A Time-Dependent Metaheuristic Algorithm for Post Enrolment-based Course Timetabling

Rhyd Lewis
Prifysgol Caerdydd/Cardiff University,
Cardiff Business School, Colum Drive, Cardiff,
WALES.
Tel: +44(0)29 2087 5559
Fax: +44(0)29 2087 4419
email: `lewisR9@cf.ac.uk`

June, 2008

Abstract

A metaheuristic-based algorithm is presented for the post enrolment-based course timetabling problem used in track-2 of the Second International Timetabling Competition (ITC2007). The featured algorithm operates in three distinct stages – a constructive phase followed by two separate phases of simulated annealing – and is time dependent, due to the fact that various run-time parameters are calculated automatically according to the amount of computation time available. Overall, the method produces results in line with the official finalists to the timetabling competition, though experiments show that this algorithm also seems to find certain instances more difficult to solve than others. A number of reasons for this latter feature are discussed.

1 Introduction

During late 2007 and early 2008 the Second International Timetabling Competition (ITC2007) was organised and run by a group of timetabling researchers from five different European Universities. The overall aim of this competition was to help people interested in timetabling from various fields to compare and contrast their timetabling methods using a common set of benchmark instances in a fair and accurate way.

On August the 1st 2007, the competition was officially started by the release of a number of problem instances into the public domain. Entrants were invited to register with the competition and to use these instances to help design algorithms that produced solutions according to the competition evaluation criteria. On January 11th 2008, two weeks before the end of the competition, a second set of problem instances was then also released. Competitors were required to submit their results from both instance sets to the competition organisers by the 25th of January. The organisers then collected the executables from selected entrants and ran these on their own benchmark machines, together with a third "hidden" set of problem instances to officially rank the various algorithms and choose a winner. Further details can be found on the competition's official webpage at `http://www.cs.qub.ac.uk/itc2007/`

Before holding ITC2007, it was decided that the competition would be split into three tracks, each reflecting a different type of university timetabling problem, namely: (1) Exam timetabling, (2) Post Enrolment-based Course timetabling, and (3) Curriculum-based timetabling. Full descriptions of each of these problems can be found in the various specification reports (McCollum et al., 2007; Lewis et al., 2007; Di Gaspero et al., 2007), available on the competition webpage.

In this paper, we present an algorithm for the Post Enrolment-based Course timetabling problem used in track-2 of the competition. This particular problem model simulates the real-world situation where students are given a choice of lectures that they wish to attend, with the timetable then being constructed according to these choices. The problem model is also based on the timetabling problem used for the first international timetabling competition run in 2003, albeit with extra features (outlined in Section 2 below). Research arising due to the first competition includes the ant colony optimisation approach of Socha and Samples (2003), the simulated annealing-based algorithm of Kostuch (2005), and the mixed metaheuristic approach of Lewis (2006). A number of metaheuristic-based approaches for this problem-version are also offered by Rossi-Doria et al. (2002). Various pieces of useful information concerning the first competition are also available its official website at `www.idsia.ch/files/ttcomp2002/`

It should be noted that the algorithm presented in this paper was not officially entered into ITC2007 due to the fact that the author was one of the competition organisers. However, in the spirit of fair-play, and in order to avoid any unfair advantages, the author chose to work within the rules of the competition and did not make use of any of the problem instances before they were officially released to all other competition entrants.

In the next section we give a specification of the timetabling problem-version considered here. Readers interested in discovering more about this problem, including the rationale of *why* this problem-version takes the form that it does, are invited to consult the problem's official specification document (Lewis et al., 2007), available on the ITC2007 website. Following this, the proposed algorithm is then described in detail in Section 3. Section 4 then contains details on the implementation and an analysis of the final results gained by the algorithm. Finally, Section 5 concludes the paper.

2 Problem description

A problem instance for track-2 of ITC2007 contains the following information:

- A set of n events that are to be scheduled into 45 timeslots (to be interpreted as five days of nine, 1-hour timeslots);
- A set of m rooms where the events are to take place, each which has a specific seating capacity;

- A set of room-features that are *required* by events and which are *satisfied* by rooms;
- A set of s students who attend various combinations of the events;
- A set of available timeslots for each of the n events (i.e. not all events are available to be scheduled in all of the timeslots);
- A set of precedence requirements stating that certain events should occur before/after others in the timetable.

Given this information, the aim is to assign all of the n events to a room and a timeslot, whilst obeying the following five hard constraints:[1]

HC_1: No student should be required to attend more than one event in a particular timeslot;

HC_2: Each event should be assigned to a room that has enough seats for all of the attending students and which satisfies all of the room-features required by the event;

HC_3: Only one event should be put into each room in any timeslot (i.e. no double-booking of rooms);

HC_4: Events should only be assigned to timeslots that are designated as "available" for those events;

HC_5: Where specified, events should be scheduled to occur in the correct order in the week

In addition to these five hard constraints, it is also desirable for the following three soft constraints to be satisfied:

SC_1: Students should not be scheduled to attend an event in the last timeslot of a day (that is, timeslots 9, 18, 27, 36, or 45);

SC_2: Students should not have to attend events in three or more successive timeslots occurring in the same day;

SC_1: Students should not be required to attend just one event in a particular day

As is typical in timetabling research, for this problem the satisfaction of the hard constraints is considered to be more important than the satisfaction of the soft constraints. Because of this, the competition rules state that a candidate solution should be evaluated according to two separate values: the *Distance to Feasibility* and the *Soft Cost.* The Distance to Feasibility is used because – in contrast to the first competition – it is necessary to allow for the fact that an algorithm may not be able to assign all of the n events into the timetable whilst obeying the hard constraints. For this competition, it is therefore permissible to allow some events to remain *unplaced* in order to ensure

[1]Note that hard constraints 1-3 are the same as those used in the first timetabling competition.

that none of the hard constraints are violated. (The ITC2007 rules stipulate that if an algorithm produces a solution that contains any violations of the hard constraints, then it is considered invalid and should be disqualified from the competition.) The Distance to Feasibility is thus calculated by considering the events that are *not* placed in the timetable. However, it is not the number of events that are considered here; rather it is the total number of students that *attend* each of these unplaced events, reflecting the real-world situation where we are interested in satisfying as many people's needs as possible within a timetable. Thus if, for example, a particular solution timetable has three events that are unplaced, and the number of students attending each of these is 10, 5, and 8, then the Distance to Feasibility is simply $(10 + 8 + 5) = 25$. Note that if all events are inserted into a timetable (whilst obeying the hard constraints), then its Distance to Feasibility is zero.

The second value used for timetable evaluation is the *Soft Cost*, which is calculated by simply counting the number of soft constraint violations in a timetable. For SC_1, if a student is scheduled to attend an event in the last timeslot of a day, then this results in one penalty point. (Naturally, if there are x students in this class, we consider this as x penalty points.) For SC_2 if one student has three events in a row we count this as one penalty point. If a student has four events in a row we count this as two, and so on. Note that adjacent events occurring over two separate days are not counted as a violation. Finally, for SC_3 each time a student attends just one event on a day, this results in one penalty point. The Soft Cost is simply the total of these three values. (Further information on this cost measure can be found in (Lewis et al., 2007) or on either of the competition webpages.)

Given these two values, the following procedure is used to compare solutions. First, the solutions' Distances to Feasibility are considered, and the solution with the lowest value is deemed the winner. However, when two or more solutions are equal in this respect, the winner is deemed the solution among these that has the lowest Soft Cost.

Sixteen problem instances are currently publicly available for track-2 of the competition: the eight "early" instances, which were released on the start date, and the eight "late" instances, released two weeks before the close of the competition. These were created using an automated generator and all are known to have at least one perfect solution – that is, a solution where all of the n events are assigned to the timetable without any constraint violations, hard or soft. For the competition, a benchmark timing program was also released which entrants were required to execute on their own machines. This program specifies a time limit for each machine and ensures that entrants use approximately the same amount of computational effort when testing their algorithms.

3 Algorithm Description

In this approach the strategy is to tackle the problem in three distinct stages, each with a strict time limit, T_1, T_2, and T_3 respectively. For our purposes, 1/3 of the available time limit T is allocated to each stage, though of course other settings could be used in practice. If a particular stage completes before reaching its time limit, then the remaining time is passed on to the next stage. If Stage 3 completes early, then the algorithm also halts early.

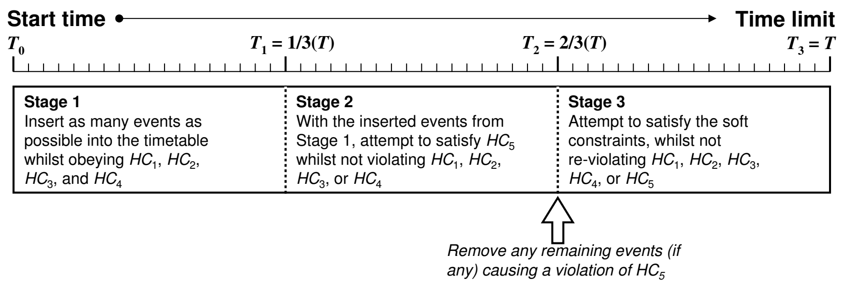
For guidance, a description of the main objectives of each stage is given in fig. 1. As this demonstrates, the idea is to arrange the constraints into three different levels of importance. At each successive stage, violations of constraints satisfied in previous stages are then disallowed. In the first stage, attempts are made to try and satisfy hard constraints 1 to 4 using specialised procedures that were proposed in some of our earlier work (Lewis, 2006). Although these methods are generally effective, they do not, however, seem immediately applicable to the remaining hard constraint HC_5; thus the second stage of the algorithm is concerned with the removal of violations of this constraint. Finally, in Stage 3, the algorithm concentrates on removing as many soft constraint violations as possible from the current timetable. At this point any unplaced events are ignored.

In Stages 2 and 3 of this algorithm, optimisation is carried out using simulated annealing (Kirkpatrick et al., 1983). Because this algorithm runs according to time limits, when applying this metaheuristic it will be useful to calculate cooling schedules that take the amount of available computation time into account. The aim is to therefore allow the algorithm to perform a slower cooling (and therefore a wider, more global search) when presented with a generous amount of run time, and a quicker cooling (with a more intensive, greedy search) when only small amounts of time are available. The method for calculating these parameters is described in Section 3.3.

In the next subsection, we will discuss some encoding and preprocessing issues that are relevant in the design of this algorithm. Subsections 3.2, 3.3, and 3.4 will then describe Stages 1, 2, and 3 of the algorithm respectively.

3.1 Encoding and Preprocessing Issues

For this approach a timetable is encoded using a two-dimensional ($r \times 45$) matrix (i.e. grid) in which rows represent rooms and columns represent timeslots. Throughout this paper we refer to this timetable matrix as tt and will use the notation $tt[i, j]$ to denote the contents at location (i, j). Each cell in this grid (i.e. *place* in the timetable) can be *blank* or will be *occupied* by exactly one



Start time
Time limit
T_0
T_1 = 1/3(T)
T_2 = 2/3(T)
T_3 = T
Stage 1
Insert as many events as possible into the timetable whilst obeying HC_1, HC_2, HC_3, and HC_4
Stage 2
With the inserted events from Stage 1, attempt to satisfy HC_5 whilst not violating HC_1, HC_2, HC_3, or HC_4
Stage 3
Attempt to satisfy the soft constraints, whilst not re-violating HC_1, HC_2, HC_3, HC_4, or HC_5
Remove any remaining events (if any) causing a violation of HC_5

Figure 1: High level description of the three stage algorithm. Here, T represents the time limit defined by the competition benchmarking program

event. Note that this latter feature means that it is impossible to double-book a room, allowing us to disregard HC_3.

It is also useful to carry out some preprocessing steps before executing the main body of the algorithm. First, two additional matrices are constructed: the *event-room* matrix, and the *conflicts matrix* (these were also used in the first timetabling competition by Kostuch (2005)). The event-room matrix is of dimensions $n \times m$ and is used to indicate which rooms are suitable for which events. This can be easily calculated for an event i by simply identifying which rooms satisfy both the seating capacity and the features required by i. The $n \times n$ conflicts matrix, meanwhile, is very much like the standard adjacency matrix used for representing graphs and indicates which pairs of events *conflict* (i.e. cannot be assigned to the same timeslot). Thus, if two events i and j have one or more students in common, or if both i and j can only be assigned to the same single room r, then it is obvious that events can never be feasibly assigned to the same timeslot, and so elements (i, j) and (j, i) in the conflicts matrix can be marked as true.

Our final act of preprocessing considers the hard constraint HC_4. First of all, note that if we have a constraint "event i must occur before event j", then this will automatically imply the constraint "event j must occur *after* event j". For this approach, this means that all occurrences of HC_5 can be conveniently stored in a compact way using an array A of n lists, where each list $A[i]$ contains only the events that need to be scheduled *before* event i in the timetable. (The "after" constraints do not need to be considered). Second, we can also make further additions to A by noting that hard constraint HC_5 is *transitive* (i.e. if event i must occur before event j, and event j must occur before event k, then this implies that event i must also occur before event k). In some of the competition instances, not all of the implied constraints due to this transitivity are present in the given problem files, and so it makes sense to calculate these in order to gain a better understanding of the number of constraints that need to be considered when trying to solve the problem.

3.2 Algorithm Description: Stage 1

In Stage 1, the objective is to insert as many of the n events into the timetable as possible without violating the first four hard constraints. A precise pseudo-code description is presented fig. 2. As is shown, this stage takes as arguments the empty timetable tt, an iteration-limit I, and a list of currently unplaced events U (to begin with, $|U| = n$). Events are then taken one-by-one from U and are inserted into suitable places in tt.[2] In order to try and maximise the number of events that are inserted at this point, heuristic rule h_1 is used to select the next event, with ties being broken using h_2, and further ties with h_3 (refer to Table 1). Note that these heuristics are akin to those used in the Dsatur algorithm for graph colouring (Brelaz, 1979), though in this case h_1 also takes the issue of room allocation into account. Rule h_1 therefore selects events based on the state of the current partial timetable tt, and prioritises those with the least remaining feasible options.

[2]In this section the term "suitable" is used to indicate a place in the timetable that will not result in the violation of hard constraints 1 to 4.

Table 1: Heuristic rules used in Stage 1.

Heuristic	Description
h_1	Choose the event with the smallest number of suitable places in *tt* to which it can be assigned.
h_2	Choose the unplaced event that conflicts with the most other events.
h_3	Choose an event randomly.
h_4	Choose the place that is suitable for the least number of other unplaced events in U.
h_5	Choose the place in the timeslot with the fewest events in.
h_6	Choose a place randomly.
h_7	Choose the event with the least number of students.

Meanwhile, rule h_2 prioritises those events that have the highest number of conflicts which, as a rule of thumb, are often the more problematic events to insert. Note that events with no remaining place-options are ignored at this point. Finally, to select a place for each event, rule h_4 is used, which chooses the place whose occupation will have the least effect on the place-options of the remaining unplaced events in U. Ties of this rule are broken using h_5 and further ties with h_6.

At the end of this assignment stage, the list U will be empty (in which case all of the events have been assigned to the timetable), or U will only contain events that have no suitable places in *tt*. In the latter case, the procedure ITERATED-HEURISTIC-SEARCH is called, which is used to try and transfer further events from U into *tt*, ensuring that hard constraints 1-4 are not violated in the process. To start, the sub-procedure HEURISTIC-SEARCH is called, which operates by repeatedly attempting to move events from U into free (i.e. blank) places in *tt* (lines (3)-(7)). While doing this, however, HEURISTIC-SEARCH also shuffles the events *within tt* so that the free places change position (lines 9-15). The rationale of this latter action is that it offers the possibility of further events in U being added to *tt* when we loop back to line (1) of HEURISTIC-SEARCH. However, although the HEURISTIC-SEARCH procedure is quite effective in reducing the number of unplaced events, in initial experiments it was also noticed that it is only able to do this for a fairly short period of time, after which the process stagnated, with no further events being transferred from U into *tt*. To counter this, ITERATED-HEURISTIC-SEARCH therefore includes a mechanism intended for re-invigorating the process, which is achieved by the procedure EXTRACT-SOME-EVENTS, which removes other events from *tt* and puts these into a second list V. Of course, by removing events from *tt*, extra free places are created that can be used by some of the events in U. Thus the events in V are put to one side temporarily, and HEURISTIC-SEARCH is again applied using U and the new, emptier timetable. Finally, upon completion of this second phase of heuristic search, the events in V are added to the events (if any) that still reside in U and the entire ITERATED-HEURISTIC-SEARCH process is repeated.

Examining ITERATED-HEURISTIC-SEARCH, two important features become apparent. First, it

```
Stage-1($tt$, $U$, $I$)
(1)   while ($\exists$ events in $U$ with suitable places in $tt$)
(2)      Select an event $e \in U$ that has suitable places in $tt$;
(3)      Choose a suitable place $p$ for $e$;
(4)      Move $e$ from $U$ into $tt$ at place $p$;
(5)   Iterated-Heuristic-Search($tt$, $U$, $I$);

Iterated-Heuristic-Search($tt$, $U$, $I$)
(1)   while ($U \neq \emptyset$ and (timelimit $T_1$ not reached))
(2)      Heuristic-Search($tt$, $U$, $I$);
(3)      if ($U \neq \emptyset$)
(4)         $V \leftarrow$ Extract-Some-Events($tt$, $|U|$);
(5)         Heuristic-Search($tt$, $U$, $I$);
(6)         $U \leftarrow U \cup V$;

Heuristic-Search($tt$, $U$, $I$)
(1)   Make a list $P$ of all the unoccupied places in $tt$;
(2)   $i \leftarrow 0$;
(3)   while ($U \neq \emptyset$ and $P \neq \emptyset$ and $i < I$)
(4)      foreach ($u \in U$ and $p \in P$)
(5)         if ($p$ is a suitable place in $tt$ to assign $u$)
(6)            Put $u$ into $p$ in $tt$;
(7)            Remove $u$ from $U$ and $p$ from $P$;
(8)      if ($U \neq \emptyset$ and $P \neq \emptyset$)
(9)         repeat
(10)           Choose a random event $e$ in $tt$ and $p \in P$;
(11)           if ($p$ is suitable place in $tt$ to assign $e$)
(12)              Move $e$ from its current place to $p$;
(13)              Update $P$ to reflect the changes;
(14)           $i \leftarrow i + 1$;
(15)        until ($i = I$ or ($e$ has been moved to $p$))

Extract-Some-Events($tt$, $q$)
(1)   $V \leftarrow \emptyset$;
(2)   for ($i \leftarrow 1$ to $q$)
(3)      Randomly choose two events $e$ and $g$ in $tt$;
(4)      Move either $e$ of $g$ (according to $h_7$) from $tt$ to $V$;
```

Figure 2: Pseudo-code description of Stage 1. Here, tt represents the $(r \times 45)$ timetable matrix and U and V are lists of unplaced events. (When Stage-1 is first called, $|U| = n$.) I defines the iteration limit of the Heuristic-Search procedure.

is noticeable that if we choose to transfer some events from tt into V (line (4)) and then subsequently add the contents of V to U (line (6)), then unless the heuristic search operation on line (5) has managed to transfer a particular number of events from U into tt, then the overall number of unplaced events may actually increase, thus conflicting with the objectives of Stage 1. However, in practice if such a situation arises, it is usually only temporary because the number of events in U is generally seen to decrease again when the algorithm loops back to the start of the procedure. The second issue, meanwhile, concerns the strategy of event extraction used in EXTRACT-SOME-EVENTS. One choice available here is to simply choose events randomly for removal. However, in this particular case it seems sensible to bias the choice towards removing smaller events from the timetable, due to the fact that unplaced events containing less students will attract a lower Distance to Feasibility measure when the timetable is evaluated. Thus, heuristic rule h_7 (Table 1) is used here. Note also that $|U|$ events are currently extracted here, though a different value could be used here in theory.

Table 2 summarises the effectiveness of Stage 1 on the sixteen problem instances. Details on the sizes of each instance are also included here: the number of events n, rooms m and students s. It can be seen that with twelve of the instances, an average of over 90% of events are inserted into the timetable by the heuristic assignment rules h_1 to h_6. With the remaining four instances, which each feature averages in the low to mid 80s, it is noticable that $n = 400$ and $m = 10$ in each case. This means that if all events are inserted into these timetables, then a proportion of $\frac{400}{450} = 0.89$ of the available places will be occupied, which is far higher than the remaining twelve instances, which each have occupancy rates of ≤ 0.44. Regardless of this however, it can also be seen that after the subsequent application of ITERATED-HEURISTIC-SEARCH, generally all of the remaining events are inserted into the timetable, thus fulfilling the objectives of Stage 1.

3.3 Algorithm Description: Stage 2

In Stage 2 of the algorithm, attention is turned towards eliminating violations of the remaining hard constraint HC_5. As mentioned earlier, this is done using simulated annealing (SA). The cost function $C(tt)$ used in this phase is:

$$C(tt) = \sum_{i=1}^{n} \sum_{j=1}^{|A[i]|} f(i, j), \tag{1}$$

where

$$f(i, j) = \begin{cases} 1 & \text{if } (\text{slot}(i) \geq \text{slot}(A[i]_j)) \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

(Here, A refers to the array of lists described in Section 3.1, $A[i]_j$ indicates the jth element in the list $A[i]$, and slot(i) indicates the timeslot that event i is assigned to in the timetable tt.) This cost function therefore reflects the number of violations of HC_5 and, obviously, the aim in this stage is to try and produce a solution tt with $C(tt) = 0$.

Table 2: Percentage of events inserted into the timetable before and after applying Iterated-Heuristic-Search. Presented figures are the means of 51 runs on each instance together with the standard deviation.

Instance	n	m	s	Before (%)	After (%)
`comp-2007-2-1`	400	10	500	86.3 ± 1.2	100 ± 0.0
`comp-2007-2-2`	400	10	500	83.8 ± 1.2	100 ± 0.0
`comp-2007-2-3`	200	20	1000	95.3 ± 1.2	100 ± 0.0
`comp-2007-2-4`	200	20	1000	92.1 ± 1.3	100 ± 0.0
`comp-2007-2-5`	400	20	300	92.8 ± 1.0	100 ± 0.0
`comp-2007-2-6`	400	20	300	92.1 ± 0.9	100 ± 0.0
`comp-2007-2-7`	200	20	500	93.2 ± 1.1	100 ± 0.0
`comp-2007-2-8`	200	20	500	93.3 ± 1.1	100 ± 0.0
`comp-2007-2-9`	400	10	500	85.6 ± 1.4	100 ± 0.0
`comp-2007-2-10`	400	10	500	81.1 ± 1.4	100 ± 0.1
`comp-2007-2-11`	200	10	1000	94.5 ± 1.3	100 ± 0.0
`comp-2007-2-12`	200	10	1000	91.3 ± 1.2	100 ± 0.0
`comp-2007-2-13`	400	20	300	90.0 ± 1.2	100 ± 0.0
`comp-2007-2-14`	400	20	300	90.8 ± 0.9	100 ± 0.0
`comp-2007-2-15`	200	10	500	91.4 ± 1.3	100 ± 0.0
`comp-2007-2-16`	200	10	500	98.0 ± 0.8	100 ± 0.0

The neighbourhood operator used in this phase randomly selects two distinct cells $tt[a, b]$ and $tt[c, d]$ in the timetable and swaps their contents. When choosing the two cells it is necessary that $tt[a, b] \neq tt[c, d]$, thereby ensuring that two blank cells are not selected together (obviously, swapping two blank cells will result in an identical timetable). If the neighbourhood move causes a violation of hard constraints 1-4, then it is immediately rejected and reset; otherwise it is accepted, and tt is re-evaluated using C.

Note that an application of this neighbourhood operator results in one of two actions. The first occurs when two occupied cells are selected, which causes the places of the two associated events to be *swapped* in the timetable. The second occurs when one occupied and one blank cell are selected, causing just one event to be *moved* to a different place in the timetable. Note also that the probability of each of these actions occurring is directly related to the proportion of occupied cells in the timetable – assuming (without loss of generality) that n events are present in a timetable with $p = 45m$ places, then the probability of a swap occurring is:

$$P(\text{swap}) = \frac{n}{p} \times \frac{n-1}{p-1}, \tag{3}$$

(i.e. the conditional probability of selecting one occupied cell in the grid, followed by another, different occupied cell). The probability of performing a *move*, meanwhile, is:

$$P(\text{move}) = \left(\frac{p-n}{p} \times 1.0\right) + \left(\frac{n}{p} \times \frac{p-n}{p-1}\right), \tag{4}$$

(i.e the sum of the probability of selecting a blank cell followed by an occupied cell, and the probability of selecting an occupied cell followed by a blank).

Given the above cost function and neighbourhood operator, a straightforward application of SA is now used: starting at an initial "temperature" t_0, during execution the temperature variable is slowly reduced according to a temperature update rule $t_{i+1} = \alpha t_i$, where α ($0 < \alpha < 1$) is a variable known as the "cooling rate". At each temperature t_i, a Markov chain of length n^2 is then generated by performing n^2 applications of the neighbourhood operator. Any move that increases the cost of the timetable is then accepted with a probability $\exp(-\delta/t_i)$, where δ is the cost change that this move causes. Any move that reduces or leaves the cost unchanged, meanwhile, is accepted automatically.

Because this algorithm is time dependent, it is a good idea to choose a cooling rate α that allows Markov chains to be generated at as many temperatures as possible between the initial temperature t_0 and some end temperature. To calculate such a cooling rate the SA algorithm is first run for 5% of Stage 2's allocated time, and the number of Markov chains generated is recorded. This figure is then used to predict the number μ of Markov chains that will be generated in the remaining 95% of time. Using μ, we can then calculate a value for α that ensures that the temperature will be reduced from t_0 to a specific end temperature t_μ in exactly μ steps as:

$$\alpha = (t_\mu / t_0)^{1/\mu}. \tag{5}$$

In our case, following other works (van Laarhoven and Aarts, 1987; Abramson et al., 1996) the initial temperature t_0 is determined automatically by performing a small sample of neighbourhood moves and then calculating the variance of the cost over these moves. The end temperature t_μ, meanwhile, needs to be set by the user (see Section 4).

Finally, Stage 2 completes either when a timetable tt with a cost $C(tt) = 0$ is found, or when the time limit T_2 is reached. If the latter occurs, then the best solution found during this stage is taken, and events that are causing a violation of HC_5 are removed one-by-one until the timetable is seen to be completely free of any hard constraint violations. This resultant timetable is then passed on to Stage 3.

Table 3 summarises the effects of Stage 2 on the sixteen competition instances. Note that there is no observable correlation between the initial and final costs of the timetables. In total, twelve of the instances feature final costs with a mean and standard deviation close to zero, indicating that the procedure is able to successfully complete its objectives in the majority, if not all of these runs. Once again, the four that are causing the most difficulties are instances 1, 2, 9, and 10, which are the same "troublesome" instances that we met in Section 3.2. Perhaps one reason for this is that candidate solutions to these instances have a higher proportion of their cells occupied (up to 89% of available places) – thus, according to eq. (3), up to 79% of applications of the neighbourhood

Table 3: Cost of the timetable (using cost function C) at the start and end of Stage 2, and the number of extra events that are removed in order to make it free of all hard constraint violations. Presented figures are the means of 51 runs on each instance together with the standard deviation.

Instance	Start	End	Removed
comp-2007-2-1	18.6 ± 3.5	0.7 ± 0.9	0.7 ± 0.9
comp-2007-2-2	17.0 ± 3.8	2.1 ± 2.1	2.1 ± 2.0
comp-2007-2-3	10.8 ± 2.1	0.0 ± 0.0	0.0 ± 0.0
comp-2007-2-4	9.7 ± 2.0	0.0 ± 0.0	0.0 ± 0.0
comp-2007-2-5	69.3 ± 8.9	0.0 ± 0.0	0.0 ± 0.0
comp-2007-2-6	68.2 ± 5.2	0.0 ± 0.1	0.0 ± 0.1
comp-2007-2-7	9.1 ± 2.5	0.0 ± 0.0	0.0 ± 0.0
comp-2007-2-8	8.2 ± 2.2	0.0 ± 0.0	0.0 ± 0.0
comp-2007-2-9	21.3 ± 3.0	3.5 ± 1.9	3.4 ± 1.7
comp-2007-2-10	19.5 ± 2.8	5.5 ± 3.0	5.3 ± 2.9
comp-2007-2-11	8.0 ± 2.1	0.0 ± 0.0	0.0 ± 0.0
comp-2007-2-12	8.5 ± 2.9	0.0 ± 0.3	0.0 ± 0.3
comp-2007-2-13	64.9 ± 7.3	0.0 ± 0.2	0.0 ± 0.2
comp-2007-2-14	69.9 ± 7.0	0.1 ± 0.3	0.1 ± 0.3
comp-2007-2-15	9.2 ± 2.4	0.0 ± 0.0	0.0 ± 0.0
comp-2007-2-16	12.3 ± 2.4	0.0 ± 0.0	0.0 ± 0.0

operator result in a swapping action (in contrast with the remaining instances, where at most 19.7% of proposed neighbourhood moves are swaps). Of course, a swapping action is more likely to be rejected than a move, as both events need to be assigned to places that are suitable. It is possible that this higher rate of rejection will make movements in the search space more restricted, making exploration and resultant improvements in cost harder to achieve. We will return to this topic in Section 4.

3.4 Algorithm Description: Stage 3

In Stage 3, attention is turned towards satisfying the soft constraints of the problem. When this part of the algorithm is invoked, one of two situations will have occurred: either a valid timetable with a Distance to Feasibility of zero will have been produced or, after having spent 2/3's of the available run time trying to deal with the hard constraints, we will have settled for a timetable that has a number of unplaced events. In the latter case, these unplaced events are not considered any further – i.e. they are effectively eliminated from the problem.

The application of SA used here is again straightforward, with the cooling scheme being calculated in the same way as Stage 2. The neighbourhood operator is also the same as Stage 2, though in this case, moves that cause a violation of hard constraint HC_5 (in addition to the previous four) are also immediately reset. Finally, the cost function used here is simply the Soft Cost (see Section 2), which is appropriate due to the fact that no hard constraint violations are permitted in the timetable from this point onwards.

4 Implementation and Analysis of Final Results

The algorithm was implemented using C++ under Linux using the g++ 4.1.1 compiler under the `-O3` optimisation option. All experiments were run on a 1.8GHtz machine with 256MB RAM, which was granted a time limit 636 seconds by the competition benchmarking program. Because many of this algorithm's parameters are calculated automatically, only two values needed to be chosen for these experiments: I, the iteration limit used for ITERATED-HEURISTIC-SEARCH in Stage 1, and the end temperature t_μ for the two annealing phases. In practice, the algorithm did not seem to be particularly sensitive to variations in I, providing that values of around $100n$ or more were used.[3] Values of I less than this tended to cause the process to stagnate too readily. Considering the end temperature t_μ, if this was set too high then it tended to mean that too many increases in cost were permitted throughout the run, making the search more of a random walk. On the other hand, if t_μ was too low, then the algorithm spent too much time at low temperatures, making it more greedy and increasing its probability of getting stuck in a local minimum. On the whole, however, the algorithm did not seem too sensitive to variations in t_μ, provided that values of around 0.000005 to 0.0001 were used. For all experiments here, parameter settings of $I = 1000n$ and $t_\mu = 0.00001$ (for both annealing phases), were used. Note that no fine tuning of these values was conducted.

Table 4 summarises the final results achieved by this algorithm after performing 51 runs (from different random seeds) on each instance. Because timetable quality is ranked according to a pair of values (see Section 2), results are summarised using the best, worst, and median, and lower and upper quartiles (we have used fifty-one runs here so that these statistics can be calculated without the need for interpolation, which would be inappropriate). We can see that the algorithm has achieved feasibility for all instances at least once. Feasibility has also been achieved in all runs with eight of the sixteen instances. For instance-8 a perfect solution has also been found in one run. Following on from observations made in previous sections, once again instances 1, 2, 9, and 10 prove to be the hardest problems to solve, with feasibility being found in just 45%, 22%, 2%, and 2% of runs respectively. Note that the algorithm is able to find feasibility in $> 95\%$ of runs with all remaining instances, however.

Earlier, in Section 3.3 it was suggested that when considering instances in which a large proportion of neighbourhood moves were being rejected, movements in the search space would be more restricted, possibly making improvements to the candidate solution more difficult to achieve. At this point, a relevant question to now ask is whether the same feature also applies when performing optimisation according to the Soft Cost. To investigate this, in trials where a distance to feasibility of zero was achieved, we recorded the proportion of all moves that were accepted during Stage 3, together with the proportion by which the Soft Cost was ultimately reduced.[4]

[3]The value n was used to allow the setting to scale with instance size.

[4]We remember that for Stage 3 a move is "accepted" if the alteration to the timetable does not cause a violation of *any* of the hard constraints.

Table 4: Summary of the final results obtained from 51 runs on each problem instance. In each case the Distance to Feasibility is presented together with the associated Soft Cost (in brackets).

Instance	Best	Q_1	Median	Q_3	Worst
`comp-2007-2-1`	0 (1294)	0 (1600)	17 (1492)	32 (1693)	105 (1944)
`comp-2007-2-2`	0 (1599)	18 (1718)	46 (1826)	80 (2016)	213 (2176)
`comp-2007-2-3`	0 (278)	0 (416)	0 (457)	0 (523)	0 (664)
`comp-2007-2-4`	0 (388)	0 (538)	0 (589)	0 (644)	0 (761)
`comp-2007-2-5`	0 (22)	0 (123)	0 (193)	0 (268)	0 (638)
`comp-2007-2-6`	0 (369)	0 (606)	0 (696)	0 (767)	20 (708)
`comp-2007-2-7`	0 (74)	0 (300)	0 (421)	0 (529)	0 (890)
`comp-2007-2-8`	0 (0)	0 (162)	0 (206)	0 (256)	0 (366)
`comp-2007-2-9`	0 (1582)	59 (1829)	80 (2312)	120 (1864)	214 (1609)
`comp-2007-2-10`	0 (2380)	83 (2339)	126 (2262)	194 (2303)	372 (2159)
`comp-2007-2-11`	0 (344)	0 (456)	0 (541)	0 (605)	0 (800)
`comp-2007-2-12`	0 (486)	0 (660)	0 (741)	0 (852)	125 (710)
`comp-2007-2-13`	0 (365)	0 (538)	0 (631)	0 (707)	19 (766)
`comp-2007-2-14`	0 (222)	0 (558)	0 (660)	0 (786)	27 (685)
`comp-2007-2-15`	0 (266)	0 (301)	0 (344)	0 (366)	0 (455)
`comp-2007-2-16`	0 (99)	0 (165)	0 (194)	0 (215)	0 (265)

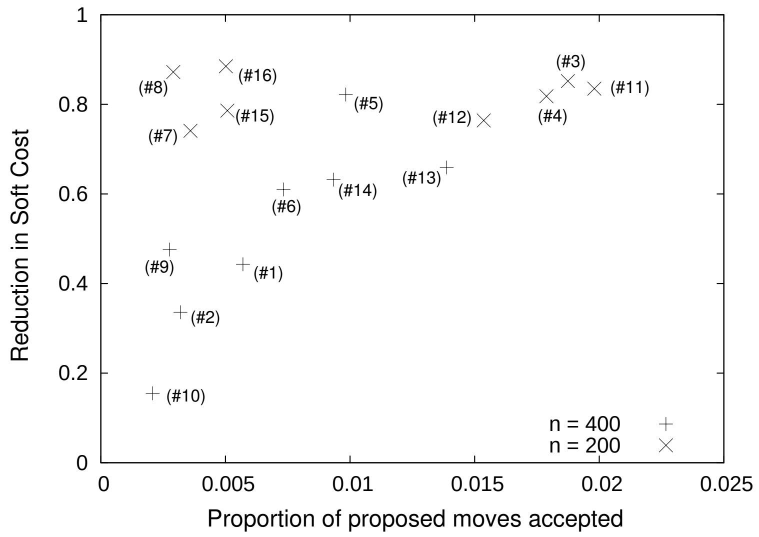


Figure 3: Scatter diagram showing the relationship between the proportion of accepted neighbourhood moves in Stage 3, and the resultant reduction in the Soft Cost (expressed as a proportion). Each point in the graph is averaged across all runs with a particular instance where a distance to feasibility of zero was achieved.

The statistics collected here are displayed in fig. 3 where a weak positive, though statistically significant, correlation between the two variables can be observed (a correlation of $r = 0.504$, using a two-tailed test at the 5% level). Curiously, one particular group of smaller instances – namely instances 7, 8, 15 and 16, which all have $n = 200$ and $s = 500$ – seems to go against this trend, with the algorithm experiencing a low proportion of accepted moves and yet still achieving large reductions in the Soft Cost. Also, if we only consider the eight instances for which $n = 400$, then the correlation rises to a significant $r = 0.8$, though with the $n = 200$ instances, there seems to be no observable correlation (with $r = 0.03$). From these admittedly limited results, we propose that there is *some* causation between the restrictiveness of the search space and the improvements in cost that are achievable, though there are certainly other factors that will have an effect here, such as the shape of the cost landscape, the amount of time that is allocated to this stage, and the amount of computation that is required for each application of the evaluation function.

5 Conclusions

In this paper, a three stage metaheuristic-based algorithm has been presented for the post enrolment-based course timetabling problem used in track-2 of the Second International Timetabling Competition. Crucially this algorithm is time-dependent, meaning that it is able to alter the intensity of the search conducted in accordance with the amount of available run time granted.

We performed a comparison between this algorithm and the five official finalists of the competition by performing ten runs on all available instances, including the ten "hidden" instances, which are not currently in the public domain. Incorporating these results into the ranking process used for choosing the competition winner reveals that our algorithm comes in 6th place with a rank-average of 39.5.[5] For reference, the resultant rank-averages of the five finalists (with these results included) are, in order, 1st place = 14.8; 2nd = 28.0; 3rd = 31.7; 4th = 33.5; and 5th = 35.5. Full results and further details of these experiments are detailed in Appendix A.

One issue with this algorithm is that for Stages 2 and 3, in order to ensure that the constraints satisfied in previous stages are not re-violated, the proposed neighbourhood operators are restricted so that moves causing such violations are automatically rejected. In Stage 3, for example, this means that the algorithm only searches in the space of feasible solutions. However, in this case, by using a restricted neighbourhood operator, there is no guarantee that all feasible solutions will communicate with one another. In other words, the feasible-only search space may, in effect, be split into a number of disjoint subspaces, with areas of non-feasibility – that cannot be traversed using the current neighbourhood operator – occupying the space in-between. A practical implication of this is that even if the algorithm is granted infinite computation time, there will be no guarantee of finding an optimal (i.e. perfect) solution, despite the fact that these problem instances are all known to feature at least one. A future avenue of research might therefore be to look at how such non-feasible spaces can be traversed – or at least circumnavigated – using, for example, other

[5]Refer to the ITC2007 website for full listings of the other algorithms' results and for details of this ranking process

types of neighbourhood operators (such as the Kempe-chain operator, proposed by Thompson and Dowsland (1998)) or by using some sort of macro-perturbation scheme to help “jump” across large areas of the search space (used, among others, by Di Gaspero and Schaerf (2006)).

References

Abramson, D., Krishnamoorthy, H., Dang, H., 1996. Simulated annealing cooling schedules for the school timetabling problem. Asia-Pacific Journal of Operational Research 16, 1–22.

Brelaz, D., 1979. New methods to color the vertices of a graph. Commun. ACM 22 (4), 251–256.

Di Gaspero, L., McCollum, B., Schaerf, A., August 2007. The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Tech. Rep. QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0/1, School of Computing, Queens University, Belfast.

Di Gaspero, L., Schaerf, A., 2006. Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modeling and Algorithms 5 (1), 65–89.

Kirkpatrick, S., Gelatt, C., Vecchi, M., 1983. Optimization by simulated annealing. Science 4598, 671–680.

Kostuch, P., 2005. The university course timetabling problem with a 3-phase approach. In: Burke, E., Trick, M. (Eds.), Practice and Theory of Automated Timetabling (PATAT) V. Vol. 3616. Springer-Verlag, Berlin, pp. 109–125.

Lewis, R., 2006. Metaheuristics for university course timetabling. Ph.D. thesis, School of Computing, Napier University, Edinburgh.

Lewis, R., Paechter, B., McCollum, B., August 2007. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Working Papers in Accounting and Finance A2007-3, Cardiff Business School, Cardiff University, ISSN: 1750-6658.

McCollum, B., McMullan, P., Burke, E., Parkes, A., Qu, R., September 2007. The second international timetabling competition: Examination track. Tech. Rep. QUB/IEEE/Tech/ITC2007/Exam/v4.0/17, School of Computing, Queens University, Belfast.

Rossi-Doria, O., Samples, M., Birattari, M., Chiarandini, M., Knowles, J., Manfrin, M., Mastrolilli, M., Paquete, L., Paechter, B., Sttzle, T., 2002. A comparison of the performance of different metaheuristics on the timetabling problem. In: Burke, E., De Causmaecker, P. (Eds.), Practice and Theory of Automated Timetabling (PATAT) IV. Vol. 2740. Springer-Verlag, Berlin, pp. 329–351.

Socha, K., Samples, M., 2003. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In: Evolutionary Computation in Combinatorial Optimization (EvoCOP 2003). Vol. 2611. Springer-Verlag, Berlin, pp. 334–345.

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

van Laarhoven, P., Aarts, E., 1987. Simulated Annealing: Theory and Applications. Kluwer Academic Publishers, Reidel, The Netherlands.

A Comparison with Competition Results

Table 5 shows the results achieved by our algorithm within the specified time limit in a random sample of ten runs on each problem instance (including the eight hidden instances, labelled `comp-2007-2-17` to `24`). Note that the algorithm was executed "blindly" on the hidden instances – that is, all refinements were made to our implementation *before* performing runs on the hidden instances. This is consistent with the process used for officially ranking the finalists of the competition.

Table 5: Results used in the comparison with the competition entries. For each instance, results are ordered from best (1) to worst (10).

Instance	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
`comp-2007-2-1`	0 (1496)	0 (1525)	0 (1585)	0 (1612)	0 (1705)	25 (1579)	32 (1693)	32 (1770)	58 (1707)	76 (1634)
`comp-2007-2-2`	0 (1770)	0 (1927)	0 (1995)	18 (1900)	28 (2045)	31 (2163)	53 (1821)	56 (1727)	78 (2139)	114 (1986)
`comp-2007-2-3`	0 (416)	0 (435)	0 (455)	0 (457)	0 (466)	0 (491)	0 (523)	0 (523)	0 (553)	0 (635)
`comp-2007-2-4`	0 (455)	0 (541)	0 (543)	0 (575)	0 (588)	0 (612)	0 (623)	0 (639)	0 (691)	0 (761)
`comp-2007-2-5`	0 (22)	0 (82)	0 (114)	0 (189)	0 (245)	0 (252)	0 (283)	0 (482)	0 (540)	0 (541)
`comp-2007-2-6`	0 (409)	0 (453)	0 (528)	0 (592)	0 (605)	0 (652)	0 (705)	0 (719)	0 (749)	0 (873)
`comp-2007-2-7`	0 (108)	0 (303)	0 (304)	0 (381)	0 (384)	0 (445)	0 (450)	0 (498)	0 (605)	0 (657)
`comp-2007-2-8`	0 (101)	0 (122)	0 (172)	0 (186)	0 (187)	0 (187)	0 (230)	0 (243)	0 (283)	0 (366)
`comp-2007-2-9`	43 (1810)	56 (1868)	59 (1829)	59 (1938)	102 (1907)	116 (2162)	129 (1813)	138 (1960)	175 (1764)	187 (1782)
`comp-2007-2-10`	57 (2065)	79 (2445)	80 (2233)	95 (2336)	101 (2150)	125 (2174)	158 (2334)	162 (2404)	172 (2469)	273 (1841)
`comp-2007-2-11`	0 (344)	0 (411)	0 (419)	0 (428)	0 (433)	0 (501)	0 (522)	0 (582)	0 (606)	0 (657)
`comp-2007-2-12`	0 (504)	0 (629)	0 (653)	0 (725)	0 (727)	0 (728)	0 (741)	0 (765)	0 (950)	125 (710)
`comp-2007-2-13`	0 (428)	0 (504)	0 (526)	0 (588)	0 (607)	0 (628)	0 (631)	0 (647)	0 (674)	19 (766)
`comp-2007-2-14`	0 (394)	0 (470)	0 (558)	0 (609)	0 (619)	0 (633)	0 (667)	0 (762)	0 (795)	0 (796)
`comp-2007-2-15`	0 (271)	0 (281)	0 (299)	0 (300)	0 (301)	0 (334)	0 (344)	0 (374)	0 (389)	0 (415)
`comp-2007-2-16`	0 (138)	0 (194)	0 (196)	0 (200)	0 (210)	0 (219)	0 (220)	0 (221)	0 (223)	0 (229)
`comp-2007-2-17`	0 (0)	0 (1)	0 (10)	0 (22)	0 (39)	0 (47)	0 (51)	0 (54)	0 (77)	0 (101)
`comp-2007-2-18`	0 (28)	0 (47)	0 (90)	0 (197)	0 (224)	0 (225)	0 (274)	0 (285)	0 (359)	0 (421)
`comp-2007-2-19`	101 (2039)	133 (2020)	212 (1872)	227 (1902)	231 (2135)	275 (1741)	322 (2055)	356 (1746)	400 (1940)	493 (1860)
`comp-2007-2-20`	0 (735)	0 (761)	0 (847)	0 (853)	0 (865)	0 (865)	0 (869)	0 (892)	0 (953)	0 (978)
`comp-2007-2-21`	0 (361)	0 (395)	0 (501)	0 (501)	0 (505)	0 (510)	0 (527)	0 (534)	0 (546)	0 (590)
`comp-2007-2-22`	719 (1547)	800 (1679)	857 (1613)	1066 (1300)	1066 (1504)	1168 (1318)	1181 (1272)	1234 (1097)	1259 (1549)	1305 (1188)
`comp-2007-2-23`	207 (4402)	232 (3787)	247 (3756)	305 (4532)	330 (3687)	334 (4474)	375 (4353)	378 (3518)	827 (3683)	943 (3718)
`comp-2007-2-24`	0 (1003)	0 (1199)	0 (1200)	0 (1297)	25 (969)	64 (1066)	66 (1177)	94 (1349)	96 (1093)	169 (994)