Timetabling Competition - SA-based Heuristic

P A Kostuch
January 2003

Abstract: This text describes the model used in the International Timetabling Competition 2003 set by the Metaheuristics Network and presents a heuristic approach towards the solution of these problems. The approach is divided into two stages: at first, a feasible timetable is constructed and afterwards Simulated Annealing is used to improve the timetable according to an objective function value. The text ends with an outlook on possible changes and improvements.

1 The Timetabling Competition model

The model used for the timetabling competition is the following: we are given a set of events that demand certain students and a room of some minimum size that is equipped with a set of features. There are 45 time-slots, with 9 time-slots forming a day, all students and all rooms are available in each time-slot. Events take exactly one time-slot and release all resources afterwards.

The hard constraints are: no students required to attend two events in the same time-slot, every event does have a room satisfying the needed specifications and there is maximal one event per room per time-slot.

The soft constraints are: no student has only one event per day (days that are completely free are acceptable), there are no students that attend an event in the last slot of a day and no student has to attend more than two events in a row on one day (consecutive overnight events do not count). If broken every constraint adds 1 penalty point to the objective function. For the constraint dealing with consecutive events 3 events in a row incur 1 penalty point with another penalty point added for every further consecutive event.

The task is to find feasible timetables with low penalty values for 20 different instances. It is known that there are feasible solutions with penalty 0 for all instances. There are two further restriction on the solutions: they must all be obtained by the same algorithm and the running time is restricted to what amounts to 432 seconds on the machine used for the algorithm described here. Although the algorithm itself may not be changed it is allowed to run it with different random seeds and to submit the the best solutions, i.e. specifying a different random seed

for every instance.

2 The algorithm

A comprehensive description of the algorithm is given in this chapter. The outlines of the programme are as follows: At first the problem description is compiled, then an initial feasible solution is constructed disregarding the objective function. Next, the sequence of the formed time-slots is changed in order to reduce the penalty score using simulated annealing and finally we allow pairs of events in different time-slots to be exchanged (if feasibility is preserved) in order to further reduce the objective function, again using simulated annealing.

2.1 Data handling and Preprocessing

The input file for every instance contains the number of events, rooms, different features and students. Next are the sizes for the different rooms, then a list of features satisfied by the rooms, the list of features needed by the different events and finally for every student a list of the events s/he wishes to attend.

The number of students, rooms and events are generic. The remaining information will be compressed into two matrices called the *incidence* matrix and the *event-room* matrix.

The event-room matrix is calculated first. It is a boolean matrix with dimension events*rooms and row i indicates which rooms are feasible for event i. For the initialisation of the (i,j) entry we check for all features whether event i demands it and if so whether room j provides it. If this is the case entry (i,j) is true otherwise it is false. The next step is to calculate the size of every event using the student attendance information. With the event-size information we modify the event-room matrix: every entry were the room-size is smaller than the event-size is set to false. This gives us the final event-room matrix.

Afterwards we calculate the incidence matrix. It is a boolean matrix with dimension events*events and entry (i,j) indicates whether events i and j can be placed in the same time-slot (it essentially is the edge matrix for a graph colouring problem). It is initialised by simply checking for every entry (i,j) whether there is a student wishing to attend both events. The matrix is altered in a second step using information form the event-room matrix: we search for events with only one true entry in its row (referring to an event that can only take place in one room). We

Instance	1	2	3	4	5	6	7	8	9	10
events	400	400	400	400	350	350	350	400	440	400
rooms	10	10	10	10	10	10	10	10	11	10
features	10	10	10	5	10	5	5	5	6	5
students	200	200	200	300	300	300	350	250	220	200
$\overline{\text{events per student}}$	17.76	17.23	17.71	17.43	17.77	17.77	17.47	17.58	17.36	17.78
average degree	81.34	83.52	94.23	90.60	109.26	91.01	72.43	68.06	75.30	80.36
Δ	140	151	139	168	193	174	196	151	146	133
$\overline{\text{room-options}}$	1.96	1.92	3.42	2.45	1.78	3.59	2.87	2.93	2.58	3.49
1-option events	133	208	222	124	262	19	115	93	129	40
edges added	3488	4600	6472	2412	5944	228	1432	1620	3684	1274

Table 1: Summary statistics for the first 10 instances

then form groups of those events that have the same room as their only option. It is evident that events in the same group cannot take place in the same time-slot and therefore the incidence matrix is updated such that the events in one group form a clique.

Provided that a significant number of events have only one room option this step seems to be very helpful since without inclusion of this information in the incidence matrix finding an assignment satisfying these additional constraints becomes a matter of trial-and-error.

Table 1 gives an overview of some summary statistics for the first 10 instances used. Once we have derived results on the different instances we will try to explain differences in the performance by looking at the difference between the instances as captured in the summary statistics.

As a conclusion of this section we will give a first ad-hoc interpretation of these values and their likely effect on the timetabling problem. The first important information is the product of rooms and time-slots (for all instances we have 40 ordinary time-slots and 5 which are feasible but penalised) in comparison to the number of events. We see that the number of 40 time-slots (which can always be achieved since the problems have perfect solutions) is tight in most cases but instances 5 to 7 show some slack which is likely to facilitate finding an initial feasible assignment. The number of different features is correlated with the average number of room-options. The tendency is that the more features the less options which is intuitively clear as it means increasing diversity in the demand patterns. The number of students varies considerable over the instances and since the average number of events per student is almost constant over the instances it is

conceivable that the value of the objective function is correlated with the number of students as students are the sole source of all penalty points.

For the graph colouring problem the information about the average degree in the graph given by the incidence matrix and the maximum degree is relevant. It is likely that high average and maximum degrees aggravate the problem. The problem is also likely to be harder as the average number of room-options per event decreases as it makes finding a maximum matching within a time-slot comprising all events assigned to it harder. The last two rows show that the effect of added edges in the incident matrix is considerable in almost all examples.

2.2 Finding an initial feasible assignment

With these preparations completed we can enter the next phase where we try to construct an initial feasible timetable using graph colouring and maximum matching algorithms. In this phase we will try to assign events to time-slots and within the time-slots to rooms in such a way that no hard constraints are violated. In the first phase we will attempt to achieve this goal using only time-slots without penalty thus implicitly catering for the soft constraint involving end of day events. If this fails we open up the yet unused but available slot.

2.2.1 Initial attempt

We start by using a sequential colouring algorithm for the graph given by the incidence matrix. The order in which events are considered for colouring is the degeneracy order. The assignment of a slot (which we identify with the colours) to the event under consideration works in the following way: we determine all those slots among the initial 40 slots that are possible for the event, i.e. there is no colour conflict with an already placed event. Among these options we assign a slot with the minimal number of events in it provided the number of events in this time-slot does not exceed the number of rooms. Events that cannot be assigned to a time-slot are placed in a pool of unassigned events for further consideration.

After this initial assignment of time-slots we run a maximum matching algorithm for bipartite graphs for each time-slot. The bipartite graph is the subgraph of the event-room matrix where we only us those rows that correspond to the events assigned to the time-slot under consideration. Events that do not get a room in a maximum matching are removed from the list of events in this time-slot and go into the pool of events that need further consideration.

2.2.2 Improvement attempt

We call the next phase *improvement attempt* and describe it here is full detail. It will be used in unchanged form in later phases. The idea is that we take every unplaced event and scan the 40 time-slots formed in the first phase: first we check whether there is a colour-conflict with a given time-slot and if there is none whether we can add the event to the time-slot such that a maximum matching algorithm can assign rooms to this event and to all events that were previously in the time-slot. If this is possible the event is assigned the time-slot and room as determined above and leaves the pool of unassigned events.

There are two mechanisms why this may work:

- events that could not find a time-slot and therefore entered the pool during the colouring phase of the initial attempt may now fit in some time-slots where events were removed during the room assignment phase.
- events removed during the room assignment phase may well fit into other time-slots.

2.2.3 Shuffling

For the events still left there is no way of placing them unless we change existing assignments. We call the first step in this direction *shuffling*. For a fixed number of repetitions (in the algorithm used this is set to be 50000) we look at every event from the pool of unplaced events and choose a time-slot at random. First we check whether there is a colour-conflict between this element and the chosen time-slot. If this is not the case we provisionally assign the event to this time-slot and run a maximum matching algorithm. If the number of events that are assigned to a room is one larger than the initial number of events in this time-slot we have managed to place an unassigned event. If the number of events with rooms is unchanged we take the one event without room and put it into the pool of unplaced events (note that the number of events with feasible room assignment cannot drop below the initial number). The hope is that in such a case the newly unassigned event is different from the initially unassigned event. If this is the true we run the procedure called improvement attempt (see 2.2.2) for this event before placing it in the pool of unassigned events hoping that it fits into a different time-slot.

To increase our chance that the initially unassigned event is part of the new maximum matching we randomise the order in which nodes appear in the maximum matching algorithm.

	Phase 1	Phase 2	Phase 3
restarts	2000	2000	100
blow-up steps	1	2	6
shuffle steps	5000	5000	20000

Table 2: Parameters for blow-ups

2.2.4 Blow-ups

The events that are left can be considered as problematic and we have to go one step further in using invasive measures that change the assignments made so far. The idea here is that we force the placement of an unassigned event in an existing time-slot by first removing all events from this time-slot (*blow-up*) and then we place the unassigned event in the now empty time-slot. Next we fill up the time-slot with those removed events that do not have a colour-conflict with the initially unassigned event. Then we assign rooms to the events and place those events that had a colour-conflict or that were not given a room in the pool of unassigned events.

This way we accept changes that may lead to an increased number of unplaced events. After this change we apply improvement attempt to the unassigned events and afterwards we run a number of repetitions of the shuffling algorithm. After a certain number of blow-up steps (i.e. blow-ups combined with improvement attempt and shuffling) we restart the the whole procedure at the assignment that was the best so far in terms of unassigned events.

In our algorithm we have three different sets of parameters that are displayed in Table 2. Note that the first phase allows for only one blow-up step and is therefore of a greedy nature while the other two phases accept a worse timetable as a new starting position. As the number of blow-up step increases the time requirement goes up steeply and we therefore only run 100 restarts in the last phase.

2.2.5 Opening the last slots

Events that are not placed despite the above efforts are distributed over the last five slots using the same algorithms as in Chapter 2.2.1. If this does not place these events conflict-free in the 5 new time-slots we could use the same methods as above in order to overcome problems but in practice this proves unnecessary as the number of events left after the blow-ups are for all instances and all random seeds below 5 which guarantees finding a feasible initial assignment.

Instance	1	2	3	4	5	6	7	8	9	10
events	400	400	400	400	350	350	350	400	440	400
initial attempt	78	59	43	63	21	31	19	65	72	84
improvement attempt	46.6	38.1	31.1	44.3	20.3	24.4	12.2	44.4	40.0	55.4
shuffling	8.3	9.2	13.8	18.4	16.8	10.7	4.6	7.8	6.2	9.8
blow-up 1	0.3	0	2.7	1.3	3.28	0	0	0.1	0	1.8
blow-up 2	0.2	0	2.04	0.4	0.64	0	0	0	0	1.4
blow-up 3	0.1	0	2	0.4	0.64	0	0	0	0	1.2
time	40.5	11.0	170.8	92.9	103.2	3.6	2	15.4	5.2	166.2

Table 3: Results for finding initial feasible assignment

2.3 Initialisation results for the first 10 instances

The number of events left after each phase for each instance averaged over 25 different random seeds can be seen in Table 3. Our first observation is that we find feasible assignments using only 40 slots for half the instances for all random seeds. For 3 more instances we find at least for some random seeds feasible assignments and even for the 2 instances where we could not find such a timetable the number of events that have to go into end of day time-slots is very small. We also see that the last blow-up phase does not contribute significantly to the performance. It may therefore be dropped to save computation time. Note that the variation between different random seeds in the results for the first improvement attempt is due to the randomised matching algorithm.

As mentioned earlier we now try to relate these results to the characteristics of the different instances. It is clear that our statistics from Table 1 do not capture all aspects of the instances that are relevant for the initial assignment problem, e.g. size of the largest clique, but we still hope to explain some observed differences. As explanatory variables we focus on the average and maximum degree in the graph, on the number of room-options available and whether the 40 slots are a lower bound for a feasible timetable (which is not the case for instances 5 to 7).

The first result we try to explain using linear regression is the number of unassigned events after the shuffling phase. The model including all 4 variables has a R^2-value of 0.94 and the only non-significant term is the number of room-options with a p-value of 0.051. As expected increasing average and maximum degree increase the number of events left while the presence of slack (i.e. 40 time-slots are no lower bound) and an increase in room-options decreases the number of events left. It has to be mentioned that when the the room-option variable is dropped slack and maximum degree become insignificant as well and the average degree

remains as the only significant explanatory variable with a R^2-value of 0.68.

Explaining the values after the blow-up phases fails and we conclude that the our summary statistics for the instances cannot be used to predict the final results of the initial assignment phase. This is either due to the fact that important information is not included, e.g. the size of the maximum clique, or that the differences are indeed just due to random fluctuation and not the result of differences in the instances.

2.4 Using SA to improve the objective function

Having found a feasible assignment which does in the majority of cases satisfy the soft-constraint of not using end of the day slots we turn our attention towards the other two soft-constraints, namely students with single events on a day and students with more than two events in a row. In the heuristic presented here we adopt a greedy point of view refusing to give up any of our previous achievement. In our case this means that we do not allow changes in the initial timetable that lead to infeasible assignments nor do we accept moves that would place events in end of day slots (in those case where such events exist we do not accept an increase in their number but do not control for their sizes).

2.4.1 Sequencing the time-slots

In the first step we take the time-slots formed during initialisation and do not attempt to change them. What we care about is their sequence. At every fixed temperature we systematically try to exchange every possible pair of time-slots. If the exchange reduces the objective function the move is accepted. If the result is an increase in the objective function value we accept the move with a probability that decreases with the size of the deterioration and the progress of the algorithm.

The exact parameters can be seen in Table 4. We note that this simulated annealing problem has a connected solution space and that the neighbourhoods are equi-sized. Our selection process is not the standard one since we choose to scan the neighbourhood deterministically instead of randomly. We chose this approach to reduce computation time since we can be sure that every possible pair exchange is considered at least once per temperature without running a large number of random steps. This is acceptable since we see this step only as a supportive measure for the main part of our programme which is described in the next section. It is not desirable to spend much computation time in order to improve the result of this phase as it is very likely that by fixing the time-slots in this feasible but arbitrary

Different temperatures	$N = 0..89$
Attempted exchanges per temperature	$O(slots^2)$
Acceptance probability	$e^{-\Delta * s}$
Value of s	0.15+.015*N
Probability for $\Delta = 1$	0.86 (N=0);0.44 (N=45); 0.23 (N=89)

Table 4: Parameters for the Sequencing

Different temperatures	$N = 0..319$
Attempted exchanges per temperature	$O(events^2)$
Acceptance probability	$e^{-\Delta * s}$
Value of s	0.25+.005*N
Probability for $\Delta = 1$	0.77 (N=0);0.35 (N=160); 0.16 (N=319)

Table 5: Parameters for the Exchange

way the optimal sequence will have an objective function value far away from the overall achievable optimum of 0.

2.4.2 Exchanging pairs of events

This phase forms the major part of our algorithm, both in terms of time consumption as well as impact. The basic idea is again a local search guided by simulated annealing. The starting position in the solution space will be the best timetable found by sequencing. The neighbourhood is defined by all timetables where exactly one pair of events has swapped their time-slot. We only accept moves where this swap does not lead to colour-conflicts nor to events that cannot be assigned a room within their time-slot. Thus we only search the space of feasible timetables. In choosing the next neighbour we again use a deterministic scheme: for every temperature we systematically try to exchange all pairs of events. This search order is fixed and does not depend on the acceptance or rejection of moves.

The exact parameters can be seen in Table 5. The acceptance temperature has been reduced in comparison with the sequencing since the number of students involved in every move is much smaller here and therefore the amount of deterioration possible is much smaller. Note that some general assumptions about good neighbourhoods in local search are violated with our approach, mainly connectivity of the search space but also neighbourhood sizes which are far away form being uniform. In choosing the next neighbour we do not apply the standard random selection but proceed deterministically.

Different temperatures	$N = 0..99$
Attempted exchanges per temperature	$O(events^2)$
Acceptance probability	$e^{-\Delta * s}$
Value of s	2.5+.1*N
Probability for $\Delta = 1$	0.08 (N=0);0.05 (N=50); 0.03 (N=99)

Table 6: Parameters for the Greedy search

2.4.3 Greedy search

As a final step in order to improve the timetable we run the simulated annealing search with lower acceptance probability starting with the best solution found so far until the time limit is reached. By reducing the temperature we confine the search to the vicinity of this solution and effectively perform a greedy search on the solutions. The values are displayed in Table 6.

2.5 Objective function results for the first 10 instances

The values for the objective function at the end of each phase can be seen in Table 7. We give the values of the best run among 25 different random seeds used. We can see that the results obtained on the different instances vary considerably. Since students are the source of penalty points it is reasonable to investigate the correlation between the objective function value and the number of students. The correlation coefficients are: 0.96 for the initial value, 0.93 after sequencing, 0.06 after exchanging and 0.03 for the final value. We take this as a sign that the pairwise exchange method is indeed the most important step in the algorithm and that its performance is influenced by other characteristics.

We now try to explain the differences in the final value with the following 5 variables: slack (i.e. whether 40 slots are a lower bound), average and maximum degree, room-options, number of students. Regression on all these variable yields only the average degree as a significant term. This result is confirmed after stepwise reduction. The final model includes only the average degree, has a R^2-value of 0.56 and the p-value for the model is 0.01. As expected the objective function value rises with increasing average degree. The low R^2-value again indicates that we have not the right set of statistics to capture the differences between the instances.

Our last observation are the run-times of the algorithm. A regression using the average degree and the number of students has a R^2-value of 0.80 with both terms

Instance	1	2	3	4	5	6	7	8	9	10
initial assignment	515	444	491	734	900	771	913	646	502	516
sequencing	226	203	239	339	318	340	327	285	239	244
exchange	98	62	107	191	195	32	28	32	44	104
greedy search	**95**	**53**	**105**	**185**	**186**	**22**	**15**	**12**	**25**	**103**
runtime	263	265	346	446	293	326	739	574	492	373

Table 7: Results for the objective function value

highly significant. An increase in average degree means a strong decrease in runtime as the number of possible moves goes down due to more colour conflicts and an increase in students means an increase in runtime. This shows that the evaluation of the new timetables drives the cost of the algorithm.

3 Further improvements

While the ability to find a feasible solution for every instance and every seed is already an achievement in itself there is no way of evaluating the quality of the solutions before the end of the competition. And although the objective function value has already been greatly reduced in comparison with early approaches there is still some room for testing changes and introducing new ideas. This chapter highlights some further steps that could be taken.

3.1 Computational aspects

The changes suggested here have no direct impact on the quality of the objective function value but it is hoped that they would reduce the run-times for every instance and thereby allowing more evaluations eventually leading to better timetables.

- Currently the maximum matching algorithm starts with the empty matching. But as a matter of fact there is already a large known matching after the exchange of pairs of events (the old maximum matching without the node that was moved to another time-slot). Using this matching as a start position we should be able to determine much faster whether the added event can be included in a new maximum matching as there can be at most one augmenting path.

- From the analysis of the run-times it was clear that the evaluation of timetables is a major factor in driving the complexity. Currently every new timetable is evaluated from scratch but the changes under the pair exchange are actually marginal. At least 3 out of 5 days are not affected at all and therefore do not have to be reevaluated. But there might be even more potential to cut time by applying some reasoning how the small change made can affect the value of the timetable. It is hoped that this would reduce computation time dramatically.

The second aspect of computational changes that should be implemented refers to the information acquired during the execution. By collecting some more data we may get a better insight into the problems.

- We do not have the correct summary statistics for the problem instances. The major problem is that the average number of room-options does not feature as significant in our models. The assumption that the room assignment is therefore not causing additional difficulty in finding a good timetable is certainly wrong. To show this we ran the same algorithm without checking that rooms were available for all events in a time-slot. This of course led to the acceptance of many more moves. The results were clear: 8 instances could be solved with values around 10 (six of them were one digit results). So the room assignment is a problem. (Most interestingly there was no significant improvement for instance 5 where the final value was 173). We therefore assume that the average number of room-options which is in itself a summary statistics of the events does not capture the important aspects of the problem. It may be interesting to look for different summary statistics describing the room-option situation, e.g. treating all events that can go in more than half of the rooms the same and emphasising those with only one or two options.
- As an intermediate information between the description of the problem instances and the performance of the algorithm we could collect statistics about the number of moves accepted during the exchange phase (how many improved the value/did not change the value at all/were accepted because of probability), and about the moves rejected (whether this was because of colour conflict, problems with the room assignment or because of the resulting objective function value).

3.2 Changing the existing algorithm

Here we suggest changes that do directly aim at the performance measured by the objective function.

- Concentrate on one or two instances as a testbed for changes.
- There is some indication that we can drop the last blow-up phase to save time without impairing the performance.
- We could try to take the objective function value into consideration when constructing the initial timetable. We could also think about ways to construct a different heuristic that uses the resource allocation model.
- Once we can predict the difficulty of an instance it is desirable to adjust the SA parameters so that we use the maximum allowed time on all instances.
- There are events with no students in them. We could move them to the end of day time-slots without incurring any penalty. That would free rooms in the important time-slots.
- To increase the flexibility we can create dummy events, i.e. new events without students that can go in every room, and place them in the last time-slots so that the algorithm can use them in the exchange phase. The problem is clearly that this increases the number of moves and thereby run-times. in our runs we created 10 dummy events and placed 2 in every end of day time-slots. Note that this is important as the algorithm cannot change the number of events in a time-slot. This has to be seen particularly critical in those instance with slack as the formation of full time-slots is desirably in order to reduce penalty. The run-times for this test exceeded the allowed 432 seconds in half the instances (sometimes dramatically, i.e. 1500 seconds). We hope to control this by making exchanges involving dummy events harder.
- Use the algorithm with the standard SA approach, i.e. random neighbour selection (or at least randomise the order of the exhaustive approach), connected solution space by relaxing hard constraints, variations in the cooling schedule (esp. geometric cooling).
- Since the non-randomised neighbour selection still produced good results we could take this approach a step further by targeting those events that do incur penalty points, e.g. by over-sampling them.
- Use more sophisticated SA approaches, e.g. dynamic schedules. Different baseline for move acceptance, e.g. not difference to current value but to

best-so-far value. This should prevent larger movement away from already achieved values.

- What are the results if we manipulate the objective function.

4 Implemented Improvements and Final Results

Only a few of the suggested changes were implemented. The only conceptual change that was tried is the use of 10 dummy events evenly distributed over the last time slots. These dummy events take part in the exchange phase in the same way as the other events. On the computational side a Δ-evaluation method for the timetables after one neighbourhood move was implemented. This brought time savings of about 60% and therefore enabled us to use the dummy approach as decribed above without violating the time constraint.

The best results for this algorithm over 50 different random seeds are given in Table 8. The sum of the penalty incurred over all 20 instances is 1011.

Instance	1	2	3	4	5	6	7	8	9	10
Results	45	25	65	115	102	13	44	29	17	61

Instance	11	12	13	14	15	16	17	18	19	20
Results	44	107	78	52	24	22	86	31	44	7

Table 8: Final results for all 20 instance