathcal{MIN}$ 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. 3.6.1), 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. 8 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 one[5], and the event is placed in one of the rooms of this additional timeslot. This of course results in an infeasible solution as number

[5]Initially $|T| = 45$

Algorithm 8 $\mathcal{MAX}$-$\mathcal{MIN}$ 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 45[6]. 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. 3.6.4. 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 $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System introduces upper and lower limits on the pheromone values – τ_{max} and τ_{min} respectively [44] – 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:

$$prob = \gamma \cdot [\frac{q(C_{global\ best})}{q(C_{local\ best})}]^{\alpha} \tag{3.5}$$

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):

$$\tau_{ep} \leftarrow \begin{cases} (1-\rho)\cdot(\tau_{ep} + \Delta\tau_{ep}) & \text{if } ep \text{ is in } C_{best} \\ (1-\rho)\cdot\tau_{ep} & \text{otherwise,} \end{cases} \tag{3.6}$$

[6]Only 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:

$$\Delta\tau_{ep} = \tau_{max} \cdot \frac{n}{|R|} \tag{3.7}$$

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

$$\tau_{ep} \leftarrow \begin{cases} \tau_{min} & \text{if } \tau_{ep} < \tau_{min}, \\ \tau_{max} & \text{if } \tau_{ep} > \tau_{max}, \\ \tau_{ep} & \text{otherwise.} \end{cases} \tag{3.8}$$

3.6.4 Local Search

The LS used here by the $\mathcal{MM}$AS 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`.

Hard Constraints

As described in section Sec. 3.6.2, 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 end up 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.

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.

3.6.5 Parameters

We investigated the choice of parameters for the $\mathcal{MM}$AS2 with regard to the imposed time limits [3]. We chose two 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 [42, 43, 44] 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. 3.7 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)[7]. The results presented, were obtained for the competition04 instance.

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. 3.8. 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

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

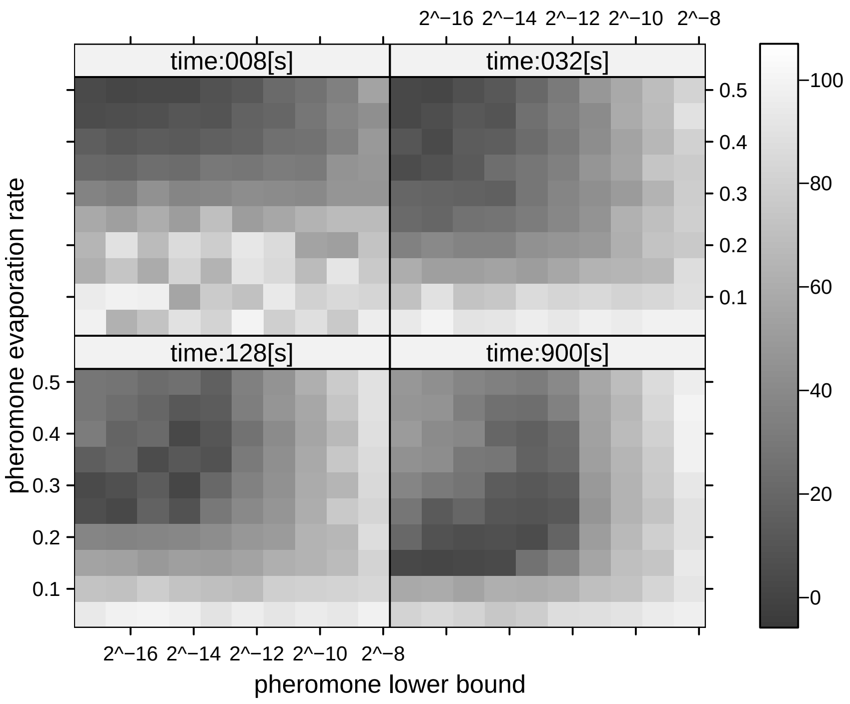

Figure 3.7: The ranks of the solution means for the `medium` problem instance with regard to the algorithm run-time. The ranks of the solutions are depicted (gray-shade-coded) as function of the pheromone lower bound τ_{min}, and pheromone evaporation rate ρ.

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, 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.

The algorithm accepts number of command line parameters that are used to tune its performance. Table 3.6 presents the parameters used together with short description and the values used for obtaining the results submitted to the International Timetabling Competition.

Table 3.6: 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

3.6.6 Performance Analysis

Finally once the new $\mathcal{MM}$AS2 algorithm has been developed, we wanted to see how the improvements made in $\mathcal{MM}$AS2 over the $\mathcal{MM}$AS translate into performance. We made a comparison of the performance of $\mathcal{MM}$AS2 and $\mathcal{MM}$AS using the same test instances that were used for comparing the $\mathcal{MM}$AS with other metaheuristics.

Fig. 3.9 shows the comparison of performance of both $\mathcal{MAX}$-$\mathcal{MIN}$ ant algorithms on the `medium02` problem instance and Fig. 3.10 the performance on `large02` problem instance. It is clear that performance of the $\mathcal{MM}$AS2 is much better than the performance of $\mathcal{MM}$AS. In fact the solution obtained by the $\mathcal{MM}$AS2 entirely dominate all the solution obtained by previous version of $\mathcal{MM}$AS.

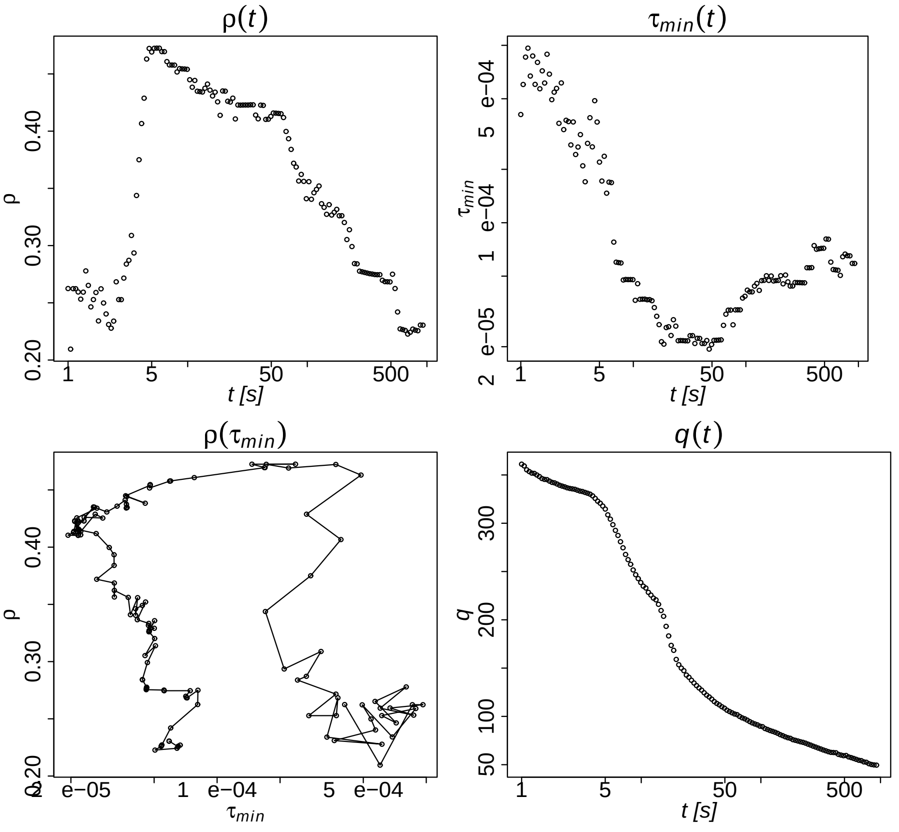

Figure 3.8: Analysis of average best ρ and τ_{min} parameters as a function of time assigned for the algorithm run (the upper charts). Also, the relation between best values of ρ and τ_{min}, as changing with running time, and the average quality of the solutions obtained with the current best parameters as a function of run-time (lower charts).

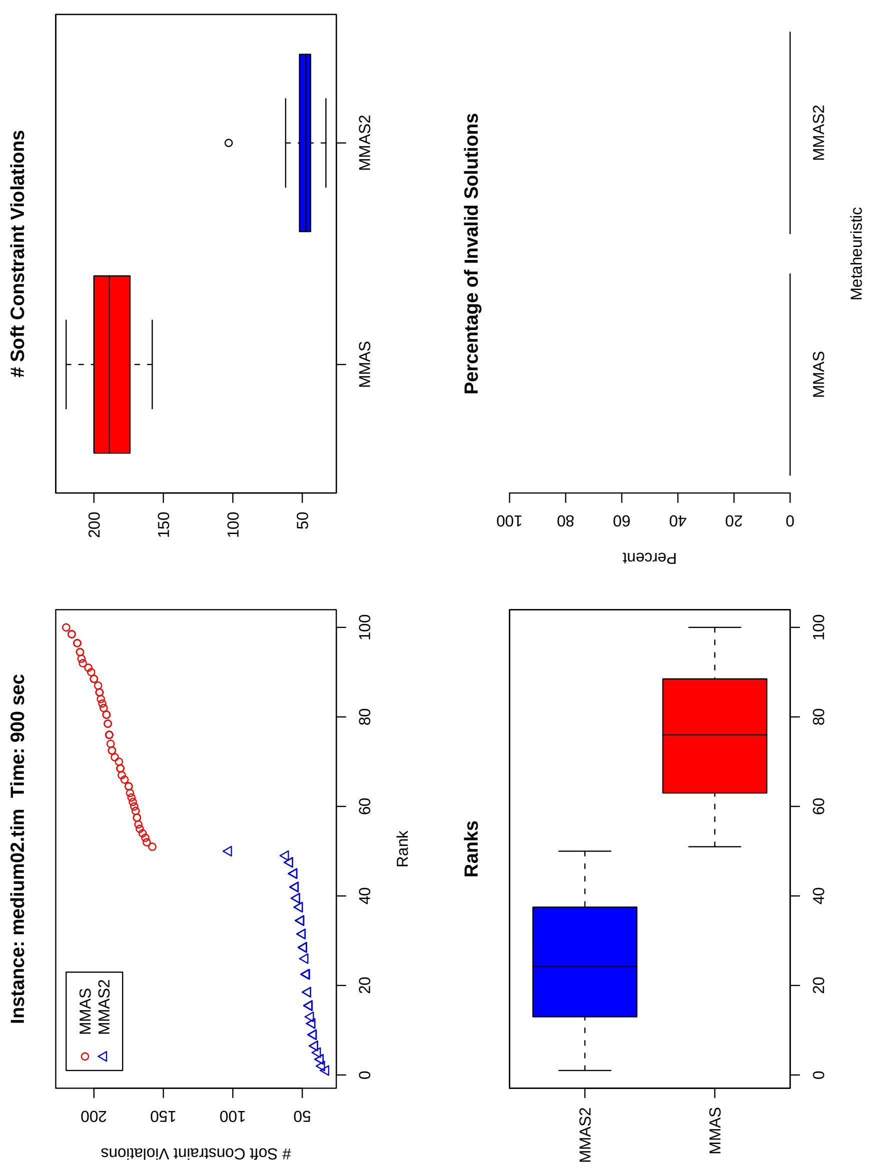

Figure 3.9: Charts showing the relative performance of the $\mathcal{MM}$AS2 compared to the $\mathcal{MM}$AS on `medium02` problem instance. All algorithms used the same neighborhood structure and local search (when applicable).

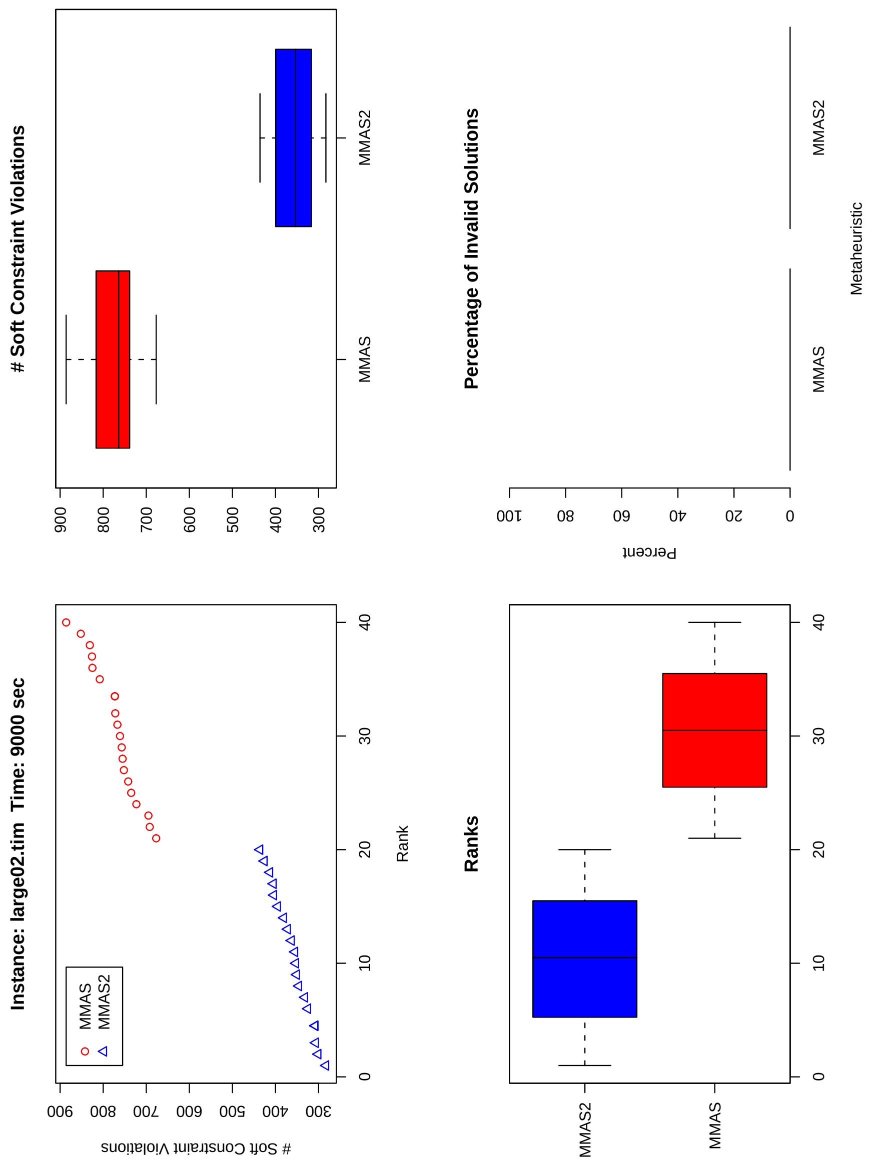

Figure 3.10: Charts showing the relative performance of the $\mathcal{MM}$AS2 compared to the $\mathcal{MM}$AS on `large02` problem instance. All algorithms used the same neighborhood structure and local search (when applicable).

Chapter 4

Conclusions

We have presented the timetabling problem. We have shown that there are several variants of the problem, yet they all are quite complex to solve. They appear to be NP-complete due to several factors. We focused on one particular variant – the University Course Timetabling Problem, and we have presented its definition, characteristics, and test instances. We have analysed the problem and shown possible approaches for tackling this problem. We have developed the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for solving this problem and shown that the results obtained are comparable and sometimes event better from other approaches.

We devised a construction graph and a pheromone model appropriate for university course timetabling. Using these we were able to specify the first $\mathcal{MM}$AS for this problem. Compared to a random restart local search, it showed significantly better performance on a set of typical problem instances, indicating that it can guide the local search effectively. Our algorithm underlined the fact that ant systems are able to handle problems with multiple heterogeneous constraints. Even without using problem-specific heuristic information it is possible to generate good solutions. With the use of a basic first-improvement local search, we found that $\mathcal{MM}$AS permits a quite simple handling of timetabling problems. With an improved local search, exploiting more problem specific operators, we would expect a further improvement in performance.

Later we compared the performance of the $\mathcal{MM}$AS with the Ant Colony System. 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.

We have done some research on the parameterization of the ACO algorithms. 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.

The possible solution for further improvement of the results obtained 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.

Eventually we developed a second version of the $\mathcal{MM}$AS algorithm which clearly outperformed the previous version as well as all other metaheuristics presented in this paper. However, it is important to notice that other metaheuristics presented here were constrained by the neighborhood structure and local search used. Hence, their results may be better should for instance a more efficient local search be used. So much improvement in the performance was possible due to improvement of the representation, additional preprocessing phase, and improved local search.

4.1 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[1]. 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.

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

Another important line of research that we plan to pursue is application of ant algorithms to the multiobjective optimization problems. The UCTP may be used as the first sample problem. When each soft constraint is considered a separate objective, this problem becomes a multiobjective optimization problem. It will be quite interesting how well may the ants deal with such problems and whether a multiobjective approach would perhaps help solving the single objective version.

Also there is a clear need for better understanding of the search landscape of the timetabling problems. So far there has been little research into the landscape analysis of this problem. A better understanding would facilitate development of more efficient algorithms. The algorithms available today, even though they are able to tackle the problem, they have rather large difficulty in solving more difficult problems to optimality. More research into landscape analysis should help to improve on that.

Acknowledgments

We would like to thank Ben Paechter and Olivia Rossi-Doria for the implementation of data structures and routines for the local search for our first version of the ant algorithm for the University Course Timetabling Problem.

The Ant Colony System used for comparison with $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System has been implemented by Max Manfrin. It is used here for illustrating particular qualities of $\mathcal{MM}$AS. The ACS development has not been part of the research of authors of this paper.

The results of the metaheuristics used as a reference to performance of the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System have been obtained with the use of algorithms developed by Marco Chiarandini (SA), Olivia Rossi-Doria (GA), Luis Paquete (ILS), Monaldo Mastrolili (TS).

The boxplots presenting the performance of various metaheuristics were prepared with the use of scripts (and expertise) of Michael Sampels.

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 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.

Published Papers

[1] M. Kisiel-Dorohinicki and K. Socha. Crowding Factor in Evolutionary Multi-Agent System For Multiobjective Optimization. In H. R. Arabnia, editor, *Proceedings of IC-AI'01 – International Conference on Artificial Inteligence*, volume 1, June 2001.

[2] M. Kisiel-Dorohinicki, K. Socha, and A. Gagatek. Applying Mechanism of Crowd in Evolutionary MAS for Multiobjective Optimization. In *Proceedings of KAEIOG'2001 – V Krajowa Konferencja: Algorytmy Ewolucyjne i Optymalizacja Globalna*, 2001.

[3] K. Socha. The Influence of Run-Time Limits on Choosing Ant System Parameters. In E. Cantu-Paz *et al.*, editor, *Proceedings of GECCO 2003 – Genetic and Evolutionary Computation Conference*, LNCS. Springer-Verlag, Berlin, Germany, July 2003.

[4] K. Socha and M. Kisiel-Dorohinicki. Agent-based Evolutionary Multiobjective Optimisation. In *Proceedings of CEC'02 – Congress on Evolutionary Computation*, volume 1, pages 109–114, May 2002.

[5] K. Socha, J. Knowles, and M. Sampels. A $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the University Timetabling Problem. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, LNCS. Springer-Verlag, Berlin, Germany, September 2002.

[6] K. Socha, M. Sampels, and M. Manfrin. 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 *LNCS*. Springer-Verlag, Berlin, Germany, April 2003.

Bibliography

[7] E. B. Baum. Iterated Descent: A Better Algorithm for Local Search in Combinatorial Optimization Problems. In D. Touretzky, editor, *Proceedings of Neural Information Processing Systems*, November 1987.

[8] E. K. Burke, D. G. Elliman, and R. F. Weare. A University Timetabling System Based on Graph Colouring and Constraint Manipulation. *Journal of Research on Computing in Education*, 27(1):1–18, 1994.

[9] E. K. Burke, J. H. Kingston, and P. A. Pepper. A Standard Data Format for Timetabling Instances. In E. Burke and M. Carter, editors, *Proceedings of the 2nd International Conference on Practice and Theory of Automated Timetabling (PATAT 1997)*, volume 1408 of *LNCS*, pages 309–321. Springer-Verlag, 1998.

[10] E. K. Burke, J. P. Newall, and R. F. Weare. A memetic algorithm for university exam timetabling. In E. Burke and P. Ross, editors, *Proceedings of the 1st International Conference on Practice and Theory of Automated Timetabling (PATAT 1995)*, volume 1153 of *LNCS*, pages 241–251. Springer-Verlag, 1996.

[11] E. Collingwood, P. Ross, and D. Corne. A Guide to GATT. Technical report, University of Edinburgh, 1996.

[12] A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics For High-School Timetabling. *Computational Optimization And Applications Journal*, 9(3):277–298, 1998.

[13] W. J. Conover. *Practical Nonparametric Statistics.* John Wiley & Sons, 3rd edition, 1999.

[14] T. B. Cooper and J. H. Kingston. The complexity of timetable construction problems. In E. Burke and P. Ross, editors, *Proceedings of the 1st International Conference on Practice and Theory of Automated Timetabling (PATAT 1995)*, volume 1153 of *LNCS*, pages 283–295. Springer-Verlag, 1996.

[15] D. de Werra. The combinatorics of timetabling. *European Journal of Operational Research*, 96:504–513, 1997.

[16] L. Di Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. In E. Burke and W. Erben, editors, *Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT 2000)*, volume 2079 of *LNCS*, pages 104–117. Springer-Verlag, 2001.

[17] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.

[18] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26:29–41, 1996.

[19] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, 1966.

[20] F. Glover. Tabu search - part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[21] F. Glover. Tabu search - part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[22] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, 1975.

[23] D. S. Johnson. Local Optimization and the Travelling Salesman Problem. In *Proceedings of 17th Colloquium on Automata, Languages, and Programming*, volume 443 of *LNCS*, pages 446–461. Springer-Verlag, 1990.

[24] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[25] J. H. Kingston. Modeling Timietabling Problems with STTL. In E. Burke and W. Erben, editors, *Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT 2000)*, volume 2079 of *LNCS*, pages 309–321. Springer-Verlag, 2001.

[26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

[27] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[28] H. R. Lorenco, O. Martin, and T. Stützle. Iterated Local Search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2002.

[29] V. Maniezzo and A. Carbonaro. Ant Colony Optimization: an Overview. In C. Ribeiro, editor, *Essays and Surveys in Metaheuristics.* Kluwer Academic Publishers, 2001.

[30] O. Martin, S. W. Otto, and E. W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5:299–326, 1991.

[31] O. C. Martin and S. W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[32] A. Meisels, E. Gudes, and G. Solotorevsky. Combining rules and constraints for employee timetabling. *International Journal of Intelligent Systems*, 12:419–439, 1997.

[33] D. Merkle, M. Middendorf, and H. Schmeck. Ant Colony Optimization for resource-constrained project scheduling. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, pages 893–900. Morgan Kaufmann Publishers, 2000.

[34] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 2nd edition, 1993.

[35] R. C. Read, editor. *Graph Colouring Algorithms In Graph Theory and Computing.* Academic Press, 1972.

[36] I. Rechenberg. Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment, Library translation*, August 1965.

[37] L. P. Reis and E. Oliveira. A Language for Specifying Complete Timetabling Problems. In E. Burke and W. Erben, editors, *Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT 2000)*, volume 2079 of *LNCS*, pages 322–341. Springer-Verlag, 2001.

[38] O. Rossi-Doria, M. Sampels, M. Chiarandini, J. Knowles, M. Manfrin, M. Mastrolilli, L. Paquete, and B. Paechter. A comparison of the performance of different metaheuristics on the timetabling problem. In E. K. Burke and P. De Causmaecker, editors, *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT 2002) (to appear)*, 2003.

[39] A. Schaerf. A survey of automated timetabling. Centrum voor Wiskunde en Informatica (CWI), 1995.

[40] T. Stützle and M. Dorigo. *ACO Algorithms for the Quadratic Assignment Problem.* McGraw-Hill, 1999.

[41] 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.

[42] T. Stützle and H. Hoos. Improvements on the ant system: A detailed report on $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. Technical Report AIDA-96-12 – Revised version, Darmstadt University of Technology, Computer Science Department, Intellectics Group, 1996.

[43] T. Stützle and H. Hoos. *The $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and Local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Combinatorial Global Optimisation.*, pages 313–329. Kluwer Academic Publishers, 1998.

[44] T. Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[45] H. M. M. ten Eikelder and R. J. Willemen. Some complexity aspects of secondary school timetabling problems. In E. Burke and W. Erben, editors, *Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT 2000)*, volume 2079 of *LNCS*, pages 18–29. Springer-Verlag, 2001.

Appendix A
Full Versions of Published Papers