Open access • Journal Article • DOI:10.1007/S10951-006-8495-8

# An effective hybrid algorithm for university course timetabling — Source link

Marco Chiarandini, Mauro Birattari, Krzysztof Socha, Olivia Rossi-Doria

**Institutions:** University of Southern Denmark, Université libre de Bruxelles, Edinburgh Napier University

**Published on:** 01 Oct 2006 - Journal of Scheduling (Kluwer Academic Publishers)

**Topics:** Hybrid algorithm, Tabu search, Metaheuristic, Algorithm engineering and Simulated annealing

J Sched (2006) 9:403–432
DOI 10.1007/s10951-006-8495-8

# An effective hybrid algorithm for university course timetabling

**Marco Chiarandini · Mauro Birattari · Krzysztof Socha · Olivia Rossi-Doria**



**Abstract** The university course timetabling problem is an optimisation problem in which a set of events has to be scheduled in timeslots and located in suitable rooms. Recently, a set of benchmark instances was introduced and used for an 'International Timetabling Competition' to which 24 algorithms were submitted by various research groups active in the field of timetabling. We describe and analyse a hybrid metaheuristic algorithm which was developed under the very same rules and deadlines imposed by the competition and outperformed the official winner. It combines various construction heuristics, tabu search, variable neighbourhood descent and simulated annealing. Due to the complexity of developing hybrid metaheuristics, we strongly relied on an experimental methodology for configuring the algorithms as well as for choosing proper parameter settings. In particular, we used racing procedures that allow an automatic or semi-automatic configuration of algorithms with a good save in time. Our successful example shows that the systematic design of hybrid algorithms through an experimental methodology leads to high performing algorithms for hard combinatorial optimisation problems.

The work was carried out while Marco Chiarandini was with the Intellectics Group, at the Computer Science Department of the Darmstadt University of Technology, Hochschulstraße 10, 64283 Darmstadt, Germany.

M. Chiarandini (✉)
Department of Mathematics and Computer Science,
University of Southern Denmark,
Campusvej 55, 5230 Odense M, Denmark
e-mail: marco@imada.sdu.dk

M. Birattari · K. Socha
Université Libre de Bruxelles, IRIDIA,
Av. Franklin D. Roosevelt 50, CP 194/6, 1050 Bruxelles, Belgium
e-mail: {mbiro, ksocha}@ulb.ac.be

O. Rossi-Doria
Napier University, School of Computing, 10 Colinton Road,
Edinburgh, EH10 5DT, Scotland
e-mail: o.rossi-doria@napier.ac.uk

**Keywords** University course timetabling · Local search methods · Metaheuristics · Hybrid algorithms · Experimental methodology · Algorithm engineering

## 1. Introduction

Timetabling can be generally defined as the activity of assigning, subject to constraints, a number of events to a limited number of time periods and locations such that desirable objectives are satisfied as nearly as possible (Wren, 1996). Practical cases where such activity arises are, among others, educational timetabling, employee timetabling, sport timetabling, transport timetabling and communication timetabling. Educational timetabling can be sub-divided into three main classes: school timetabling, course timetabling and exam timetabling (De Werra, 1985).

In university course timetabling, a set of lectures must be scheduled into rooms and timeslots subject to constraints that are usually divided in two categories: 'hard' and 'soft'. Hard constraints must be strictly satisfied with no violation allowed, while in the case of soft constraints it is desirable, but not essential, to minimise violations. Typical of course timetabling, with respect to school and exam timetabling, are the availability of a limited number of timeslots and the requirements of (i) allocating lectures only into suitable rooms, (ii) having no more than one lecture per room and (iii) scheduling lectures with common students in different timeslots. It is then desirable that each student has the best balance between too many or too few courses in a day (Schaerf, 1999). Constraints and their importance differs however, significantly among countries and among institutions (Carter and Laporte, 1997).

Course timetabling can be formulated as an optimisation problem. Given the large number of events (lectures) to be scheduled and the diversity of constraints, the problem to be solved can be very difficult and a wide variety of solution approaches from the field of Operations Research and Artificial Intelligence have been proposed. Many applications of different solution techniques to timetabling have been published in the recent years and the proceedings of the biannual International Conference on the Practice and Theory of Automated Timetabling (Burke and Ross, 1996; Burke and Carter, 1998; Burke and Erben, 2001; Burke and de Causmaecker, 2003; Burke and Trick, 2005) collect the state of the art in this field. Metaheuristics are the most used techniques and the current trend indicates that multi-stage or hybrid metaheuristics are the most successful approaches. However, all implementations are specifically designed for particular cases of timetabling and an objective comparison of the methods is rarely possible.

Current research has focused on the formulation of standard timetabling problems, which include supersets of constraints, where portable programmes can be compared (the Timetabling Problem Database maintained by Merlot[1] is an example). Generalised frameworks that can handle a wide range of problems or that provide the practitioners with as many implemented components as possible constitute another important direction of research. Examples are standardised models for handling constraints (Chand, 2004; Custers et al., 2005) or libraries for metaheuristics such as EasyLocal ++ (Gaspero and Schaerf, 2002) and HotFrame (Fink and Voß, 2002), or other commercial packages like SpaceMap (Burke et al., 2005) and Harmony (Post and Veltman, 2005). However, as correctly pointed out already by De Werra (1985), the idea of developing a universal timetabling solver which could be used everywhere does not seem reasonable. The diversity of timetabling cases makes very unlikely the existence of an effective heuristic method which is good for all constraints of different nature and for all their densities. The attention

[1] L. Merlot. "Operations Research Group: Timetabling Problem Database." 2003. <http://www.or.ms.unimelb.edu.au/timetabling/> (22 October 2004)

should be therefore shifted towards automatic systems that handle a variety of heuristics (Burke and Petrovic, 2002).

In this article, we introduce a systematic methodology for automatically and efficiently selecting and configuring an appropriate algorithm for the problem at hand. This methodology is based on the racing algorithm introduced by Birattari et al. (2002). Racing algorithms can be used for choosing among a large set of techniques or different configurations of algorithms. Each candidate technique is evaluated sequentially on available instances and the candidates that show poor results are discarded as soon as sufficient statistical evidence is gathered against them. This technique is suitable for two relevant scenarios: guiding the development of highly performing algorithms for the specific case of timetabling and selecting automatically among a range of already implemented algorithmic functions and algorithm parameters those that best fit the class of instances at hand. In this work, an innovative use of this methodology is reached by turning the racing algorithm, originally fully automatic, into an interactive procedure. This characteristic is particularly helpful in the first scenario, the development phase, where new candidate algorithms can be inserted in the race on the basis of ongoing results.

The application of this experimental methodology is presented in a practical context where an algorithm is required for solving a specific class of instances of a specific problem and must be delivered within a precise deadline and with certain guarantees of performance. Such a realistic context was provided by the International Timetabling Competition.[2] This competition, held in 2002–2003, aimed at attracting researchers of the field and at comparing their results on a new class of randomly generated instances for the university course timetabling problem proposed by Paechter.[3] The problem consists of three hard constraints and three soft constraints, each representative of different typologies. The problem, the instance class, the rules and the deadline for the entry of the submission were published on the web when the competition was officially announced. The competition had 24 feasible submissions from all over the world. As members of the Metaheuristics Network, which organised the competition, we could submit an algorithm but we were not allowed to win the prize. In fact, according to the evaluation criterion of the competition, our algorithm outperformed the official winner which, in turn, outperformed, by a quite large margin, all the other submissions.

The algorithm submitted is a hybrid algorithm mainly based on construction heuristics and metaheuristics, although it also uses, in some circumstances, a fast exact assignment of rooms to events in a single timeslot. The algorithm deals separately with hard and soft constraints. Local search and tabu search procedures are used for solving the hard constraints, while a timetable is improved in terms of soft constraints by means of variable neighbourhood descent and simulated annealing. In particular, simulated annealing plays a significant role. The algorithm was developed, configured and tuned through the race-based experimental methodology.

The article is organised as follows. We first introduce the problem and the benchmark instances in Section 2. We then describe the development phase in Section 3 in which a novel experimental methodology to select among many possible algorithmic configurations is introduced and indications of general relevance for the use of metaheuristics are derived. In Section 4, we describe the details of the hybrid algorithm and in Section 5 we analyse the contribution of its components. We conclude discussing the insights of general applicability arose from our study in Section 6.

[2]The International Timetabling Competition was organised by Paechter and Gambardella members of the Metaheuristics Network and was sponsored by PATAT. For details on goals and rules of the competition we refer to the official web site. Metaheuristics Network. "International Timetabling Competition." 2003. <http://www.idsia.ch/ Files/ttcomp2002/> (22 October 2004).

[3]Ben Paechter. "Ben Paechter's Home Page." 1999. <http://www.dcs.napier.ac.uk/~benp> (22 October 2004).

## 2. The university course timetabling problem

The university course timetabling problem (UCTP) that we consider was proposed in the context of the International Timetabling Competition and also treated in the research of the Metaheuristics Network. It is an abstraction of a real UCTP arising at Napier University in Edinburgh and we denote it as UCTP-C.

### 2.1. Problem description

In the UCTP-C are given a set of events (classes) $E$ to be scheduled into 45 timeslots (5 days of 9 timeslots each), a set of rooms $R$ where events can take place and a set of students $S$. Every event has associated a list of students $L^E$ who have to attend it. Moreover, each event has a set of required features $F^E$, and each room has a size $S^R$ and a set of satisfied features $F^R$. A feasible timetable is one in which all events are assigned a timeslot and a room so that the following three hard constraints are satisfied:

H1: Only one event is assigned to each room at any timeslot.
H2: The room is big enough for hosting all attending students and satisfies all the features required by the event.
H3: No student attends more than one event at the same time.

In addition, a candidate timetable receives a penalty cost for violating any of the following three soft constraints:

S1: A student should not have a class in the last slot of a day.
S2: A student should not have more than two classes in a row.
S3: A student should not have a single class on a day.

Note that the soft constraints are representative of three different types of constraints: constraint S1 can be checked without knowledge of the rest of the timetable; S2 can be checked while building a solution; and S3 can be checked only when the timetable is complete and all events have been assigned a timeslot.

The objective is to find a feasible assignment that minimises the number of soft constraint violations. An infeasible assignment is considered useless and is therefore discarded.

We can formalise the problem as follows. Let $\mathcal{A}$ be the set of all possible complete assignments, i.e. each event with a room and a timeslot, $\Theta = \{\mathsf{H1}, \mathsf{H2}, \mathsf{H3}\}$ be the set of the hard constraints and $\tilde{\mathcal{A}} \subseteq \mathcal{A}$ be the set of *feasible* assignments, i.e. those that satisfy the constraints in $\Theta$. Defined an *evaluation function* $f(a)$ which associates to each assignment $a \in \tilde{\mathcal{A}}$ a penalty cost $f(a)$ that accounts for violations of constraints in $\Sigma = \{\mathsf{S1}, \mathsf{S2}, \mathsf{S3}\}$, the task is finding an assignment $a^* \in \tilde{\mathcal{A}}$ of minimal cost, that is, an assignment such that $f(a^*) \leq f(a)$ for all $a \in \tilde{\mathcal{A}}$.

### 2.2. The evaluation function

The evaluation function $f(a)$ defines the quality of a feasible assignment $a \in \tilde{\mathcal{A}}$ and is given by

$$f(a) = \sum_{s \in S} (f_1(a, s) + f_2(a, s) + f_3(a, s))$$

where

$f_1(a, s)$ is the number of times a student $s$ under assignment $a$ has a class in the last timeslot of the day;

$f_2(a, s)$ is the number of times a student $s$ under assignment $a$ has more than two consecutive classes, each time weighted by the number of classes exceeding two in a sequence (for example, $f_2(a, s) = 1$ if $s$ has three consecutive classes, $f_2(a, s) = 2$ if $s$ has four consecutive classes, etc.);

$f_3(a, s)$ is the number of times a student $s$ under assignment $a$ has to attend a single event on a day (for example, $f_1(a, s) = 2$ if $s$ has 2 days with only one class).

The smaller the function $f(a)$ the better is the quality of the assignment.

### 2.3. The benchmark instances

The 20 instances of the competition were generated by a random generator to reflect real-world UCTP instances at Napier University in Edinburgh. By construction it is known that for each instance there exist at least one assignment with an evaluation function value equal to zero. We report few statistical data of these 20 instances in Table 1. The number of timeslots is fixed to 45 while events range between 350 and 440 and students between 200 and 300. The number of rooms suitable for each event (rooms/event) is quite low. This suggests that an exact algorithm may be feasible for the assignment of rooms to scheduled events. The number of events per student (events/student) and the number of students per event (students/event) are, instead, informative of the difficulty to satisfy all the constraints. Since it is known that for each instance a zero cost assignment exists, for constraint S3 there exists a feasible assignment that uses only 40 timeslots. Indicatively, with 400 events and 10 rooms, this corresponds to completely filling up the 40 timeslots available with events!

Concerning the hard constraints, the UCTP-C can partially be reduced to a graph colouring problem in which events are the vertices of a graph and timeslots are the colours to be assigned to each event. Events are connected by edges if they cannot be scheduled in the same timeslot. We call two events with this property *conflicting*.

*Observation 1.* In the UCTP-C, two events $E_i$ and $E_j$ are conflicting if at least one of the following conditions is verified:

1. $E_i$ and $E_j$ share one or more students;
2. $E_i$ and $E_j$ can be assigned only to a room which is the same for both the events.

We call then *degree* of an event $E_i$ the number of its conflicting events. Finding a feasible colouring to the graph $G$ associated to an instance is already an $\mathcal{NP}$-complete problem (Garey and Johnson, 1979) but it is not enough to create a feasible solution for the UCTP-C, since constraints H1 and H2 are not considered. Intuitively, the hardness of the underlying graph colouring instance provides an indication of the hardness of solving an UCTP-C instance. On our benchmark instances, one of the best approximate algorithms for graph colouring, the tabu search of Hertz and de Werra (1987), finds always a feasible colouring using much less than 40 colours, except on instances 3, 10, 12 and 7, where 40 colours must be used. The difference in terms of colours of considering only one or both the conditions of Observation 1 is relevant (in 13 instances this difference amounts to more than eight colours).

### 2.4. The evaluation of candidate algorithms in the competition

In the International Timetabling Competition, the participants were asked to submit the best solution found by their algorithms on each of the 20 instances. The winner was the participant

**Table 1** Statistics for the benchmark instances

| Instance identifier | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| No. of events | 400 | 400 | 400 | 400 | 350 | 350 | 350 | 400 | 440 | 400 | 400 | 400 | 400 | 350 | 350 | 440 | 350 | 400 | 400 | 350 |
| No. of students | 200 | 200 | 200 | 300 | 300 | 300 | 350 | 250 | 220 | 200 | 220 | 200 | 250 | 350 | 300 | 220 | 300 | 200 | 300 | 300 |
| No. of rooms | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 10 | 10 |
| Rooms/event | 1.96 | 1.92 | 3.42 | 2.45 | 1.78 | 3.59 | 2.87 | 2.93 | 2.58 | 3.49 | 2.06 | 1.96 | 2.43 | 3.08 | 2.19 | 3.17 | 1.11 | 1.75 | 3.94 | 3.43 |
| Events/student | 17.75 | 17.23 | 17.70 | 17.43 | 17.78 | 17.77 | 17.48 | 17.58 | 17.36 | 17.78 | 17.41 | 17.57 | 17.69 | 17.42 | 17.58 | 17.75 | 17.67 | 17.56 | 17.71 | 17.49 |
| Students/event | 8.88 | 8.62 | 8.85 | 13.07 | 15.24 | 15.23 | 17.48 | 10.99 | 8.68 | 8.89 | 9.58 | 8.79 | 11.05 | 17.42 | 15.07 | 8.88 | 15.15 | 8.78 | 13.28 | 14.99 |

that scored the best result across the 20 instances. More precisely, each participant $j$ was associated a penalty value $p^j$ computed as

$$p^j = \sum_{i \in I} \frac{f_i(a_i^j) - f_i(a_i^b)}{f_i(a_i^w) - f_i(a_i^b)}$$

where $I$ is the set of the 20 instances of the competition, $f_i$ is the evaluation function value for instance $i$, $a_i^j$ the assignment submitted by a participant $j$ for the instance $i$ and $a_i^b$ and $a_i^w$ are, respectively, the best and the worst assignment for the instance $i$ among all those submitted. In other terms, for each instance a participant was given a penalty value between 0 and 1, 0 if its assignment resulted to be the best for that instance and 1 if the assignment was the worst. The total penalty value was then obtained by the sum of the single values on the instances. The winner was the participant that scored the lowest final penalty value.

The maximum time available to solve an instance was defined by a benchmark code which had to be run on the same machine as the algorithm. The times produced by the benchmark code are reasonable times for desktop computers, for example, on a Pentium III with clock speed 700 MHz the time available was about 18 min.

## 3. A novel methodology for developing metaheuristics

In this section, we discuss some of the diverse possible choices that arise when applying metaheuristics to solve the UCTP-C. In this way, we bring to the attention the need for a methodology that could guide in this decision-making process. We show that the selection issue can be solved effectively by using an experimental methodology based on racing procedures. By a deeper analysis of our experience we gain few insights for the application of metaheuristics which are likely to be of more general validity.

### 3.1. Hybrid metaheuristic approaches

As part of our research within the Metaheuristics Network, basic versions of tabu search, simulated annealing, iterated local search, evolutionary algorithms and ant colony optimisation were implemented and compared on the UCTP-C. All metaheuristics used the same local search procedure described in Rossi-Doria et al. (2002). The instances for the comparison were created by the same random generator but were different from those of the competition. They were grouped into three classes according to their size. The results of the comparison were in part inconclusive. The performance of metaheuristics varied between instances of different sizes and with respect to the capability of solving hard or soft constraints. The main pattern found was that tabu search and iterated local search behaved better than other metaheuristics with respect to the capability of reaching feasibility but once a feasible assignment was found, simulated annealing appeared to be the best choice for minimising the violations of soft constraints. For an extensive description of this work and its results we refer to Rossi-Doria et al. (2003) and Chiarandini and Stützle (2002). It was clear, however, that no metaheuristic appeared particularly more appropriate than others for this problem.

It was then decided to develop a high performance algorithm for the UCTP-C and to participate at the International Timetabling Competition. To this end, all the metaheuristics of the previous comparison were left free to exploit at their best the characteristics of the underlying problem and new techniques and approaches were included in the analysis. Particular effort was put in the design of hybrid algorithms, as this appeared to be a promising direction from the previous results.

The following new algorithms were introduced in the analysis:

- An *ant colony optimisation* algorithm in its $\mathcal{MAX}$-$\mathcal{MIN}$ ant system variant (Socha, Sampels and Manfrin, 2003; Socha, 2003).
- An *evolutionary algorithm*, more precisely defined as heuristic memetic algorithm that uses construction heuristics for the initialisation of the population as well as for the recombination of individuals and local search after every generation. A heuristic mutation operator is also sometimes used in addition to a random mutation (Burke, Newall and Weare, 1996; Newall, 1999).
- An *iterated local search* that uses variable neighbourhood descent as local search, random moves as perturbations and annealing as acceptance criterion (Lourenço, Martin and Stützle, 2002).
- An *iterated greedy search* algorithm which is inspired by the methods for the graph colouring problem (Culberson, 1992); this algorithm tries to overcome local optima by destructing and reconstructing with construction heuristics a part or the whole assignment.

Hybrid algorithms were developed based on the following components.

**Construction heuristics:** More than 60 construction heuristics were developed. They were inspired by the work on graph colouring and on timetabling applications (Burke, Elliman and Weare, 1995; Carter, Laporte and Lee, 1996; Newall, 1999).

**Local searches:** Seven different neighbourhood structures and various neighbourhood examination schemes were considered giving rise to several local search procedures. The best alternatives are described in detail in Section 4.5 while other attempts are mentioned in Rossi-Doria et al. (2002) or are omitted.

**Metaheuristics:** The following general-purpose guidance criteria appeared to guide successfully the local search beyond local optima assignments.

- *variable neighbourhood descent*, consisting in a sequence of local search procedures in different neighbourhood structures; the sequence is iteratively repeated until no further improvement is found in any neighbourhood;
- *tabu search*, based on the idea of accepting also worsening solutions during local search but avoiding to cycle through already visited assignments;
- *simulated annealing*, based on the idea of accepting worsening solutions encountered in the local search process according to a probabilistic acceptance criterion.

Whether used alone, as in the case of the new algorithms introduced in this phase, or combined together, as in the case of hybridisations, these algorithms require to be configured and tuned on the class of instances at hand. This process can be regarded to as an algorithm engineering process. If many possible combinations of components and sets of parameters should be considered and all of them be tested exhaustively, finding the best configuration would be a task that goes quickly beyond the available computational power. An ingenious experimental methodology is therefore needed in order to avoid the pitfall of relying only on the intuition of the developer. Based on statistical considerations on the class of instances to solve, the methodology we propose reduces the number of experiments to run and is particularly well suited for the engineering of metaheuristics.

### 3.2. The experimental methodology

The experimental methodology that we to choose the best algorithmic configuration for a set of instances is based on the *F-RACE* method introduced by Birattari et al. (2002). In

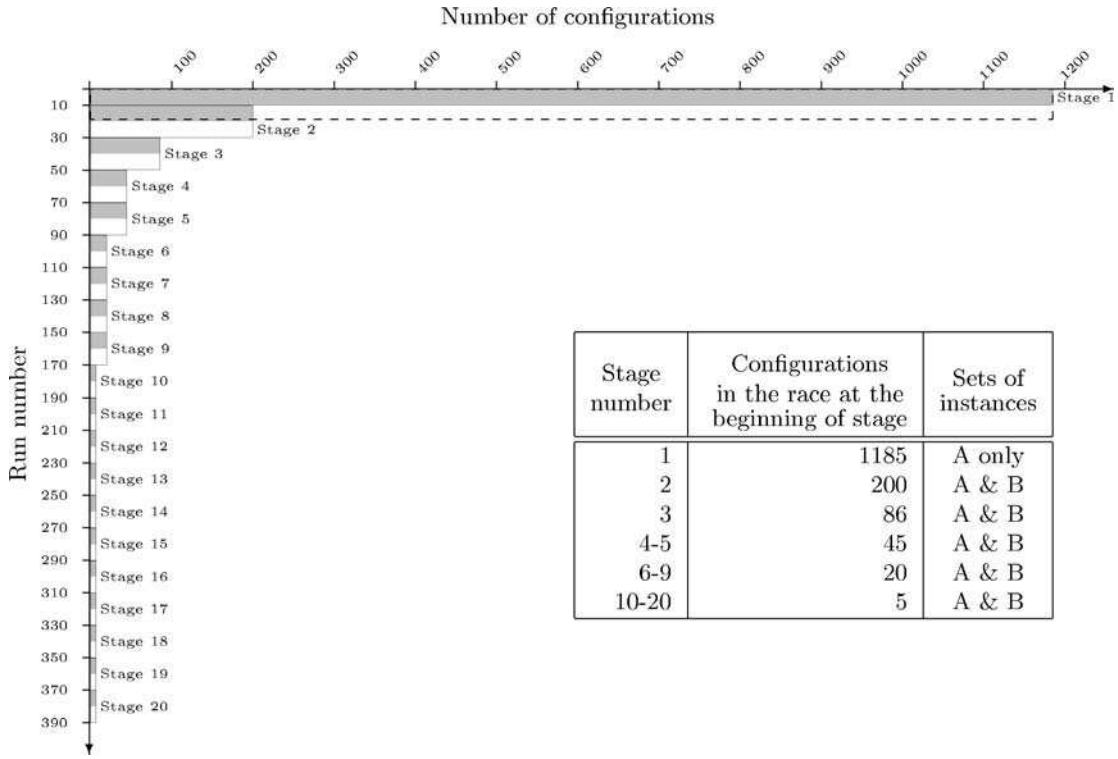
this *racing approach*, an initial set of candidate algorithms is sequentially evaluated on the available instances. During the race, poor candidates are discarded as soon as statistically sufficient evidence is gathered against them. The elimination of inferior candidates speeds up the procedure (since less candidate algorithms need to be tested on the available instances) and allows a more thorough evaluation of the most promising ones.

Our final goal was to submit the best metaheuristic configuration to the International Timetabling Competition having about 1 month of time available for running the experiments. In fact, not any arbitrary combination of the algorithm components mentioned above seemed to yield a reasonable approach. Results of the previous study gave some indication towards some specific combinations. Accordingly, we defined a set of 879 initial candidate algorithms. For matching the rules and the characteristics of the competition a slightly modified version of the race was adopted. In this version, an elementary experimental unit is a *stage*. A stage consists in testing all surviving candidates on the available instances, with a single run on each instance. At the end of each stage a decision is made on which candidate configuration should be discarded. Contrary to F-RACE, which is a completely automatic procedure where decisions are solely based on the outcome of the *Friedman two-way analysis of variance by ranks* (Conover, 1999), the method adopted in this work is of a more interactive nature and various statistical tests were used at each stage for supporting human decisions on the selection of the good candidates. In particular, the tests used were the Friedman test, the Wilcoxon matched pairs signed ranks test and the paired $t$-test with level of confidence adjusted for multiple comparison by Bonferroni's and Holm's methods (Sheskin, 2000). The general policy was to become more conservative as the number of candidates became smaller. For the final two best configurations we also used the exact binomial test considering only the best results attained on the instances, thus being somehow more compliant with the evaluation method of the competition. More details on the race and algorithmic outlines of some of its variants can be found in Birattari (2004). An implementation, to be used under R, the free software suite for statistical computing, has also been made available in the public domain (Birattari, 2003).[4]

In the competition 10 instances were made available since the beginning (Set A), while other 10 were published only 15 days before the deadline for the submission of the algorithms (Set B). The first phase of the race consisted of one stage and was conducted only on the Set A of instances. Therefore, each of the candidates performed at least 10 runs, one on each of the 10 instances, before being discarded. When the Set B of instances was released, we were ready to start the second stage of our race with a number of surviving candidates that had already dropped to one sixth of the initial number. From then on, each stage of the race consisted of running once each surviving candidate on each of the 20 available instances. Results of runs on previous stages were aggregated with new results to increase statistical significance.

The variant of the race here adopted allowed at the end of each stage to insert new candidates combining features of already tested candidates that showed good performance. Newly entered candidates were run on each instance for a number of times equal to the current number of runs before taking part in the test for survival. In this way, the total number of candidates considered by the selection procedure reached a final value of 1185. This interactive feature is a novelty in experimental design and we believe it can be very useful in the field of algorithm engineering. Indeed, it was from one of the refinements made possible by the analysis of ongoing results that we assembled the algorithm that finally won the race.

[4] M. Birattari. "CRAN—Package race". <http://cran.r-project.org/src/contrib/Descriptions/race.html> (22 October 2004)




| Stage number | Configurations in the race at the beginning of stage | Sets of instances |
|---|---|---|
| 1 | 1185 | A only |
| 2 | 200 | A & B |
| 3 | 86 | A & B |
| 4-5 | 45 | A & B |
| 6-9 | 20 | A & B |
| 10-20 | 5 | A & B |

**Fig. 1** Graphical visualisation of the race including all configurations which were finally tested. The *y*-axis reports the number of runs collected for each individual configuration. Runs on the Set A of instances are represented in *gray*, while runs on the Set B are represented in *white*. Stage 1 was an exception in the race because it considered only instances of Set A. The *dashed box* is relative to a brute-force approach which occupies the same computational surface as the race. The table provides in a numerical form the same information conveyed by the graph

Figure 1 reports graphically the details of the race. The total surface covered by the gray and white bars represents the total amount of computation required by the racing algorithm. The whole procedure took about 1 month of alternating phases of computation, analysis and human decision and involved up to a maximum of 30 PC running Linux, with CPU clock speed in the range between 300 and 1800 MHz, each using its own benchmark time given by the competition benchmark code (running the race on a single average machine would have required 270 days, more than 9 months). In the figure, the surface of the race is equal to the one covered by the dashed box: This indicates that a brute-force selection method in the same time would have performed only a single run of each of the initial candidates on 18 instances. The race permitted, instead, to collect 390 runs for each of the five best configurations that reached the last stage, corresponding to 19 runs on each instance of the Set B and 20 runs on each instance of the Set A. Running all the 1185 configurations 19 times on all the 20 instances would have required about 15 years on a single machine but, as shown by Birattari et al. (2002), would not have produced a statistically significant better outcome than the one produced by the race.

At the end of the race, we had still time to run the selected candidate 77 more times on each of the 20 available instances (Set A + Set B) and to choose the best solutions to submit to the competition out of 96 and 97 runs per instance.

### 3.3. General indications from the race

Beside configuring the winning algorithm, the racing approach can also be used to get insights of more general significance, as suggested by den Besten (2004). Candidates can indeed be grouped by common features, as for example, candidates which use simulated annealing and whose only difference is in the values of parameters, or candidates based on ant colony optimisation but with different population size. For each candidate it is then recorded when it was discarded and its survival rate computed in correspondence of each stage of the race. The groups, whose survival rate drops to zero after few steps of the race, use very likely a wrong approach to solve the problem. In our case the following observations were possible.

- All metaheuristics require to be hybridised with local search in order to achieve good results. Pure ant colony optimisation and genetic algorithms were indeed discarded very early if they did not use local search on at least the best of their individuals.
- Concerning local search, the local search scheme we describe in Section 4.5 is preferable to the one proposed by Rossi-Doria et al. (2002) while the acceptance of a certain number of non-worsening moves in all local search procedures contributes to improve the final solution quality.
- The use of variable neighbourhood descent strongly enhances local search and its presence was common in all candidates with high survival rate.
- Concerning the general solution approach, the race indicated quite clearly that keeping hard and soft constraints separated, solving first the hard constraints and then optimising the soft is preferable to weighting all the constraints together in a unique evaluation function and optimise them at once.
- Configurations which alternate local search with other heuristic mechanisms, like perturbations in iterated local search, reconstruction in iterated greedy or recombination in evolutionary algorithm, work better if the feasibility is not broken during these passages. In general, indeed, reproducing feasibility is very costly in terms of soft constraints. Since no way for doing this in iterated greedy was found, its survival rate remained very low.
- Tabu search appeared not suitable for optimising the soft constraints and maintained a low survival rate in spite of several attempts to reinsert it in the race with improved features. Its

main drawback is the need of exploring exhaustively a neighbourhood which is too large. No effective way to restrict considerably the neighbourhood was found.
- Population-based metaheuristics do not perform better than the single-solution-based. A possible explanation is the fact that approaches which work on single solution are continuously improving the solution quality even close to the time limit. Handling more than one solution has a cost in terms of time which is not paid off in terms of quality. Ant colony optimisation with a powerful local search maintained, however, the highest survival rate together with simulated annealing until the last stage.
- Finally, less strongly, it was indicated that the choice of an initial solution from a set of solutions constructed by different construction heuristics is better than generating a single initial solution. The difference between these two configurations arose at the last stage.

In addition to this, other indications were gathered by some incomplete candidate runs with the only purpose of understanding more detailed information. For example, candidates made solely of variable neighbourhood descent were used to understand the best sequential order of the neighbourhoods and whether repetitions without improvements of variable neighbourhood descent were profitable. Single local searches were also used as candidates to distinguish between different examination strategies (i.e. best improvement against first improvement).

In general, the use of problem-specific knowledge was confirmed to be chiefly important. Insights to restrict the neighbourhood of local search to only promising solutions, recognition of sub-problems to which fast exact algorithms can be applied, ingenious ways to perturb or construct a solution, known theoretical results concerning the problem or sub-problems are all elements which provide improvements over basic implementations. Finally, the fast evaluation of neighbour solutions by computing only local changes and the optimisation of the code have a determinant role on the performance of all metaheuristics.

## 4. The hybrid algorithm

We describe the hybrid algorithm, first, by giving a top-level view and telling about the management of data, then, going through the details of all the sub-procedures involved. Two phases are recognisable in the algorithm: one to solve the hard constraints and another to minimise the number of soft constraint violations.

### 4.1. High-level procedure

The algorithm creates several feasible assignments, selects the most promising of them and improves it until the maximum computation time is reached. Figure 2 schematically reproduces this procedure that can be summarised in three steps: first a group of initial assignments is constructed by Build_Assignment; next each assignment is made feasible by the Hard_Constraint_Solver and assessed by a fast local search Look_Ahead_Local_Search. Finally, the best assignment is selected (Select_Best_Assignment) and further improved by means of procedure Soft_Constraint_Optimiser.

Build_Assignment constructs assignments according to a given construction heuristic $h_i \in \mathcal{H}$, where $\mathcal{H}$ is the set of 60 different construction heuristics. Preliminary experiments showed that the initial assignment produced by such heuristics is never feasible and Hard_Constraint_Solver has to be applied. The two sub-procedures Hard_Constraint_Solver and Soft_Constraint_Optimiser are two local search procedures that deal with hard and soft constraints, respectively. When trying

```
Function UCTP-C-Solver(I_k);                          /* I_k ∈ I is an instance of the competition */
Preprocessing(I_k);
Â → ∅;
for i = 1 to |H|;                                     /* H is the set of construction heuristics */
do
    a_i ← Build_Assignment(h_i) ;                     /* h_i ∈ H */
    a_i ← Hard_Constraint_Solver(a_i);
    a_i ← Look_Ahead_Local_Search(a_i);
    Â → Â ∪ {a_i};
end
a_best ← Select_Best_Assignment(Â);
a_best ← Soft_Constraint_Optimiser(a_best);
return a_best
```

**Fig. 2** Pseudo-code for the UCTP-C solver

to find a feasible solution, soft constraints are not taken into account, while when minimising the number of soft constraint violations, it is not allowed to break any hard constraint.

### 4.2. Data management

Some data provided by the instances can be processed at the beginning of the algorithm to obtain more synthetic information. In Preprocessing, the sizes and features of the rooms, $S^R$ and $F^R$, are used to determine which rooms are suitable for each event on the basis of $L^E$ and $F^E$. In addition to this, we use these data to create a matrix representation of the graph $G$ determined by the events that cannot be scheduled in the same timeslot according to Observation 1. We also maintain some redundant data structures which are useful for speeding up the neighbourhood exploration in the local search. The overall data are as follows:

- A matrix of size $|E| \times |R|$ with each entry indicating whether a room is feasible for an event.
- A list for each room with the events that can take place in it.
- A matrix of size $|E| \times |E|$ representing the adjacency matrix of the graph $G$.
- A list for each event with the students that attend it.
- A matrix of size $|S| \times |E|$ with each entry indicating if a student has to attend an event.

The Standard Template Library of C++ is used for defining and handling these data structures.

### 4.3. The assignment representation

An assignment of events to rooms and timeslots can be defined by a rectangular matrix $X_{|R|\times|T|}$, such that rows represent rooms and columns represent timeslots, where $|R|$ and $|T|$ are the number of rooms and timeslots in the instance. A number $k \in \{-1, 0, \ldots, |E|\}$, corresponding to one of the $|E|$ events or to none of them $(-1)$, is assigned to each cell of the table. If $X_{i,j} = k$, event $E_k$ is assigned to room $R_i$ in timeslot $T_j$, if $X_{i,j} = -1$, the room $R_i$ in timeslot $T_j$ is free. With this representation, we assure that there will be no more than one event in each room at any timeslot, which implies that the constraint H1 is always satisfied. An example of assignment is given in Fig. 3.

The room assignment can be done by some local search together with the timeslot assignment or by an exact algorithm once events have been assigned to a timeslot. In this latter case, the problem to be solved is a bipartite matching (Papadimitriou and Steiglitz, 1982). Bipartite

| Rooms \ Timeslots | $T_1$ | $T_2$ | | $T_i$ | | $T_j$ | | $T_{45}$ |
|---|---|---|---|---|---|---|---|---|
| $R_1$ | $-1$ | $E_4$ | $\cdots$ | $\mathbf{E_{10}}$ | $\cdots$ | $E_{14}$ | $\cdots$ | $-1$ |
| $R_2$ | $E_1$ | $E_5$ | $\cdots$ | $E_{11}$ | $\cdots$ | $\mathbf{E_{15}}$ | $\cdots$ | $-1$ |
| $R_3$ | $E_2$ | $E_6$ | $\cdots$ | $\mathbf{E_{12}}$ | $\cdots$ | $-1$ | $\cdots$ | $-1$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | | $\vdots$ | | $\vdots$ | |
| $R_r$ | $E_3$ | $E_7$ | $\cdots$ | $E_{13}$ | | $E_{16}$ | $\cdots$ | $-1$ |

**Fig. 3** The assignment matrix representation. Events are assigned to rooms and timeslots; for example, event $E_1$ is assigned to room $R_2$ in timeslot $T_1$, while room $R_1$ has no event assigned in the same timeslot. Bold face is used for emphasising the events used in the example to describe Kempe chain interchanges (see text).

matching can be treated as a maximum flow problem and it can be solved in $\mathcal{O}(\sqrt{|V|}|C|)$, where $V$ is the union of the set of events assigned to the timeslot and the set of rooms feasible for these events and $C$ is the set of possible connections between the events and the rooms at hand (Rossi-Doria et al., 2002; Sedgewick, 1988). We chose to use instead a breadth first search augmenting paths algorithm which runs in $\mathcal{O}(|V||C|)$. Despite its higher complexity, this algorithm was considerably faster on the small matching instances that we had to solve for the UCTP-C (from Table 1, the number of rooms involved in the matching never exceeds 11 and rooms/event is always a small value). The analysis of this algorithm as well as its implementation is due to Setubal (1996) .

### 4.4. Creation of initial assignments

For the creation of the initial assignments construction heuristics are used which are mainly taken from a previous work of Rossi-Doria and Paechter (2003). Assignments are built by a sequential algorithm that inserts events into the assignment matrix one at a time. At each step, first a still unscheduled event is selected and then a pair timeslot–room is decided. The process can be presented as follows:

*Step 1.* Initialise the set $\hat{E}$ of all unscheduled events with $\hat{E} = E$.
*Step 2.* Choose an event $E_i \in \hat{E}$ according to a heuristic.
*Step 3.* Let $\bar{X}$ be the set of all positions for $E_i$ in the assignment matrix in which the number of violations of constraints H2 and H3 is minimised.
*Step 4.* Let $\bar{X} \subseteq \hat{X}$ be the subset of positions of $\hat{X}$ in which the cost of violations of the soft constraints $\Sigma$ is minimised.
*Step 5.* Choose an assignment for $E_i$ in $\bar{X}$ according to a heuristic.
*Step 6.* Remove $E_i$ from $\hat{E}$ and repeat from step 2 until $\hat{E}$ is empty.

Two heuristic rules are used, one to choose which event to insert next in the timetable at Step 2 and one to choose the position in the matrix for the selected event at Step 5. We considered six heuristic rules to choose the event and 10 heuristic rules to choose the position in the matrix, that is, the pair timeslot–room. The different heuristic rules at Step 2 choose:

– the event with the maximal number of conflicting events;
– the event with the maximal number of conflicting events weighted by the number of students shared;
– the event with the maximal number of students and the maximal number of conflicting events;
– the event with the maximal number of possible rooms and the maximal number of conflicting events;

– the event with the maximal number of required features and the maximal number of conflicting events;
– the event with rooms suitable for the maximal number of other events and the maximal number of conflicting events.

Unresolved ties are broken by label order.

The set $\bar{X}$ is implemented as an ordered list of pairs room–timeslot which represent positions in the assignment matrix $X$ where violations of the constraints H2, H3 and $\Sigma$ are minimised; higher priority is given to H2 and H3. We note that $\bar{X}$ is not empty, otherwise the instance is unsolvable because it has more events than places available. The list is ordered column-wise with respect to the assignment matrix (alternatively it could be ordered randomly). The following heuristic rules are used to select the pair room–timeslot from $\bar{X}$:

– The smallest possible room and ties broken in favour of a timeslot with the largest number of events already assigned to it.
– The least used room and ties broken in favour of the timeslot with the lowest label number.
– The latest timeslot in the last day and ties broken in favour of the room suitable for the least events.
– The room suitable for least events and ties broken in favour of the latest timeslot.
– The room suitable for least events and ties broken in favour of the timeslot with the lowest label number.
– The timeslot among those earliest in the day that has most parallel classes and ties broken in favour of the room with the lowest label number.
– The timeslot according to a pre-established timeslot order that tries to keep difficult events spread out in the timetable and not in adjacent timeslots (three different orders are attempted, derived from Terashima-Marín, Ross and Valenzuela-Rendón (1999), giving rise to three different heuristics) and ties broken in favour of the room with the lowest label number.
– The pair with the lowest label number in the list.

All 60 possible combinations of these heuristic rules were implemented since preliminary experiments indicated that none of them was clearly dominated (see further discussion in Section 5). Other heuristics for timetabling are described in Carter, Laporte and Lee (1996) and Terashima-Marín, Ross and Valenzuela-Rendón (1999).

### 4.5. The local search

Once a complete assignment has been created, it is improved by means of local search and enhancements thereof. We describe the basic components.

#### *4.5.1. Neighbourhood structures*

Different neighbourhood structures are used when dealing with hard and soft constraints. When only hard constraints are considered, we use two neighbourhoods.

1. The neighbourhood $\mathcal{N}_1(a)$ is defined as the set of assignments obtained from $a$ by moving a single event to a different room and timeslot. More formally, given $a$ with an event $E_k$ assigned to $R_i$ in $T_j$ and a free cell $X_{l,m}$ in the assignment matrix, a neighbour assignment $a'$ is obtained from $a$ by setting $X_{i,j} = -1$ (i.e. making cell $X_{i,j}$ free), $X_{l,m} = E_k$, and all other positions left unchanged.
2. The neighbourhood $\mathcal{N}_2(a)$ is defined as the set of assignments obtained from $a$ by swapping timeslots and rooms of two events. More formally, given $a$ with two events $E_h$ and $E_k$,

assigned respectively, to $R_i$ in $T_j$ and to $R_l$ in $T_m$, a neighbour assignment $a'$ is obtained from $a$ with $X_{i,j} = E_k$, $X_{l,m} = E_h$, and all other positions left unchanged.

In practice, the local search in $\mathcal{N}_1$ considers only assignments obtained by moving an event involved in at least one hard constraint violation, while the local search in $\mathcal{N}_2$ considers only assignments obtained by swapping pairs of events of which at least one causes a hard constraint violation.

If soft constraints violations are considered in the local search, we define the two neighbourhoods $\mathcal{N}_1' \subseteq \mathcal{N}_1$ and $\mathcal{N}_2' \subseteq \mathcal{N}_2$, obtained by considering only neighbours that do not introduce violations of the hard constraints. In this case, we also use two further neighbourhoods that maintain the feasibility of an assignment.

3. The neighbourhood $\mathcal{N}_3(a)$ is defined as the set of assignments obtained from $a$ by swapping all events assigned to two timeslots. Formally, given two timeslots $T_i$ and $T_j$ of an assignment $a$, a neighbour assignment $a'$ is the assignment $a$ with the two columns of the assignment matrix $X_{.,i}$ and $X_{.,j}$ swapped and all other positions left unchanged. Clearly, no hard constraint is affected by this change and feasibility is preserved.
4. The neighbourhood $\mathcal{N}_4(a)$ is defined as the set of assignments obtained from $a$ by a Kempe chain interchange (Morgenstern and Shapiro, 1990; Thompson and Dowsland, 1998). Focusing only on events and timeslots, an assignment can be represented as a colouring of the graph $G$ (see Section 2.3). A feasible colouring is a partition of the graph in independent sets. A *Kempe chain* $K$ is a set of vertices that form a maximal connected component in the subgraph $S$ of $G$ induced by the vertices that belong to two different independent sets $I_i$ and $I_j$, $i \neq j$. A Kempe chain *interchange* produces a new feasible partition by swapping the colours assigned to the vertices belonging to $K$, i.e. replacing $I_i$ with $(I_i \setminus K) \cup (I_j \cap K)$ and $I_j$ with $(I_j \setminus K) \cup (I_i \cap K)$. An example of Kempe chain interchange on the UCTP-C is given in Fig. 3. Focusing on the timeslots $I_i$ and $I_j$, and assuming that event $E_{15}$ is conflicting with both events $E_{10}$ and $E_{12}$, the Kempe chain interchange of $K = \{E_{15}, E_{10}, E_{12}\}$ brings $E_{15}$ in timeslot $T_i$ and $E_{10}$ and $E_{12}$ in timeslot $I_j$. After a Kempe chain interchange, rooms are re-assigned by the matching algorithm. If no feasible matching of rooms can be found, the current Kempe chain interchange would break feasibility and it is discarded. The case $K = I_i \cup I_j$ is not considered, because the corresponding interchange is already accounted by neighbourhood $\mathcal{N}_3$.

Finally, we define the union operator $\mathcal{N}_i(a) \cup \mathcal{N}_j(a)$ between neighbourhood structures to indicate that a neighbour assignment $a'$ of $a$ belongs either to $\mathcal{N}_i(a)$ or to $\mathcal{N}_j(a)$.

#### *4.5.2. Neighbourhood examination scheme and evaluation function*

In all cases, the local search is a *stochastic first improvement local search* in which the examination of the neighbourhood is randomised and the first non-worsening neighbour is accepted. The passage from an assignment $a$ to an accepted neighbour is a local search *move* and moves to neighbours that do not change the evaluation function value of the current assignment are called side walk moves. Computational experience shows that accepting *side walk moves* is profitable in the neighbourhoods $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_1'$ and $\mathcal{N}_2'$ while it is not in $\mathcal{N}_3$ and in $\mathcal{N}_4$. In particular, in $\mathcal{N}_4$ are not appealing because they slow down significantly the search. Note that, in graph colouring, neighbours with equal quality often abound, being easy that more than one colour assigned to a vertex generate the same number of conflicts. In the UCTP-C, the presence of different criteria for evaluating a solution decreases the chances for neighbours to have the same quality, but

chances remain, however, high. For example, the swap of two events assigned to last timeslots of the days is very likely to leave the quality of the assignment unchanged.

In the neighbourhood structures $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_1'$ and $\mathcal{N}_2'$, a list of randomly ordered events is created. In turn, for each event in the list all possible positions in the assignment matrix are tried and the first non-worsening move found in this process is accepted. The search continues with the next event in the list. When the whole list of events has been scanned and no improvement has been found, the procedure ends and the current assignment is recognised as a *local optimum* (side walk moves are not counted as improvements). In $\mathcal{N}_1 \cup \mathcal{N}_2$ and $\mathcal{N}_1' \cup \mathcal{N}_2'$, for each event in turn all timeslots are considered and a move in the first neighbourhood is attempted before a move in the second neighbourhood. Finally, in the structures $\mathcal{N}_3$ and $\mathcal{N}_4$ the examination is done considering all possible pairs of timeslots in random order.

The evaluation function depends on whether the violations of H2 and H3 or the violations of $\Sigma$ are minimised. In the latter case, the evaluation function to minimise is the one defined in Section 2.2. In the former case, we define $g(a)$ for the hard constraints H2 and H3 as follows:

$$g(a) = g_2(a) + g_3(a)$$

where, $g_2(a)$ is the number of events which are assigned to a non-suitable room and $g_3(a)$ is the number of events sharing students in the same timeslot.

In any case, only local changes in the evaluation function are computed and, with neighbourhoods $\mathcal{N}_1'$ and $\mathcal{N}_2'$ moves that would introduce hard constraint violations are quickly recognised thanks to the pre-processed data and discarded.

#### *4.5.3. Comment*

An important issue for local search is the connectivity of the search space, i.e. the existence of a path between any two solutions in the search space. In highly constrained problems, areas where feasible solutions are located may be disconnected. This is a strong drawback for a local search that tries to minimise soft constraints while maintaining feasibility, because the search process could remain trapped in non-promising regions. Given the multiplicity of the neighbourhoods defined earlier and the instances of the problem at hand, this seems not to be our case. Indeed, since a perfect solution always exists, there are always free slots in the matrix $X$ (at least those in the last timeslot of the days because of constraint S1) that can be used to move from one solution to another without breaking hard constraints. The only hard constraint that could be violated is assigning two conflicting events to the same timeslot, but this appears unlikely to be the only available possibility. The landscape appears, therefore, connected, even if it may be not always possible to walk through the path of minimum distance.

### 4.6. Hard constraint solver

Figure 4 sketches the hard constraint solver. First, only Local_Search_in_$\mathcal{N}_1 \cup \mathcal{N}_2$ is applied. Once a local optimum has been reached, the local search is run again, since, given that side walk moves are accepted, it is possible that after some moves new regions for improvements are found. If after 5 runs of local search a feasible solution is not found, a second phase begins in which local search is alternated with tabu search.Tabu_Search_in_$\mathcal{N}_1$ implements a best improvement strategy and ends after a number of iterations in which no improvement in the evaluation function is found. The local search at the beginning provides a rapid descent towards assignments that violate only few hard constraints and therefore exhibit a neighbourhood which may be effectively

```
Function Hard_Constraint_Solver(I_k, a) ;      /* I_k ∈ I is an instance of the competition, a ∈ A */
loops ← 0;
while a is infeasible and time limit not exceeded do
    a ← Local_Search_in_N_1 ∪ N_2(a);
    if a is infeasible and loops > 5 and time limit not exceeded then
        a ← Tabu_Search_in_N_1(a);
    end
    loops ← loops + 1;
end
return a
```

**Fig. 4** Pseudo-code for the hard constraint solver

used with tabu search. The number of repetitions of local search was tuned empirically on the instances of the International Timetabling Competition.

More in detail, Tabu_Search_in_$\mathcal{N}_1$ is inspired by a successful tabu search application to graph colouring. A move is forbidden if it tries to place an event into a timeslot to which it was assigned less than $tl$ steps before. The tabu length is $tl = \text{ran}(10) + \delta \cdot n_c(a)$ where ran(10) is a random number in $\{0, \ldots, 10\}$, $n_c(a)$ is the number of events in the current assignment involved in at least one hard constraint violation and $\delta$ is a parameter. An aspiration criterion accepts a tabu move if it improves the best current assignment. The procedure ends, when the best assignment is not improved for a given number of steps. We fixed this number to $0.4 \cdot |T| \cdot |E| \cdot |R|$ and $\delta$ to 0.6.

After a feasible solution has been produced, its quality with respect to the soft constraints is assessed by means of Look_Ahead_Local_Search, which consists in a single repetition of local search in, successively, $\mathcal{N}'_1$ and $\mathcal{N}'_2$.

### 4.7. Soft constraint optimiser

An outline of the soft constraint optimiser is reported in Fig. 5. It consists of two parts. First, variable neighbourhood descent is applied in which several local search operators (Local_Search) in different neighbourhoods are run until none of them can improve further the assignment. Next, simulated annealing (Simulated_Annealing_in_$\mathcal{N}'_1 \cup \mathcal{N}'_2$_withMatching) is applied until the time available expires. Usually, simulated annealing gets good performance when long computation time is allowed. Here, we start simulated annealing from a good solution rather than from a

```
Function Soft_Constraint_Optimiser(I_k, a) ;   /* I_k ∈ I is an instance of the competition, a ∈ A */
repeat
    a ← Local_Search_in_N'_1(a);
    a ← Local_Search_in_N'_1 ∪ N'_2(a);
    a ← Local_Search_in_N'_1_withMatching(a);
    a ← Local_Search_in_N'_1 ∪ N'_2_withMatching(a);
    a ← Local_Search_in_N_3(a);
    a ← Local_Search_in_N_4(a);
until no improvement found ;
a_best ← Simulated_Annealing_in_N'_1 ∪ N'_2_withMatching(a);
return a_best
```

**Fig. 5** Pseudo-code for the soft constraint optimiser

random one in order to get quicker to good quality assignments. This choice was supported by experimental results.

In this phase, extensions of the neighbourhoods $\mathcal{N}_1'$ and $\mathcal{N}_2'$ are considered that use the matching algorithm for the insertion of selected events in a timeslot. This mechanism increases the chances of finding a place for an event in a specific timeslot since all rooms in that timeslot are reassigned possibly making available new feasible places that were previously occupied. If for a timeslot no more rooms are available, the move is discarded.

In simulated annealing (SA), a move is accepted according to the following probability distribution, dependent on the virtual temperature $\tau$:

$$p_{\text{accept}}(\tau, a, a') = \begin{cases} 1 & \text{if } f(a') \leq f(a) \\ \exp\left( - \dfrac{f(a') - f(a)}{\tau} \right) & \text{otherwise} \end{cases}$$

where $a$ is the current assignment and $a'$ is a neighbour of $a$. The temperature parameter $\tau$, which controls the acceptance probability, is allowed to vary over the course of the search process.

Our SA implementation uses as termination criterion the time limit imposed by the competition. The main components of SA are implemented as follows:

*Neighbourhood examination scheme:* For each event in a randomly ordered list, in turn, all timeslots are considered. In a first attempt, the event is assigned to the new timeslot and the matching algorithm applied to reschedule the assignment of rooms in that timeslot (neighbourhood $\mathcal{N}_1'$ with matching of rooms). If no feasible assignment of rooms can be found, the move is discarded, otherwise, it is accepted according to the SA probabilistic criterion. If the move is not accepted, swaps with all events in the timeslot are tried. The matching algorithm is applied at each attempt and only moves that lead to a feasible assignment are considered for acceptance (neighbourhood $\mathcal{N}_2'$ with matching of rooms).

*Initial temperature:* For determining the initial temperature $\tau_0$ a sample of 100 neighbours of the initial assignment in $\mathcal{N}_1' \cup \mathcal{N}_2'$ is considered. The average value of the variation of the evaluation function multiplied by a factor $\kappa$, $\kappa > 0$, is assigned to the initial temperature.

*Temperature schedule:* We used a non-monotonic temperature schedule determined by the interaction of two strategies: a standard geometric cooling and a temperature re-heating. In the standard geometric cooling the temperature $\tau_{n+1}$ is computed from $\tau_n$ as $\tau_{n+1} = \alpha \times \tau_n$ where $\alpha$, $0 < \alpha < 1$, is a parameter called *cooling rate*.

The number of iterations at each temperature, called *temperature length* $\tau l$, is kept proportional to the size of the neighbourhood, as suggested by the majority of SA implementations (Johnson and Aragon, 1991). We set, $\tau l = \rho \cdot |E| \cdot |R| \cdot |T|$, where $\rho$ is a parameter, and count as iteration each attempt to make a change in the assignment, independently by the fact that the change breaks feasibility and it is therefore refused, or that the change is accepted.

When the search appears to "stagnate", the temperature is increased by adding the initial temperature $\tau_0$ to the current temperature. This is done when no improvement is found for a number of steps given by $\eta \cdot \tau l$, where $\eta$ is a parameter. The assignment that we consider for testing improvements is the best since the last re-heating occurred.

For selecting the best combination of parameters $\alpha$, $\kappa$, $\rho$ and $\eta$ we used the race described in Section 3.2. The outcome was the setting: $\kappa = 0.15$, $\rho = 10$, $\alpha = 0.95$ and $\eta = 5$.

## 5. Analysis

The racing approach provides guarantees that the algorithm developed exhibits high performance and is essential in its components. In this section, we bring further external evidence to support this claim.

### 5.1. Benchmark comparisons

The final aim of our work was to compare our algorithm against those submitted to the International Timetabling Competition. According to the rules and the evaluation methodology of the competition presented in Section 2.4, our algorithm scored the lowest penalty value and ranked the best.

The evaluation procedure of the competition is influenced by the submissions of all participants. The official winner, the algorithm by Kostuch (2003), scored a penalty value clearly lower than all other competitors and therefore we give main priority to the comparison with that algorithm.

In Table 2, the first two rows report the best results for the two algorithms. Our algorithm obtains a better solution on 18 instances out of 20 and results significantly better, at a confidence level of 95%, according to the paired *t*-test, the randomisation test (Cohen, 1995), the exact binomial test and the paired Wilcoxon signed rank test (Sheskin, 2000).

In the table we also report the overall best results on the 20 instances among all the official submissions of the competition. These results are obtained basically by two algorithms, the algorithm of Kostuch and the algorithm of Burke et al. (2003), which ranked third in the competition. For a complete list of results and descriptions of the algorithms submitted we refer the reader to the official web site of the competition.

For the sake of fairness, we report that the official winner of the competition had much less runs available for the selection of the best submissions (P. Kostuch, priv. comm.). Yet, we interpret this fact as further evidence of the effectiveness of our experimental methodology. Indeed, with a brute-force approach it would have been possible to collect only one single run on 18 instances, while, had we wanted to collect at least few runs per each of the 20 instances, we should have reduced drastically the number of tested configurations. Instead, identifying the most promising algorithm out of 1185 provides a clear guarantee of performance, important above all with heuristics and metaheuristics whose selection is determined essentially by the fact of performing *empirically* well.

After the submission, we polished and optimised the code of the algorithm. We then collected 50 new runs of our algorithm on each instance. In Table 2, we present the best results attained, which are always the best even compared with the overall best results of the competition, while in Fig. 6 we show the box-plots of solution quality distributions. Recently Kostuch (2004) has also published new results for this set of instances. These new results are not anymore significantly different from ours. It is likely that both algorithms can be further improved; nevertheless, our main intent here is showing the effectiveness of a systematic design methodology under the circumstance in which there exists a limit on the time available for the development (in fact, a very realistic circumstance).

As far as the instances of the competition are concerned from the results in Fig. 6 two considerations arise: (i) some instances are easier than others (in particular instance 20 is the easiest followed by instances 6, 8, 7); (ii) the harder the instance the higher is the variance of the best evaluation function value found by our algorithm.

**Table 2** The first two rows report the official results of the International Timetabling Competition on the 20 instances relative to the algorithm of Kostuch and to ours. A value in bold face indicates that our algorithm wins. The third row reports the overall best results among the official submission of the competition and the fourth row reports our post-competition results out of 50 runs per instance attained by an optimised version of our algorithm

| Instance | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Official winner | 45 | 25 | 65 | 115 | 102 | 13 | 44 | 29 | 17 | 61 | 44 | 107 | 78 | 52 | 24 | 22 | 86 | 31 | 44 | 7 |
| Our subm. | 57 | 31 | **61** | **112** | **86** | **3** | **5** | **4** | **16** | **54** | **38** | **100** | **71** | **25** | **14** | **11** | **69** | **24** | **40** | **0** |
| Overall best | 45 | 25 | 65 | 115 | 77 | 6 | 12 | 29 | 17 | 61 | 44 | 79 | 78 | 36 | 24 | 22 | 79 | 31 | 44 | 0 |
| After optim. | 45 | **14** | **45** | **71** | **59** | **1** | **3** | **1** | **8** | **52** | **30** | **75** | **55** | **18** | **8** | **5** | **46** | **24** | **33** | 0 |

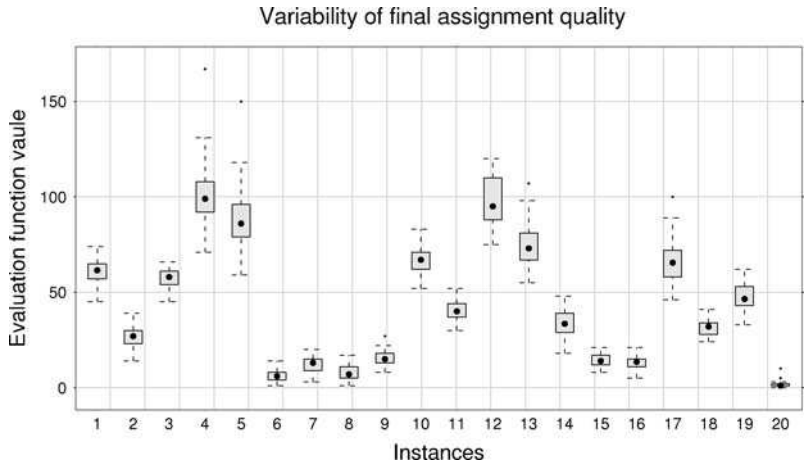



**Fig. 6** Box-plots that summarise the distributions of solution quality. The *central box* shows the data between the first and the third quartile, the median is represented by a *dot* in the *central box*. *Whiskers* extend to data points which are no more than 1.5 times the interquartile range away from the box. *Points* represent very extreme data

## 5.2. Contribution of algorithm components

The analysis that follows aims at determining whether the components of the hybrid algorithm are all needed and the impact of such components on the final performance.

A first relevant observation is obtained by profiling our code. Construction heuristics and local searches on hard and soft constraints, consume only 10% of the total time allowed for a run of the algorithm, while the remaining 90% is used by simulated annealing.

### *5.2.1. Construction heuristics*

To understand if some heuristics are more useful than others, we run the algorithm in Fig. 2 on the different instances until the procedure Select_Best_Assignment was completed and we recorded the number of times each construction heuristic provided the initial solution to produce $a_{\text{best}}$. If some heuristics were never selected we could discard them, while knowing which heuristics are more used could help us in exploiting features of the problem. All heuristics are deterministic but generate an assignment which is still infeasible and the application of the hard constraint solver together with the look ahead local search makes the selection of the heuristics a stochastic process. Hence, we replicated the observation 50 times per instance using different random seeds.

In Fig. 7, left, we report the histogram of the number of times each heuristic produced the best assignment. The Kolomogorov–Smirnov test (Conover, 1999) indicates that there is not enough significance to reject the hypothesis that the observed distribution is sampled from a discrete uniform distribution. Visual evidence of this fact is given in Fig. 7, right. This is an interesting result: Apparently there is no heuristic emerging over the others and each heuristic have the same probability of leading to $a_{\text{best}}$. The same analysis repeated on each single instance, confirms that, in all the instances except two, the distribution of selection frequency remains not significantly different from a uniform distribution.

However, the use of the construction heuristics is not ineffectual. This arises from the comparison of an algorithm that uses the 60 heuristics against an algorithm that generates 60 initial

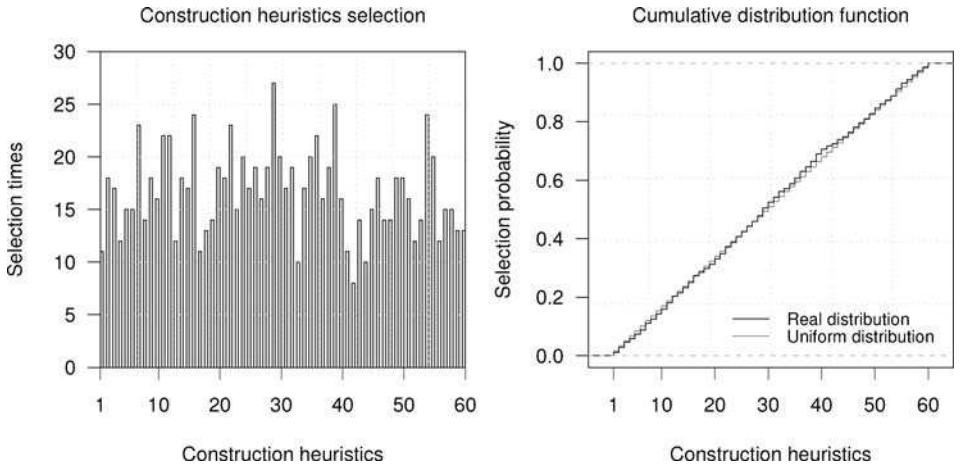



**Fig. 7** On the left, the histogram reports the number of times each heuristic led to the best feasible solution. On the right, a visual comparison of the observed distribution against a discrete uniform distribution in support of the insight that all heuristics are chosen with equal probability

solutions in a semi-random manner. The semi-random construction can be summarised as follows. First all events are assigned randomly to timeslots. Then an exact matching algorithm is applied to each timeslot to assign rooms to events. If in the timeslot there are more events than rooms or if it is not possible to schedule all events in a suitable room, some events do not receive a room. After the matching algorithm has been applied to all the timeslots, the events left without room are assigned to suitable rooms in different timeslots from the one they were assigned initially, where the number of violations is minimised. In case, priority between events is decided by label order. The comparison shows that the use of the 60 construction heuristics is to be preferred. A possible explanation of this fact is that with assignments which are closer to feasibility longer time is left to the soft constraint optimiser to produce better final solutions. The 60 construction heuristics are therefore replaceable only by heuristics which produce better assignments.

Finally, it was profitable also to choose among a set of initial solutions, rather than using a single initial solution. A set of 60 initial solutions, to which it corresponds the use of 1/10 of the total running time, resulted to be a good compromise between searching for good initial solutions and leaving enough time to the soft constraints optimiser.

### *5.2.2. Hard constraint solver*

Once a single initial assignment has been constructed by one of the construction heuristics, making the assignment feasible is not a hard task. To show this, we run an algorithm, consisting of one single construction heuristic (events assigned according to the maximum number of conflicting events and rooms and timeslots in label order) and the procedure Hard_Constraint_Solver, for 30 times on each instance, halting it when a feasible solution was reached. In Fig. 8, we represent the distribution of the time needed to reach a feasible assignment over all the instances through two statistics, the median and the maximum computation time. Considering the maximum time (the dashed line), we observe that for any of the instances, at most 15 s are needed to find a feasible solution. More likely, considering the median time (the solid line), a feasible solution

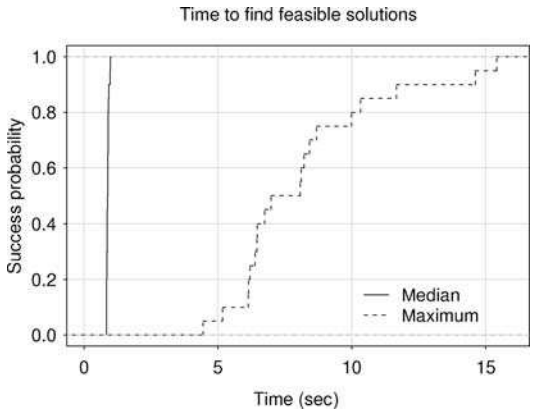



**Fig. 8** Empirical cumulative distribution function of the median (*solid line*) and maximum (*dashed line*) computation time needed for reaching feasibility. A point in the curve gives the empirically determined probability to find a feasible solution after the indicated time

for a given instance is produced already after only 1 s on a machine[5] where the total time limit is 1074 s.

### *5.2.3. Simulated annealing*

Table 3 shows the improvement in solution quality determined by SA. The improvement is relevant and without the SA phase results in the competition would have been very poor.

Profiling the behaviour of the temperature and the evaluation function value during one run of SA, we noted that the re-heating feature is used in average 2–3 times and contributes to improve the search, because in many runs the best solutions are obtained close to the time limit. In average, simulated annealing performs about $70 \times 10^4$ thousand iterations per second for a total of about $40 \times 10^7$ iterations. The cost of a single iteration, however, varies considerably depending on how soon a proposed change is refused or accepted.

## 5.3. Run-time distributions

For stochastic algorithms the time needed to produce a solution of a given quality is a random variable and is described by a probability distribution. This distribution can be empirically characterised. In particular, one can run the algorithm for a given number of times on a specific instance and record the time needed to find a solution of a fixed level of quality. The cumulative distribution associated with these observations is known as *run-time distribution* (Hoos and Stützle, 1998) and is used to describe the behaviour of the algorithm over time. In Fig. 9, we report the run-time distributions for different final solution qualities on instance 20, which was the easiest instance to solve. We observe that an assignment of cost at least 3 will always be reached in the given amount of time, while an optimal assignment will be found in about 30% of the cases. The main observation is that it can be useful to increase the time limit if one wants to attain better solutions because the distributions show a positive slope. As a consequence of this, we left our algorithm run longer also on instances 6, 7, 8 and, indeed, we reached optimal solutions. This fact also indicates that further improvements can be achieved by optimising the code.

Run-time distributions also allow other kinds of analysis. Of particular interest is the understanding of whether the algorithm 'stagnates' after a certain time, that is, if a restarting could

[5] Pentium III, 700 MHz cpu clock, 256 KB cache memory, and 512 MB RAM.

**Table 3** The contribution of simulated annealing in terms of evaluation function values. Reported are the median values on 96 trials per instance

| Instance | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Before SA | 199 | 150 | 208 | 385 | 388 | 286 | 101 | 140 | 120 | 193 | 210 | 228 | 260 | 263 | 229 | 115 | 356 | 156 | 273 | 157 |
| After SA | 77 | 42 | 78 | 162 | 120 | 14 | 18 | 20 | 31 | 82 | 58 | 143 | 109 | 49 | 5 | 24 | 121 | 46 | 68 | 4 |

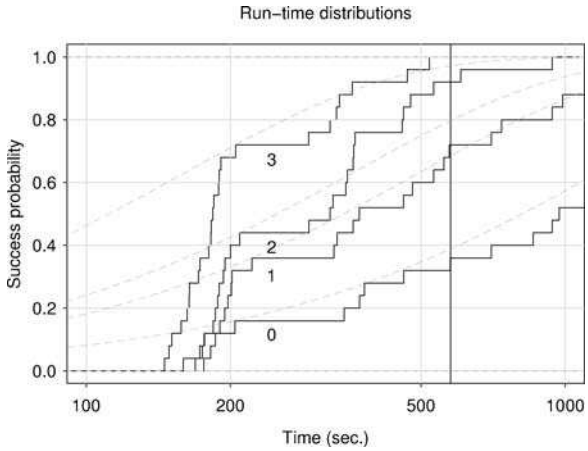



**Fig. 9** Run-time distributions for four different solution quality on instance 20. At each time a point in the curve represents the empirical probability to find a solution of the specified quality. Empirical data are taken from 25 runs. The *dotted lines* superimposed represent the exponential distributions while the *vertical line* is the total time limit on a machine with processor AMD Athlon, 1000 MHz cpu clock, 256 KB cache memory and 512 MB RAM. The graph is in semi-logarithmic scale

be more effective than leaving the algorithm run until the end. This is achieved by comparing the empirical distributions with exponential distributions (Hoos and Stützle, 1999). In Fig. 9, we observe that the empirical run-time distributions assume after a while the same steepness of an exponential distribution, thus indicating that restarting would not bring any improvement. The 'step-form' of the empirical distributions is due to the re-heating phase in SA, while differences at the beginning of the time are caused by the initialisation phase. We can conclude that, under the set of parameters used, our algorithm is well tuned and more likely to improve the solution if left running than if re-started.

### 5.4. Further comments

Combining the criteria into a single scalar value is not the only way to cope with multi-criteria optimisation. Burke, Bykov and Petrovic (2001) propose a weighted aggregation function which allows to control the degree of compensation and the importance for each criterion. However, in our case this approach is hardly practicable because the worst value for each criterion must be known but its computation is not straightforward. Another interesting alternative is the multi-phase or lexicographic approach in which one criterion at a time is optimised while constraints are imposed on the others. The drawback of this approach is the risk of making the search space disconnected and the landscape flat, which are often bad characteristics for local search methods.

Further research efforts were spent to find a model that relate the final solution quality with features of the instances. Such models, based on multiple linear regression, are useful diagnostic tools that academic departments can use to negotiate features or undertake other preventive measures. In our case, the features with significant influence are (i) average event degree, (ii) standard deviation of event degree, (iii) number of events, (iv) number of rooms (Kostuch and Socha, 2004; Chiarandini, 2005).

Finally, Arntzen and Løkketangen (2003) use a simple construction heuristic that alone is able to produce feasible assignments for the instances of the competition (Arntzen and Løkketangen, 2003). The adoption of this heuristic in our algorithm could make it simpler and improve its performance.

## 6. Conclusions

This work described a hybrid metaheuristic algorithm for solving the university course timetabling problem. The algorithm was tested on the UCTP-C, a standardisation of the problem which includes a subset of the most common constraints. The UCTP-C has been object of the International Timetabling Competition which gathered researcher from all over the world. The algorithm here discussed ranked the best according to the rules of the competition.

Most of the successful approaches in the competition (10 out of the 11 best ranked methods) had the common feature of being based on metaheuristic techniques and hybridisations. More specifically, they use different methods for producing feasible solutions but then, when soft constraints are to be minimised, metaheuristics are the choice. For example, similarly to our approach, also the official winner of the competition strongly relies on simulated annealing Kostuch (2004).

Our hybrid algorithm is based on a framework which consists in the successive application of construction heuristics, variable neighbourhood descent and simulated annealing. Simulated annealing can be then further alternated with variable neighbourhood descent. This framework is of general applicability. Recently, for example, one of the authors has been involved in a competition on a car sequencing problem, a well-different problem from timetabling, and the algorithm, engineered through a process independent from the work presented here, resulted using the same framework (Risler et al., 2004). In that competition the algorithm ranked in its category the best in the first phase and the third in the second final phase. This fact leads us to conclude that the framework is promising to achieve good results also on other timetabling problems such as examination timetabling and employee timetabling.

The contribution of this work goes, however, beyond the mere introduction of an effective algorithm. The most important and portable message is indeed the introduction and use of a systematic experimental methodology for the design and development of optimisation algorithms for complex combinatorial problems. The success of the hybrid algorithm is mainly due to this methodology which allowed the effective selection and assemblage of several construction heuristics, local searches and metaheuristics. Chiefly important, the selection is carried out with sound statistical testing. Specifically, a formalised procedure, the race, can be applied in which a set of candidate configurations is sequentially evaluated on available instances and poor candidates are discarded as soon as statistically significant evidence is gathered against them. This fast elimination of non-promising candidates has two main advantages: it speeds up the comparison and it focuses on promising candidates that can be better tuned. The decision upon elimination can be done automatically according to a suitable statistical test. Moreover, this work has shown that this process can be even conducted interactively. Ongoing results can be analysed and indications can be collected to assemble new stronger candidates to insert in the race. In this way the two phases of development and testing overlap and the time-to-market can be reduced. A post-competition analysis, emphasized the adequacy of this engineering process by showing that all the obtained components contribute positively to the final algorithm and are well tuned.

In the specific field of timetabling, the introduction of automated techniques for the configuration of solution algorithms is particularly important. The interest in this direction is demonstrated by other recent examples such as case-based reasoning systems (Burke et al., 2005) and hyper-heuristics to handle heuristics (Burke, Kendall and Soubeiga, 2003). The racing methodology appears robust and well suited to fulfil the need for flexibility typically presents in timetabling.

We conclude with a general remark on the use of metaheuristics in combinatorial optimisation. Rather than searching for a metaheuristic superior over the others, they can be regarded to as mechanisms to achieve specific goals at different stages or contexts of the search process. Thus, for example, local search can be used to get very fast improvements, tabu search to escape

from local optima when the neighbourhood is relatively small and the landscape is flat, variable neighbourhood descent to improve the performance of local search while maintaining its speed and simulated annealing when the neighbourhood size and the time available are large and fixed so that a correct tuning of parameters can be achieved. Though based on general concepts, the performance of these mechanisms is problem dependant and re-assemblage and re-configuration must occur on each new class of instances of a specific real-life problem. In this context the phase of algorithm engineering assumes main relevance and the use of a systematic experimental methodology is needed. The racing approach, here adopted, appears appropriate to fulfil this need.

**Acknowledgments** The authors want to thank Thomas Stützle for discussions, valuable suggestions and revisions that contributed to the development of this work. We are also grateful to Philipp Kostuch for providing the results of the comparison, and to the anonymous referees for their useful comments. This 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.

## References

Arntzen, H. and A. Løkketangen, "A tabu search heuristic for a university timetabling problem," in *Proceedings of the Fifth Metaheuristics International Conference*, Kyoto, Japan (Aug. 2003).

Birattari, M., *The Problem of Tuning Metaheuristics, as seen from a Machine Learning Perspective.* PhD thesis, Université Libre de Bruxelles, Brussels, Belgium (2004).

Birattari, M., "The race package for R. Racing methods for the selection of the best," Technical Report TR/IRIDIA/2003-37, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2003).

Birattari, M., T. Stützle, L. Paquete, and K. Varrentrapp, "A Racing algorithm for configuring metaheuristics," in W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, and N. Jonoska (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pp. 11–18, New York, 2002. Morgan Kaufmann Publishers.

Burke, E. K., A. J. Eckersley, B. McCollum, S. Petrovic, and R. Qu., "Analysing similarity in examination timetabling," in E. K. Burke and M. Trick, *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling. 2004*, pp. 557–559.

Burke, E. K., C. Beyrouthy, J. D. Landa Silva, B. McCollum, and P. McMullan, "SpaceMAP-applying metaheuristics to real world space allocation problems in academic institutions," in E. K. Burke and M. Trick (eds.), *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling. 2004*, pp. 441–444.

Burke, E. K., D. G. Elliman, and R. F. Weare, "Specialised recombinative operators for timetabling problems," in *AISB Workshop on Evolutionary Computing, Springer Verlag Notes in Computer Science Volume 993*, pp. 75–85. Springer Verlag, Berlin, Germany (1995).

Burke, E. K. , G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, **9**(6), 451–470 (2003).

Burke, E. K., J. P. Newall, and R. F. Weare, "A memetic algorithm for university exam timetabling," in E. K. Burke and P. Ross *Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 1153*, pp. 241–250.

Burke, E. K. and M. A. Trick (eds.), *Practice and Theory of Automated Timetabling V, 5th International Conference, PATAT 2004*, vol. 3616 of Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany (2005).

Burke, E. K., and M. Trick (eds.), *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling. PATAT 2004*, Pittsburgh, PA (Aug. 2004).

Burke, E. K. and M. W. Carter (eds.), *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT 1997*, vol. 1408 of Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany (1998).

Burke, E. K. and P. de Causmaecker (eds.), *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2002*, vol. 2740 of Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany (2003).

Burke, E. K. and P. Ross (eds.), *Practice and Theory of Automated Timetabling, First International Conference, PATAT 1995*, vol. 1153 of Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany, (1996).

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

Burke, E. K. and W. Erben (eds.), *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000*, vol. 2079 of Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany (2001).

Burke, E. K., Y. Bykov, and S. Petrovic, "A multicriteria approach to examination timetabling," in E. K. Burke and W. Erben, *Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 1153*, pp. 118–131.

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

Carter, M. W. and G. Laporte, "Recent developments in practical course timetabling," in E. K. Burke and M. W. Carter (eds.), *Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 1408*, pp. 3–19.

Carter, M. W., G. Laporte, and S. Y. Lee, "Examination timetabling: Algorithmic strategies and applications," *Journal of the Operational Research Society*, **47**, 373–383 (1996).

Chand, A., "A constraint based generic model for representing complete university timetabling data," in E. K. Burke and M. A. Trick (eds.), *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling. 2004*, pp. 125–150.

Chiarandini, M. and T. Stützle, "Experimental evaluation of course timetabling algorithms." Technical Report AIDA-02-05, Intellectics Group, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany (April 2002).

Chiarandini, M., *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems.* PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany, (Aug. 2005).

Cohen, P. R., *Empirical Methods for Artificial Intelligence.* MIT Press, Boston (1995).

Conover, W. J., *Practical Nonparametric Statistics.* 3rd edn, John Wiley & Sons, New York, NY, USA (1999).

Culberson, J. C., "Iterated greedy graph coloring and the difficulty landscape," Technical Report 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada (June 1992).

Custers, N. P. De Causmaecker, P. Demeester, and G. V. Berghe, "Semantic components for timetabling," in E. K. Burke and M. A. Trick (eds.), *Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 3616*, pp. 17–33.

De Werra. D., "An introduction to timetabling," *European Journal of Operational Research*, **19**(2), 151–162 (1985).

den Besten, M. L. *Simple Metaheuristics for Scheduling: An Empirical Investigation into the Application of Iterated Local Search to Deterministic Scheduling Problems with Tardiness Penalties*. PhD thesis, Darmstadt University of Technology, Darmstadt, Germany, October (2004).

Di Gaspero, L. and A. Schaerf, "Writing local search algorithms using EASYLOCAL++," in S. Voß and D.L. Woodruff, (eds.), *Optimization Software Class Libraries*, pp. 81–154. Kluwer Academic Publishers, Boston, MA, USA (2002).

Fink, A. and S. Voß, "A heuristic optimization framework," in S. Voß and D.L. Woodruff (eds.), *Optimization Software Class Libraries*, pp. 81–154. Kluwer Academic Publishers, Boston, MA, USA (2002).

Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness.* Freeman, San Francisco, CA, USA (1979).

Hertz, A. and D. de Werra, "Using tabu search techniques for graph coloring," *Computing*, **39**(4), 345–351 (1987).

Hoos, H. H. and T. Stützle, "Characterising the behaviour of stochastic local search," *Artificial Intelligence*, **112**(1–2), 213–232 (1999).

Hoos, H. H., and T. Stützle, "Evaluating Las Vegas algorithms—Pitfalls and remedies," in G. F. Cooper and S. Moral (eds.), *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pp. 238–245. Morgan Kaufmann Publishers, San Francisco, CA, USA (1998).

Johnson, D. S., C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning," *Operations Research*, **39**(3), 378–406 (1991).

Kostuch, P., "The university course timetabling problem with a three-phase approach," in E. K. Burke and M. A. Trick, *Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 3616*, pp. 109–125.

Kostuch, P., "University course timetabling," *Transfer Thesis*, Oxford University, England (2003).

Kostuch, P. and K. Socha, "Hardness prediction for the university course timetabling problem," in J. Gottlieb and G. R. Raidl, (eds.), *Evolutionary Computation in Combinatorial Optimization*, *Springer Lecture Notes in Computer Science Volume 3004*, pp. 132–141. Springer Verlag, Berlin, Germany (2004).

Lourenço, H. R., O. Martin, and T. Stützle, "Iterated local search," in F. Glover and G. Kochenberger (eds.), *Handbook of Metaheuristics*, 321–353. Kluwer Academic Publishers, Norwell, MA, USA (2002).

Morgenstern, C. and H. Shapiro, "Coloration neighborhood structures for general graph coloring," in *Proceedings of the first Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 226–235. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1990).

Newall, J. P., *Hybrid Methods for Automated Timetabling.* PhD thesis, Department of Computer Science, University of Nottingham, UK, May (1999).

Papadimitriou, C. H. and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

Post, G. and B. Veltman. "Harmonious personnel scheduling," in E. K. Burke and M. A. Trick (eds.), *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling. PATAT 2004*, pp. 557–559.

Risler, M., M. Chiarandini, L. Paquete, T. Schiavinotto, and T. Stützle, "An algorithm for the car sequencing problem of the ROADEF 2005 challenge," Technical Report AIDA-04-06, Intellectics Group, Computer Science Department, Darmstadt University of Technology (2004).

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

Rossi-Doria, O., B. Paechter, C. Blum, K. Socha, and M. Samples, "A local search for the timetabling problem." in E. Burke and P. Causmaecker (eds.), *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2002*, pp. 124–127 Gent, Belgium (August 2002).

Rossi-Doria, O., M. Samples, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle, "A comparison of the performance of different metaheuristics on the timetabling problem," in E. K. Burke and P. De Causmaecker, *Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 2740*, pp. 329–351.

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

Sedgewick, R. *Algorithms.* 2nd edn., Addison-Wesley, Reading, MA, USA (1988).

Setubal, J. C., "Sequential and parallel experimental results with bipartite matching algorithms," Technical Report EC-96-09, Institute of Computing, University of Campinas, Brasil (1996).

Sheskin, D. J., *Handbook of Parametric and Nonparametric Statistical Procedures.* 2nd edn., Chapman & Hall (2000).

Socha, K., "The influence of run-time limits on choosing ant system parameters," in Cantu-Paz et al. (eds), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, vol. 2723 of Lecture Notes in Computer Science, pp. 49–60. Springer Verlag, Berlin, Germany (July 2003).

Socha, K., M. Sampels, and M. Manfrin, "Ant algorithms for the university course timetabling problem with regard to the state-of-the-art," in Günther R. Raidl, Jean-Arcady Meyer, Martin Middendorf, Stefano Cagnoni, Juan J. Romero Cardalda, David Corne, Jens Gottlieb, Agnès Guillot, Emma Hart, Colin G. Johnson, and Elena Marchiori (eds.), *Applications of Evolutionary Computing: Proceedings of Evo Workshops 2003*, vol. 2611 of Lecture Notes in Computer Science, pp. 334–345. Springer Verlag, Berlin, Germany (2003).

Terashima-Marín, H., P. Ross, and M. Valenzuela-Rendón, "Evolution of constraint satisfaction strategies in examination timetabling," in W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. J. Jakiela, and R. E. Smith (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pp. 635–642. Morgan Kaufmann Publishers, San Francisco, CA, USA (1999).

Thompson, J. and K. Dowsland, "A robust simulated annealing based examination timetabling system," *Computers and Operations Research*, **25**(7–8), 637–648 (1998).

Wren, A.,"Scheduling, timetabling and rostering—a special relationship?," In E. K. Burke and P. Ross (eds.), *Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science Volume 1153*, pp. 46–75 (1996).