A local search for the timetabling problem

Olivia Rossi-Doria[1], Christian Blum[2], Joshua Knowles[2], Michael Sampels[2], Krzysztof Socha[2], and Ben Paechter[1]

[1] School of Computing, Napier University,
10 Colinton Road, Edinburgh, EH10 5DT, Scotland
{o.rossi-doria, b.paechter}@napier.ac.uk

[2] IRIDIA, Université Libre de Bruxelles, CP 194/6,
Av. Franklin D. Roosevelt 50, 1050 Bruxelles, Belgium
{cblum, jknowles, msampels, krzysztof.socha}@ulb.ac.be

Abstract. This work is part of the Metaheuristic Network, a European Commission project that seeks to empirically compare the performance of various metaheuristics on different combinatorial optimization problems. In this paper we define a representation, a neighbourhood structure and a local search for a university course timetabling problem. Our motivation is to provide a common search landscape for the metaheuristics that we aim to compare, allowing us to make a fair and meaningful analysis of the relative performance of these methods on a timetabling problem.

1 Introduction

The aim of the Metaheuristics Network* — a European Commission project undertaken jointly by five European institutions — is to compare and analyse the performance of various metaheuristics on different combinatorial optimization problems including timetabling.

Every educational institution periodically faces the problem of producing a timetable for its courses. A general course timetabling problem consists in assigning a set of events (classes, lectures, tutorials, etc) into a limited number of timeslots, so that a set of constraints are satisfied. Constraints are usually classified in hard and soft. Hard constraints must not be violated under any circumstances, *e.g.* students cannot attend two different events at the same time. Soft constraints should preferably be satisfied, but they can be accepted with a penalty associated to their violation, *e.g.* students should not attend the last class of the day. These problems have been extensively studied over the last few years [**?**,**?**,**?**].

We study here a reduction of a typical university course timetabling problem, that has been designed by Ben Paechter to reflect aspects of the real timetabling problem of Napier University.

The main objective of this work is to provide a common framework as part of the definition of a controlled and unbiased experimental methodology for the

* http://www.metaheuristics.net/

comparison of five different metaheuristics. In particular we aim here to define a common search landscape to use in the implementations of straightforward versions of the metaheuristics that we seek to compare. We ultimately aim to compare under similar circumstances the performance of an Evolutionary Algorithm, an Ant Colony Optimization, an Iterated Local Search, a Simulated Annealing, and a Tabu Search. The results of this comparison are given in [**?**].

2 The problem

The problem consists of a set of events or classes E to be scheduled in 45 timeslots (5 days of 9 hours each), a set of rooms R in which events can take place, a set of students S who attend the events, and a set of features F satisfied by rooms and required by events. Each student attends a number of events and each room has a size. A feasible timetable is one in which all events have been assigned a timeslot and a room so that the following hard constraints are satisfied:

- no student attends more than one event at the same time;
- the room is big enough for all the attending students and satisfies all the features required by the event;
- only one event is in each room at any timeslot.

In addition, a candidate timetable is penalised equally for each occurrence of the following soft constraint violations:

- a student has a class in the last slot of the day;
- a student has more than two classes in a row;
- a student has a single class on a day.

Note that the soft constraint have been chosen to be representative of three different classes: the first one can be checked with no knowledge of the rest of the timetable; the second one can be checked while building a solution, taking into account the events assigned to nearby timeslots; and finally the last one can be checked only when the timetable is complete, and all events have been assigned a timeslot.

3 The instances

A problem instance generator produces problem instances with different characteristics for different values of given parameters. All instances produced have a perfect solutions, *i.e.* a solutions with no constraint violations, hard or soft. The parameters used to create each instance are: the number of events, the number of rooms, and the number of features, the approximate average number of features to be assigned to each room, the approximate percentage of features used by the events, the number of students, the maximum number of events that any student can attend in the week, the maximum number of students that can attend any one event, and an integer used in the random process - using the same

integer will produce the same problem instance - a different integer will produce a different instance with the same characteristics.

Three sizes of instances have been selected for testing purposes, respectively called small, medium and large. They are generated with the sets of parameters reported in Table **??**.

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

Table 1. Parameters used to produce the different instance classes.

4 The solution representation

The solution representation chosen is a direct representation. A solution consists of an ordered list of length $|E|$ where the positions correspond to the events (position i corresponds to event i for $i = 1, ..., |E|$). An integer number between 1 and 45 (representing a timeslot) in position i indicates the timeslot to which event i is assigned. For example the list

$$3 \;\; 27 \;\; 43 \;\; \ldots \;\; 10$$

means that event 1 is assigned to timeslot 3, event 2 is assigned to timeslot 27, and so on.

The room assignments are not part of the explicit representation; instead we use a matching algorithm to generate them. For every timeslot there is a list of events taking place in it, and a preprocessed list of possible rooms to which these events can be assigned according to size and features. The matching algorithm gives a maximum cardinality matching between these two sets using a deterministic network flow algorithm. If there are still unplaced events left, it takes them in label order and puts each one into the room of correct type and size which is occupied by the fewest events. If two or more rooms are tied, it takes the one with the smallest label. This procedure ensures that each event-timeslot assignment corresponds uniquely to one timetable, *i.e.* a complete assignment of timeslots and rooms to all the events.

Note that with this representation we are implicitely taking care and never breaking the hard constraint that requires that an event has to be assigned to a room suitable for it. We might however introduce a different kind of soft constraint violations by rather allowing two events to take place in the same room at the same time.

5 The neighbourhood structure

The solution representation described above allows us to define a neighbourhood using simple moves involving only timeslots and events. The room assignment are taken care of by the matching algorithm.

We considered three types of move on a timetable involving one, two and three events respectively:

- a move of type 1 moves one event from a timeslot to a different one;
- a move of type 2 swaps two events in two different timeslots;
- a move of type 3 permutes three events in three distinct timeslots in one of the two possible ways.

Each type of move defines a neighbourhood denoted N_1, N_2, N_3, respectively. The complete neighbourhood N is then defined as the union of these three neighbourhoods: $N = N_1 \cup N_2 \cup N_3$.

6 The local search

Given the size of the neighbourhood we use a first improvement local search which works as follows. Consider in order moves of type 1, 2 and 3 for every event involved in constraint violations. Initially ignore soft constraint violations. Then, if the timetable reaches feasibility, consider soft constraints as well, but never go back from a feasible to an infeasible solution. For each potential move re-apply the matching algorithm to the affected timeslots and delta-evaluate the result. The local search is ended if no improvement is found after trying all possible moves on each event of the timetable. The search is stochastic as it looks at events in a randomly generated order.

A more detailed description of the algorithm follows:

1. Ev-count $\leftarrow 0$;
   Generate a circular randomly-ordered list of the events;
   Initialize a pointer to the left of the first event in the list;
2. Move the pointer to the next event;
   Ev-count $\leftarrow$ Ev-count + 1;
   if (Ev-count = $|E|$) {
   Ev-count $\leftarrow 0$;
   goto 3.; }
   (a) **if** (current event NOT involved in hard constraint violation (hcv))
   { **goto** 2.; }
   (b) **if** ($\nexists$ an untried move for this event) { **goto** 2.; }
   (c) Calculate next move (first in N_1, then N_2, then N_3)** and generate resulting potential timetable;

** That is, for the event being considered, potential moves are calculated in strict order. First, we try to move the event to the next timeslot, then the next, then the next etc. If this search through N_1 fails then we move through the N_2 neighbourhood, by trying to swap the event with the next one in the list, then the next one, and so on. The same then occurs for N_3.

(d) Apply the matching algorithm to the timeslots affected by the move and delta-evaluate the result;
(e) **if** (move reduces hcvs) {
Make the move;
Ev-count $\leftarrow$ 0;
goto to 2.;}
(f) **else goto** 2.(b);
3. **if** ($\exists$ any hcv remaining) END LOCAL SEARCH;
4. Move the pointer to the next event;
Ev-count $\leftarrow$ Ev-count + 1;
if (Ev-count = $|E|$) END LOCAL SEARCH;
(a) **if** (current event NOT involved in soft constraint violation (scv))
{ **goto** 4.; }
(b) **if** ($\nexists$ an untried move for this event) { **goto** 4.; }
(c) Calculate next move (first in N_1, then N_2, then N_3)** and generate resulting potential timetable;
(d) Apply the matching algorithm to the timeslots affected by the move and delta-evaluate the result;
(e) **if** (move reduces scvs without introducing a hcv) {
Make the move;
Ev-count $\leftarrow$ 0;
goto 4.; }
(f) **else goto** 4.(b);

7 The experimental results

The local search described above is very slow, and when used within the context of a metaheuristic technique it is more efficient to reduce its depth and width, allowing the metaheuristic a greater proportion of CPU time. For this purpose we introduced four parameters on the local search, precisely: the probability of each of the three types of move to be performed, respectively p_1, p_2, and p_3 for moves of type 1, 2, and 3, and a time limit t.

We tested different values of these four parameters for the local search within a simple iterated local search. The perturbation is just a random move in the complete neighbourhood N, and we use an "Accept if Better" acceptance criterion. One representative of each class of instances, small, medium and large, has been selected for the experiments. We run ten trials of each parameter configuration on them respectively for 90, 900, and 9000 seconds on a PC with an AMD Athlon 1100 Mhz.

The results are summarized in Figures 1, 2, 3, 4 and 5 using boxplots. A box shows the range between the 25% and the 75% quantile of the data. The median is indicated by a bar. The whiskers extend to the most extreme data point which is no more than 1.5 times the interquantile range from the box. Outliers are indicated as circles.

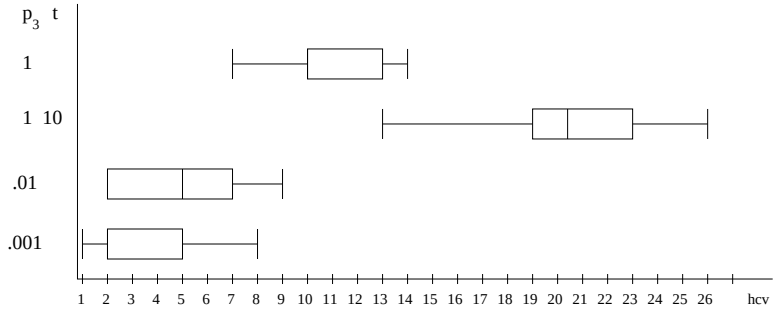

Fig. 1. Hard constraint violations on the large instance for $p_1 = p_2 = 1$ and $p_3 \neq 0$.

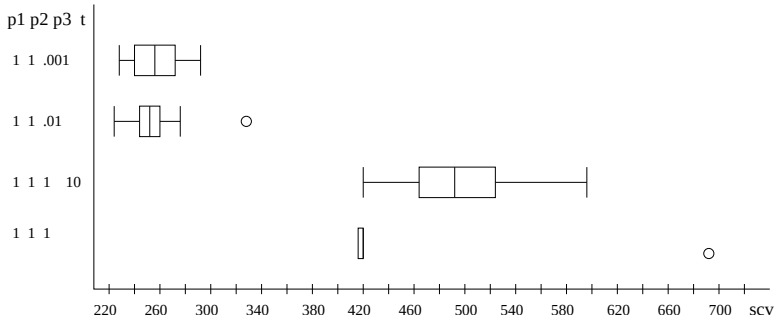

Fig. 2. Soft constraint violations on the medium instance for $p_1 = p_2 = 1$ and $p_3 \neq 0$.

The search through neighbourhood N_3 is very time consuming for all the three instances with no good results, as shown for the large and medium instances in Figure 1 and 2.

For $p_3 = 1$ on the large instance not even a single run of the local search finishes in the total time, and this is as well mostly true for the medium and small instances, where only occasionally the algorithm managed to complete one or two runs. More iterations are completed but performance does not improve much when the moves of type 3 are performed with smaller probability, even when it is as low as .001, or when the local search is cut with a time limit of 10 seconds. Better results are achieved discarding the search through neighbourhood N_3. For this reason we tried different parameters for p_1 and p_2 keeping $p_3 = 0$, and results for these experiments are discussed in the following.

For the large instance feasibility is difficult to achieve. Only for $p_1 = .5$, $p_2 = 1$ the algorithm managed to find two feasible solutions out of ten, while for $p_1 = 1$, $p_2 = 1$ and $p_1 = 1$, $p_2 = 0$ it found feasible timetables only once. Moreover for minimizing hard constraint violations $p_1 = 1$, $p_2 = 1$ and $p_1 = .8$, $p_2 = 1$ seem to perform best, so that reducing a bit but not too much (results for $p_1 = .2$, $p_2 = 1$ are slightly worse) the search through neighbourhood N_1 but going through all neighbourhood N_2 appears to be the best strategy. On the contrary keeping $p_1 = 1$ results are better for greater values of p_2. The attempt to

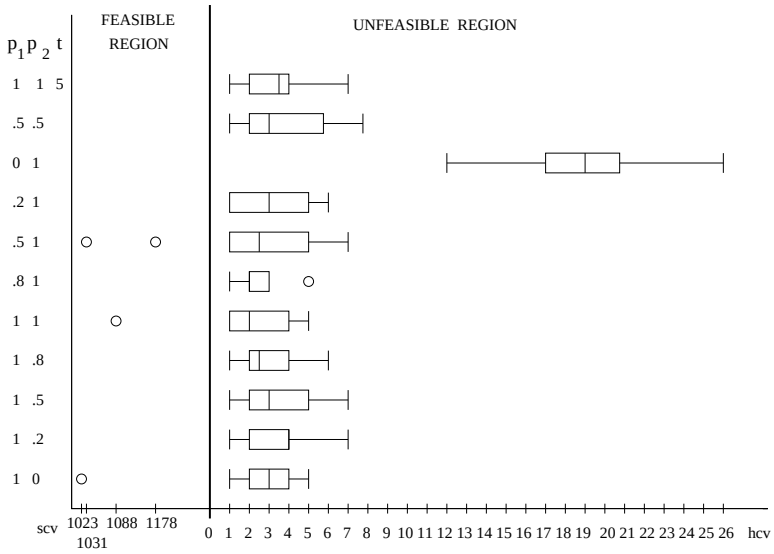

Fig. 3. Solution quality on the large instance for $p_3 = 0$.

search both neighbourhood N_1 and N_2 with probability 0.5 gives worse results. Results of the experiments for the large instance are summarized in Figure 3.

For the medium instance when p_3 is zero the best results are once again reached for $p_2 = 1$ but, with it fixed, values for p_1 go the other way around than in the hard instance and smaller is better. The search through the two neighbourhoods N_1 and N_2 with equal probability 0.5 is this time better than the full search through N_1, where smaller values for p_2 are again better than greater ones. Experiments for the medium instance are summarized in Figure 4.

For the small instance, the use of N_3 still gives not very good results but performance is increasingly better as p_3 gets smaller. Little difference is noticeable for different values of p_1 and p_2 when p_3 is zero, performance being slightly better for $p_1 = 1$, $p_2 = .8$, for $p_1 = .8$, $p_2 = 1$ and for $p_1 = .5$, $p_2 = 1$. The complete search through neighbourhood N_1 only gives comparable results for this smaller instance where evidently there is no much need of a bigger neighbourhood. Results for the small instance are summarized in Figure 5.

The search through only neighbourhood N_2 gives the worst results across all three instances, even when a probability as small as .2 for p_2 gives reasonable results. This is of course not really surprising, since neighbourhood N_2 is not connected.

We have presented here a neighbourhood structure and a parametrized local search designed as common search landscape for the comparison of different metaheuristic techniques. This work was part of a first phase of a study on the timetabling problem carried out for the Metaheuristics Network.

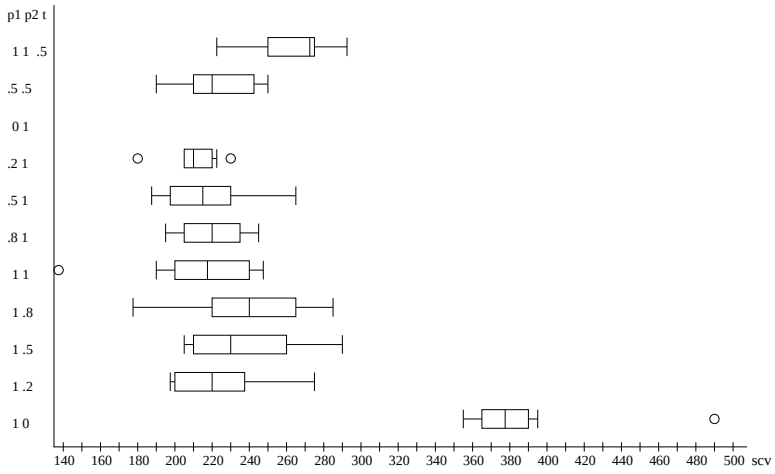

Fig. 4. Soft constraint violations for $p_3 = 0$ on the medium instance.

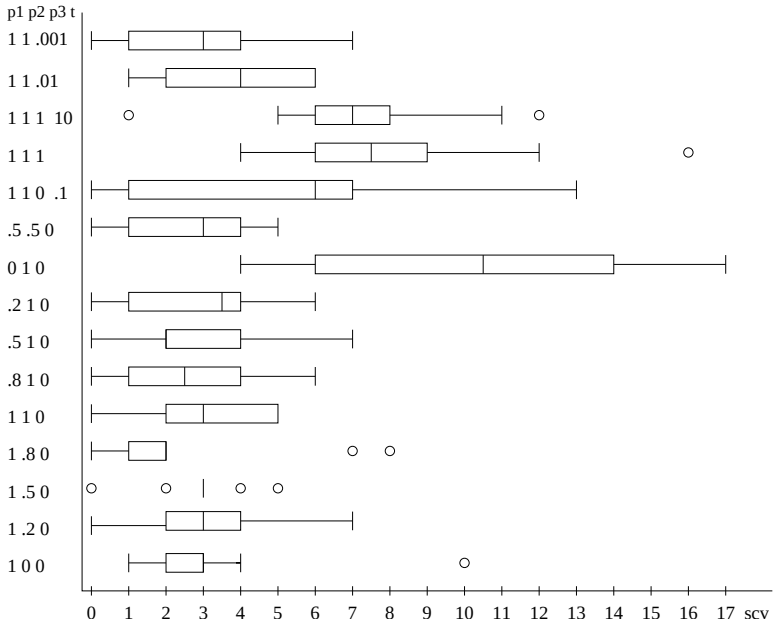

Fig. 5. Soft constraint violations for the small instance.

Acknowledgments: This work was supported by the *Metaheuristics Network*, a Research Training Network funded by the Improving Human Potential program 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

1. M. W. Carter, G. Laporte. Recent developments in practical course timetabling. In [**?**], 3–19, 1997.
2. E. K. Burke, M. Carter (eds.) The Practice and Theory of Automated Timetabling: Selected Papers from the Second International Conference. Lecture Notes in Computer Science 1408, Springer-Verlag, Berlin, 1997.
3. B. Paechter, R. C. Rankin, A. Cumming, T. C. Fogarty. Timetabling the classes of an entire university with an evolutionary algorithm. Parallel Problem Solving from Nature (PPSN) V. Lectures Notes in Computer Science 1498, Springer-Verlag, Berlin, 865–874, 1998.
4. O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, T. Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. In Proceedings of the 4th international conference on the Practice And Theory of Automated Timetabling (PATAT 2002), Gent, Belgium, 115–123, 2002.
5. A. Schaerf. A survey of Automated Timetabling. Artificial Intelligence Review, 13:87–127, 1999.