Ann Oper Res (2012) 194:111–135
DOI 10.1007/s10479-010-0737-7

Local search and constraint programming for the post enrolment-based course timetabling problem

Hadrien Cambazard · Emmanuel Hebrard · Barry O'Sullivan · Alexandre Papadopoulos

Published online: 6 June 2010

Abstract We present a variety of approaches for solving the post enrolment-based course timetabling problem, which was proposed as Track 2 of the 2007 International Timetabling Competition. We approach the problem using local search and constraint programming techniques. We show how to take advantage of a list-colouring relaxation of the problem. Our local search approach won Track 2 of the 2007 competition. Our best constraint programming approach uses an original problem decomposition. Incorporating this into a large neighbourhood search scheme seems promising, and provides motivation for studying complete approaches in further detail.

1 Introduction

Timetabling problems have a wide range of applications in education, sport, manpower planning, and logistics. A diverse variety of university timetabling problems exist, but three main categories have been identified (de Werra 1985; Carter and Laporte 1997; Schaerf 1999): school, examination and course timetabling. The Post Enrolment University Course Timetabling Problem (Lewis et al. 2007) occurs in an educational context whereby a set of events (lectures) have to be scheduled in timeslots and assigned to appropriate

This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886).

H. Cambazard · E. Hebrard · B. O'Sullivan (✉) · A. Papadopoulos
Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Cork, Ireland
e-mail: b.osullivan@4c.ucc.ie

H. Cambazard
e-mail: h.cambazard@4c.ucc.ie

E. Hebrard
e-mail: e.hebrard@4c.ucc.ie

A. Papadopoulos
e-mail: a.papadopoulos@4c.ucc.ie

rooms. The problem tackled in this paper was proposed as Track 2 of the 2007 International Timetabling Competition organised by PATAT.[1] A similar problem was also used in the 2003 competition. In the 2007 variant, two new hard constraints were introduced, which are discussed in Sect. 2. These new constraints were introduced to make the search for feasible timetables difficult, moving the competition closer to real-world timetabling where this can be a very challenging task. Finding feasible timetables in the 2003 competition was relatively easy and, consequently, the best algorithms were incomplete and focused on optimising the soft constraints of the problem. In contrast, the 2007 competition provided a strong motivation to study complete approaches, especially constraint programming (CP) techniques, and compare them with the state-of-the-art local search. While our complete CP models are not competitive in a competition setting on their own, they can be used successfully in conjunction with local search techniques. Specifically, a promising large neighbourhood search (LNS) scheme (Shaw 1998) is proposed, which contrasts with all previous published local search work on this problem (Chiarandini et al. 2006; Kostuch 2004; Rossi-Doria et al. 2002; Abdullah et al. 2005; Di Gaspero and Schaerf 2006).

Our *main contribution* in this paper is a comprehensive study of the problem using a wide range of techniques highlighting both pitfalls and positive results. A list-colouring relaxation of the problem is highlighted as the core computational challenge, and we show how to take advantage of it through the various approaches we develop. Our main *technical novelty* lies in the analysis of complete approaches with original CP models and lower bounds for the costs associated with the soft constraints of the problem, including algorithms to maintain them. We also present an original local search (LS) approach that can deal with the challenge of finding feasible timetables; this approach was the winning entry from amongst thirteen in Track 2 of the 2007 International Timetabling Competition.

The remainder of the paper is organised as follows. Section 2 defines the problem tackled in this paper. Section 3 presents a local search approach based on the ideas used in the 2003 challenge, but that addresses, efficiently, the more difficult task of finding feasible timetables. This local search constitutes a baseline against which we evaluate our complete approaches. Section 4 investigates several CP models for finding feasible timetables. The most promising one is extended in Sect. 5 in order to find optimised timetables, based on a set of additional soft constraints defined in the problem specification. A hybrid of the incomplete and complete approaches is proposed as a large neighbourhood search scheme in Sect. 6. We present a detailed empirical evaluation of each of our approaches in Sect. 7. A number of concluding remarks are made in Sect. 8.

2 Problem description

In the post enrolment-based course timetabling problem we are given a set E of n events, to be scheduled in 45 timeslots $\{1, \ldots, 45\}$ (5 days of 9 hours each) using a set R of m rooms. Each room is characterised by its seating capacity, which we will refer to as its size, and its features, defining the set of services available in it. We are also given a set S of students, along with the set of events that each student wishes to attend. To solve the problem, each event must be assigned a room and a timeslot while obeying a set of constraints. The constraints of the problem are partitioned into two sets: the *hard* constraints define the requirements of a feasible timetable, while the *soft* constraints define an optimal timetable.

[1] http://www.cs.qub.ac.uk/itc2007/.

2.1 The hard constraints of the problem

The *hard constraints* of the problem are the following:

1. No student can attend more than one event at the same time.
2. The room of each event must be large enough to accommodate the number of students attending it, and must provide all of the features required by the event.
3. Only one event can be assigned to a room in each timeslot.
4. An event can only be timetabled to take place in one of its pre-defined "available" timeslots.
5. When specified, events must occur in the correct order in the week.

In the 2007 competition, each of the benchmark instances provided to the competitors was guaranteed to admit at least one timetable that satisfied all these hard constraints. However, finding such feasible timetables was a challenging task. Therefore, the organisers of the competition also introduced the notion of "distance to feasibility" to be able to compare entries that do not find any feasible timetables. We ignore this point in our study, and regard all infeasible timetables as mere failures.

2.2 The soft constraints of the problem

While feasibility is important, is does not represent the full story. The quality of a feasible timetable is evaluated in terms of the following soft constraints. Students should not have to:

1. attend an event in the last timeslot of the day $\{9, 18, 27, 36, 45\}$;
2. attend more than two events in a row on a given day;
3. attend exactly one event during a day.

The penalty given to a feasible schedule is computed in the following way (taken from the description of the problem from the competition web-site):

- Count the number of occurrences of a student having a class in the last timeslot of the day.
- Count the number of occurrences of a student having more than two classes consecutively (3 consecutively scores 1, 4 consecutively scores 2, 5 consecutively scores 3, etc.). Classes at the end of the day followed by classes at the beginning of the next day do not count as consecutive.
- Count the number of occurrences of a student having just one class on a day (count 2 if a student has two days with only one class in each, etc.).

The total penalty of the timetable is the total of these three values.

2.3 A list colouring perspective and the colouring relaxation of the problem

The problem defined only by the hard constraints can be seen as a list-colouring with additional constraints. The colouring is defined in a graph where each node corresponds to an event and an edge is added between two events that must be timetabled into different timeslots; the set of colours available to each node are the timeslots available to the corresponding event in the timetable. This graph is primarily made of many large overlapping cliques, referred to as *student cliques*, defined by the set of events chosen by each student. In addition, two events that share a unique possible room, because of their size and features,

must be timetabled into different timeslots. Sets of events sharing such a unique room define *room cliques*. Finally, precedences between events also imply timeslot differences and can be added to the list colouring graph.

Table 1 summarises the size and number of cliques found in the colouring graphs of the first eight instances[2] made available to competitors in Track 2. Both our LS and CP approaches will try to take advantage of these cliques. Also shown is the density of the *basic graph*, i.e. the original graph of student choices including the precedence edges, and the *full* graph, i.e. the same graph augmented with room information.

The *final cliques* of Table 1 are obtained using a heuristic process. Let the neighbourhood of a clique c be the set of the nodes connected to *every* node in c. The nodes in the neighbourhood of c are not necessarily connected to each other. However, if they also belong to another clique, they are connected to each other and thus can be added to extend c to a bigger clique. The final cliques are then obtained by applying such a process iteratively, starting from the student/room cliques until a fixed point is reached. Note that while the density of the full graph is not much bigger than that of the basic graph, the added edges can significantly improve the maximum and average size of the cliques.

Finally, we define a very useful relaxation of the problem, which we will refer to as the *colouring relaxation*, which we obtain by relaxing Hard Constraint 2. This relaxation is a list colouring problem defined on the previous graph where each colour can occur at most m times, where m is the number of rooms, and precedences exist between the colours of some pair of nodes. Hard Constraints 1, 4 and 5 are the core list-colouring problem with precedences; Hard Constraint 3 is expressed here as a cardinality constraint over the occurrences of each colour in the colouring. Hard Constraint 2 is not completely ignored, but is relaxed since the room edges are included in the graph; these edges ensure that events that have the same unique possible room are timetabled into different timeslots.

A solution to the colouring relaxation is not necessarily feasible. In fact, knowing if a set of events can be assigned to a given timeslot with respect to room features and room capacity is a bipartite matching problem: the events assigned to a given timeslot can only fit in a given set of rooms depending on their features and size. Assigning each event to a suitable room is, therefore, an "event to room" matching problem. Thus, the problem defined by the hard constraints can be seen as a *list-colouring* combined with many *matching problems* (one per timeslot).

3 A local search approach

Our local search approach builds upon the various approaches used in the 2003 competition (Chiarandini et al. 2006; Kostuch 2004; Rossi-Doria et al. 2002). However, since finding feasible timetables was much more challenging in the 2007 competition, some advances over the 2003 approaches were required.

In this section we study two alternative approaches. The first one, similar to most of those used in 2003, uses a two-phase approach: we first try to identify a feasible timetable, and then try to reduce its cost of violating the soft constraints. The second approach, referred to as the *colouring strategy*, also involves two phases, but the first deals with the colouring relaxation.

[2]See http://www.cs.qub.ac.uk/itc2007/postenrolcourse/course_post_index.htm.

Table 1 Some statistics about the list colouring graph structure in the first eight competition instances from Track 2 of the International Timetabling Competition 2007

Inst.	Inst. statistics			Student cliques			Room cliques				Final cliques				Density	
	n	Students	m	Min.	Max.	Avg.	Number	Min.	Max.	Avg.	Number	Min.	Max.	Avg.	Basic	Full
1	400	500	10	18	25	21.02	6	12	32	17.17	506	12	32	22.07	0.33	0.34
2	400	500	10	19	24	21.03	5	8	32	18.80	505	8	32	21.92	0.37	0.37
3	200	1000	20	10	15	13.38	13	1	7	3.85	906	7	28	19.55	0.47	0.48
4	200	1000	20	10	15	13.40	10	1	10	3.90	925	4	33	21.75	0.52	0.52
5	400	300	20	19	23	20.92	14	2	21	9.07	314	5	25	20.66	0.30	0.31
6	400	300	20	18	24	20.73	17	4	17	11.12	317	7	26	20.62	0.29	0.30
7	200	500	20	10	15	13.47	19	3	18	8.26	498	5	29	18.57	0.52	0.53
8	200	500	20	11	15	13.83	19	2	13	7.26	503	7	25	17.65	0.51	0.52

3.1 Finding a feasible timetable

The search for a feasible timetable is performed by considering a unit cost for each hard constraint violation: an infeasible timeslot or room for an event, two events sharing a student in the same timeslot, two events violating a precedence between them.

3.1.1 Basic scheme

We present a local search with a composite neighbourhood called *the basic scheme* and describe two ideas to improve it by adding a more complex move for intensification and relaxing the matching constraint to increase the density of solutions in the search space.

Solution representation The *position* of an event is defined by a given timeslot and room. The solution is represented by the position of each event as opposed to the solution representation described in Rossi-Doria et al. (2002), which ignores the rooms and maintains the room violations by solving a matching problem per timeslot. For efficiency reasons, the lists of events per timeslot, as well as the list of all free positions in the timetable (positions where no event is currently assigned), are added to the representation.

Neighbourhood We define a composite neighbourhood (Abdullah et al. 2005; Di Gaspero and Schaerf 2006) based on the following alternative moves:

1. TrE: translates an event to a free position in the timetable.
2. SwE: swaps two events by interchanging their position in the timetable.
3. SwT: swaps two timeslots t_i and t_j, i.e. translates all events currently placed in t_i to t_j and all events in t_j to t_i.
4. Ma (Matching): reassigns the events within a given timeslot to minimise the number of events assigned to an unsuitable room; to allow violations, a maximum matching is solved. Events left unassigned in the matching are put into arbitrary rooms.
5. $TrE + Ma$: translates an event to a given timeslot and evaluates if this does not violate the room constraints by checking the corresponding matching problem; if the matching is infeasible, the move is rejected.

The order of exploration amongst these alternative moves is chosen randomly. The move is thus selected randomly but the exploration of the neighbourhood associated to the move is performed from the last point where it was left (following exactly the method in Kostuch 2004) for efficiency reason. This also ensures that all events and positions in the timetable are regularly examined. There is therefore no specific strategy for picking events or timeslots.

Search We start from a randomly generated timetable, since we found that starting from one generated by a greedy heuristic showed no benefit. Improving and sideways moves, i.e. moves that keep the current violation cost constant, are always accepted and search is not guided by events involved in hard constraints violations. Instead, we believe that moves TrE and SwE are very important since they can be performed very quickly and, therefore, provide a diversification mechanism. This also explains why we choose a solution representation that includes the room information explicitly, since this is mandatory for TrE and SwE.

By maintaining a simple tabu list, we can avoid cycling during search by forbidding an event from being moved to a timeslot it was assigned in the last k iterations; we set $k = 10$ in practise. This approach is similar to Chiarandini et al. (2006) and is classic in graph colouring (Galinier and Hertz 2006).

3.1.2 Improving the basic scheme through intensification

Assuming that a set of events have to go into different timeslots (for example, a clique of the colouring graph) and that they share no precedences, then their optimal placement in the current timetable is an assignment problem. The improvement we examine introduced intensification into the search process by enriching the neighbourhood with a more complex move, *Hu*, based on the Hungarian method for solving assignment problems. *Hu* picks a set of events $\{e_1, \ldots, e_k\}$ assigned to different timeslots ($k \leq 45$) that do not have precedences defined between them, and reassigns them optimally by solving an assignment problem with the Hungarian method (Kuhn 1955). Any violation of the hard constraints for placing each event in each timeslot is known, since it does not depend on the other removed events, they do not share precedences, and only a single event is removed per timeslot. We solve $45 \times k$ maximum matching problems to evaluate the cost, due to the room capacities, of placing each event in each timeslot (to know whether the new event can fit in the timeslot without causing one more room conflict).

This move, being quite costly, is not included by default in the neighbourhood but rather used as a greedy intensification procedure as follows. Firstly, move *Ma* is applied on every timeslot. The intensification step applies move *Hu* on each clique. All events of the clique must be in different timeslots and define an assignment problem in the current timetable. All cliques containing an event involved in a constraint violation are considered, and simplified by removing events sharing precedences inside the clique. This intensification is applied every 1000 non-improving iterations and loop over all the "final cliques" of Table 1.

3.2 Optimising the timetable

Once a feasible solution has been found, another local search optimises its soft cost.

Representation of the solution We extend the previous representation by adding the student view in order to reason about the soft constraints in the problem specification. The timetable of each student (needed for Soft Constraint 2) is maintained as a three dimensional matrix of size $|S| \times 5 \times 9$ where each entry is equal to the event attended by the student at the corresponding day and timeslot, if there is one, and set to $\perp$ otherwise. Moreover, the number of events attended by each student, each day, is stored so we can reason about Soft Constraint 3.

Neighbourhood The only move used in this phase is $TrE + Ma$. Furthermore, the only moves considered are those that preserve the feasibility of the timetable. However, this is a challenge for the search due to the tightness of the hard constraints. The main motivation for the colouring strategy (Sect. 3.3) as well as the large neighbourhood search (Sect. 6) is to compensate for this disadvantage.

Search We found that a tabu search appeared inefficient for optimising the costs of the soft constraints, and better results can be obtained using simulated annealing (SA) (Kirkpatrick et al. 1983). This seems to match the experience of Chiarandini et al. (2006), Kostuch (2004) and the study made in Rossi-Doria et al. (2002).

Improving and sideways moves are always performed, and degrading ones are accepted with a probability depending on their cost variation Δ:

$$P_{acceptance}(\Delta, \tau) = e^{-\frac{\Delta}{\tau}},$$

where the parameter τ, the temperature, controls the acceptance probability and is decreased over time. The temperature is *cooled* at each step using a standard geometric cooling:

$$\tau_{n+1} = 0.95 \times \tau_n.$$

Two parameters are needed to define the cooling: the initial temperature τ_0, and the length of a temperature step, L, i.e. the number of iterations performed at each temperature level.

As the time demand varies a lot from one problem instance to the other, we try to predict "the speed" of our soft solver during an initialisation phase by running the SA at a temperature of 1 for 20,000 iterations and set τ_0 and L in the following way. Firstly, τ_0 is set to the average of the cost variation observed during the initialisation step; then, based on the time needed to perform the initialisation, we get an estimation of the number of iterations that will be performed in the remaining time, I. By setting a final temperature to $\tau_f = 0.2$, we also know the number of temperature steps, *nbSteps*, needed to go from τ_0 to τ_f and therefore L is set to $L = \frac{I}{nbSteps}$. A reheating is performed if the neighbourhood is scanned without accepting any moves. This can happen if the number of feasible moves is limited; the SA is more likely to reject all choices as the temperature decreases.

3.3 An alternative approach based on the colouring strategy

The local search approaches to solve the hard and soft constraints can be used to solve the colouring relaxation as defined in Sect. 2.3 by ignoring the room allocation. The representation of the solution is restricted to the timeslots with a capacity of m (the number of rooms). The representation itself guarantees that at most m events can be assigned to the same timeslot.

In the case of hard constraints, the neighbourhood is restricted to TrE, SwE, SwT and Hu. In this context, Hu is simply ignoring the rooms and does not need to solve any matching to compute the violation cost of each event-timeslot pair. In the case of the soft constraints, the neighbourhood is restricted to TrE (instead of $TrE + Ma$). Moreover, only moves that preserve feasibility of the colouring relaxation are accepted.

3.3.1 The colouring strategy for finding feasible timetables

The colouring approach for satisfying the hard constraints is very simple. Once a solution to the colouring relaxation is found, it is used to initialise the local search that also operates on the rooms, rather than using a random initial timetable. By relaxing the room allocation, we increase the solution density of the search space on which the local search operates, thus allowing us to find colourings that would otherwise be difficult to reach without strongly violating the matchings.

3.3.2 The colouring strategy for generating optimised timetables

The colouring strategy for the whole problem is divided into four stages:

1. Hard constraints on colouring: Find a feasible solution S_1 to the colouring relaxation (applying the colouring strategy on the hard constraints).
2. Soft constraints on colouring: Improve the soft constraint violation of S_1 with the colouring local search for soft constraints and obtain S_2.
3. Hard constraints: Initialise the local search solver for hard constraints with S_2 and compute from that point a feasible solution S_3.

4. Soft constraints: Improve the soft constraint violation of S_3 with the local search for soft constraints and return the final result.

We allocate 90% of the time available to solve Stages 1 and 2. The idea is to invest most of the time in Stage 2 to find a good colouring solution. Stages 3 and 4 only intend to repair this good colouring solution into a feasible and hopefully good solution. By choosing 90% we assume that a good colouring solution can be easily repaired to a feasible one. However, finding a good tradeoff between the two first stages and the two last ones can probably affect the performance significantly, and this parameter could be certainly tuned more accurately.

3.4 Experimental results for the local search

We conclude the presentation of the local search approach by showing the behaviour of the search in each of the two stages, i.e. feasibility and optimisation, on a typical timetabling instance presented in Fig. 1. We will then evaluate the local search approach using the available timetabling instances from Track 2 of the International Timetabling Competition.

Both plots show the evolution of the costs at each iteration. The cooling is also indicated for the SA. The search for feasibility proceeds by moving over large plateaux of configurations of equivalent violation cost, i.e. the cost is never degraded in practise. Sideways moves appear to be very frequent for feasibility (Fig. 1(a)). Therefore, the search can remain on the same plateau for quite some time as it does not focus on events involved in constraint violations and accepts sideways steps; this is why favouring moves TrE and SwE brings diversification over the plateaux.

Sideways moves are less likely to occur at the optimisation stage (Fig. 1(b)) and one can see the effect of the cooling by observing that the cost variation is decreasing while the best known cost is converging toward its final value (no reheating occurs in this example). The choice of the different metaheuristics for feasibility and optimisation, with their resulting behaviours, is also motivated by the fact that, in the first case, we try to get a feasible solution as soon as possible, whereas in the second case we aim for the best possible solution within a given time-limit.

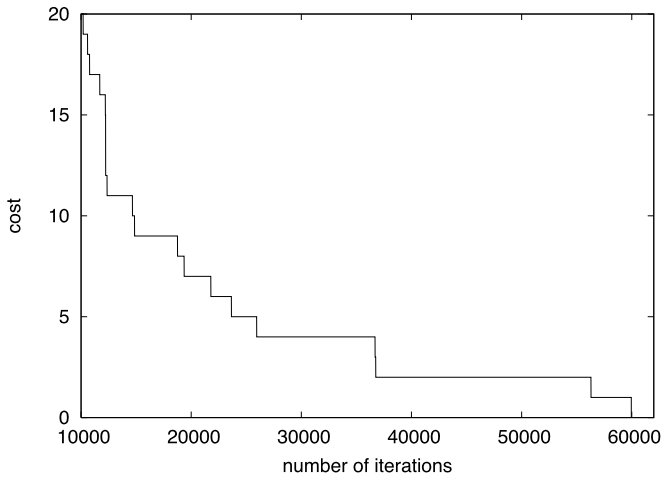
3.4.1 Detailed results for finding feasible timetables

Table 2 compares (over 100 runs with different seeds) the basic scheme described in Sect. 3.1 enhanced with the intensification procedure (LS) and the colouring strategy (LS-Colouring) on the publicly available 2007 ITC instances for Track 2. We recall that the colouring strategy splits the search for optimised timetables into two steps: a colouring solution is sought first, which is then repaired to a feasible solution (Sect. 3.3). LS was the algorithm submitted to the 2007 ITC with small variations.[3]

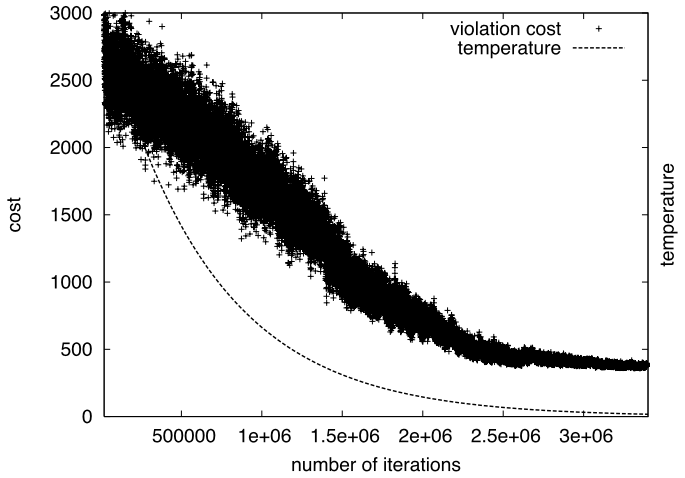
Table 2 shows the percentage of feasible solutions found over 100 runs with different seeds within the time limit set using a software utility provided by the ITC organisers for benchmarking across different hardware platforms.[4] The average time reported in the table is computed only on runs that have found a solution. The effects of the two improvements can

[3]The randomisation in the algorithm submitted was different, but systematic tests showed later that randomising the order of the moves alone achieves better performance.

[4]These experiments were run in a single thread on a Dual Quad Core Xeon CPU @ 2.66 GHz with 12 MB of L2 cache per processor and 16 GB of RAM overall, running Linux 2.6.25 × 64. The time limit given by the benchmarking system of the competition was 324 s.



(a) Tabu search - feasibility stage



(b) Simulated Annealing - optimisation stage

Fig. 1 Evolution of the violation cost per iteration for the two stages of the local search approach

mainly be seen on the hardest instances (Instances 2, 9, and 10). The colouring strategy is significantly better, justifying the usefulness of exploiting the search space of the relaxation. Satisfiable colourings can be found much faster in the search space of the relaxation and quickly lead to feasible solutions for the matchings. LS-Colouring is now, to our knowledge, the best approach for finding feasible solutions.

3.4.2 Detailed results for finding optimised timetables

Table 3 compares the local search described in Sect. 3.2 (SA), which was submitted to the competition, against SA-Colouring, the colouring strategy for the soft cost introduced in

Table 2 Percentage of feasible solutions found, with average time, using the basic scheme, a local search similar to the one submitted in the competition (LS) and the colouring strategy for feasibility (LS-Colouring). 100 runs per instance were performed with different seeds

	Instances	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Basic scheme	% solved	100	99	100	100	100	100	100	100	99	**48**	100	100	100	100	100	100
	Avg. time (s)	15.22	48.67	0.49	0.57	4.69	5.67	2.42	1.45	67.85	223.31	0.52	0.84	8.86	7.97	0.8	0.55
LS	% solved	100	98	100	100	100	100	100	100	99	**60**	100	100	100	100	100	100
	Avg. time (s)	14.37	50.72	0.57	0.67	4.43	5.19	2.33	1.4	60.11	206.81	0.65	1.04	7.36	6.38	0.84	0.61
LS-Colouring	% solved	100	100	100	100	100	100	100	100	100	**98**	100	100	100	100	100	100
	Avg. time (s)	7.31	15.8	0.47	0.48	2.77	3.47	0.59	0.49	14.78	53.87	0.63	0.73	3.86	3.75	0.6	0.5

Table 3 Overall results on 100 runs using a time-limit of 324 s (see Footnote 4), reporting the average, median, min and max cost for the SA and SA-Colouring techniques

Inst.	SA					SA-Colouring				
	% feas	Avg.	Med.	Min.	Max.	% feas	Avg.	Med.	Min.	Max.
1	100	830	845	358	1313	100	**547**	**555**	**15**	**1072**
2	100	924	969	11	1965	100	**403**	**356**	**9**	**1254**
3	100	**224**	**220**	**156**	**396**	100	254	237	174	465
4	100	**352**	**351**	**61**	**471**	100	361	362	249	666
5	100	**3**	**3**	0	**14**	100	26	11	0	154
6	100	**14**	0	0	285	100	16	0	0	**133**
7	100	11	8	5	83	100	**8**	8	**1**	**32**
8	100	0	0	0	0	100	0	0	0	0
9	100	1649	1643	1049	2377	100	**1167**	**1251**	**29**	**1902**
10	**98**	2003	1999	773	2940	89	**1297**	**1394**	**2**	**2637**
11	100	**311**	**309**	**157**	**456**	100	361	366	178	496
12	100	408	420	**0**	782	100	**380**	**397**	14	**676**
13	100	**89**	**74**	0	**270**	100	135	111	0	425
14	100	**1**	1	0	**14**	100	15	1	0	139
15	100	80	0	0	311	100	**47**	0	0	**294**
16	100	**19**	**11**	1	**120**	100	58	23	1	245

Sect. 3.3. We report the average, median and min/max cost obtained over 100 runs with different seeds.

SA-Colouring outperforms SA on instances 1, 2, 9, and 10. Those instances are the tightest from a feasibility point of view, as shown previously. The difficulty of finding feasible timetables strongly handicaps the SA approach, which maintains feasibility while improving the cost. The colouring strategy gains flexibility by ignoring the matchings. However, the latter strategy might rely on the fact that a perfect solution (a feasible timetable of soft cost 0) is guaranteed to exist on this benchmark so that good configurations could also be likely to be feasible solutions. If optimal soft cost configurations were not feasible, optimising the soft cost before having a feasible solution, as per the colouring strategy, could lead us outside the feasible region. However, this conjecture needs to be confirmed by generating instances without this property.

3.4.3 Comparisons with other solvers in the competition

Five algorithms[5] were chosen for the final phase of the Track 2 of the 2007 ITC. These were evaluated on 24 timetabling instances: the 16 already mentioned and 8 unknown competition ones. Since all solvers were randomised, 10 runs per instance were performed giving 50 runs per instance. Each run for each solver was ranked among the 50 for each instance and the average rank across all runs and all instances was used to give a rank to each solver. Table 4 shows the ranking of each solver, with the number of times they have found the best solution among all runs for a given instance, and the number of times they have failed to find a feasible solution.

[5]The other four algorithms were developed by: M. Atsuta, K. Nonobe, T. Ibaraki; M. Chiarandini, C. Fawcett, H. Hoos; C. Nothegger, A. Mayer, A. Chwatal, G. Raidi; and T. Muller.

Table 4 Ranking of the five finalists from the tests ran by the organisers on the 24 instances

	Cambazard et al.	Atsuta et al.	Chiarandini et al.	Nothegger et al.	Muller
Average rank (out of 240 runs)	**13.9**	24.43	28.34	29.52	31.31
Number of best solutions (out of 24 instances)	**13**	11	3	11	0
Number of failures on feasibility (out of 240 runs)	17	43	**4**	54	53
Rank in the competition	**1**	2	3	4	5

Our local search approach, with a score of 13.9, did significantly better than all other entries. Our approach appeared generally more robust both for finding feasible and good solutions. Chirandini et al. were best on feasibility only. We believe that LS-Colouring is now the best approach to finding feasible timetables. It also does not take the soft cost into account therefore being independent of the perfect solution property of the benchmark. Our approach also obtained the best results on many instances. The detailed results of the competition (not presented here) show that our approach was however outperformed (e.g. by Nothegger et al.) on instances we considered to be very hard on feasibility, e.g. instance 10. Therefore, we believe that there is a significant room for improvement in our results.

4 Constraint programming models for feasible timetables

The post enrolment-based course timetabling problem has been tackled by a number of local search techniques (Chiarandini et al. 2006; Kostuch 2004; Rossi-Doria et al. 2002; Abdullah et al. 2005; Di Gaspero and Schaerf 2006). We are not aware, however, of any complete approach. To this end we considered several CP models, none of which were able to match the efficiency of local search. However, as we shall see in Sect. 6, the CP approach can still be valuable to provide complex neighbourhoods for incomplete algorithms. It also constitutes a sound basis for future work on complete approaches for this problem. We present here the most promising CP model as well as two less successful ones and give some insights into their inefficiency.

4.1 Basic model

For an event i we introduce two variables $eventTime_i \in \{1, \ldots, 45\}$ and $eventRoom_i \in \{1, \ldots, m\}$, for the timeslot and room associated to event i, respectively. Let R_i be the set of rooms that can accommodate event i; T_i be the set of timeslots available for event i; $student(i)$ be the set of students attending event i; and let $prec$ be the set of the pairs of ordered events. We define the first model as follows:

Model 1

$$\forall i, j \leq n \text{ such that } student(i) \cap student(j) \neq \emptyset \qquad eventTime_i \neq eventTime_j \quad (1)$$
$$\forall i \leq n \qquad eventRoom_i \in R_i \quad (2)$$
$$\forall i, j \leq n \qquad (eventTime_i \neq eventTime_j) \vee (eventRoom_i \neq eventRoom_j) \quad (3)$$
$$\forall i \leq n \qquad eventTime_i \in T_i \quad (4)$$
$$\forall (i, j) \in prec \qquad eventTime_i < eventTime_j \quad (5)$$

In this viewpoint, the constraints (1), (4) and (5) correspond to a *list colouring* problem with precedences on the variables *eventTime*. Constraints (2) and (3) enforce that events must be allocated to suitable rooms, and that within a given timeslot, every event must be put into a different room, respectively. They correspond to a set of *matching* problems conditioned by the result of the above colouring problem. In this model, the colouring and matching aspects of the problem are relatively disconnected. In fact, as long as an event is not committed to a given timeslot, we do not know in which matching it will participate, because of the disjunctions (3). If an early decision on the colouring part prevents a consistent room allocation, it will not be discovered until very late in the search, leading to a *trashing* behaviour where large unsatisfiable subtrees are explored again and again. We explored two ways of resolving this issue. First, we modelled the relation between the room allocation (matching) and timeslot allocation (colouring) using a global constraint (Aggoun and Beldiceanu 1993) to achieve stronger inference between these two aspects and detect mistakes earlier. We describe this model in Sect. 4.2. The second solution was to separate the solving of the colouring and the matchings, so that we explore more diverse colourings, and hopefully avoid trashing. We describe this model in Sect. 4.3.

4.2 Matching constraint

As mentioned in the problem description (Sect. 2), knowing if a set of events can fit in a given timeslot with respect to room availability and capacity is a bipartite matching problem (events to rooms). The objective is to remove the *eventRoom* variables from the search space. In other words, we will make sure that constraint propagation alone ensures that an assignment of all events can be extended to a matching for each timeslot. As a result, we solve a colouring problem where we only assign events to timeslots subject to timeslot availability and precedences, and such that the remaining matching sub-problems are backtrack-free.

The room allocation sub-problem can be represented in a bipartite graph $G = (V_1, V_2, E)$ where $V_1 = \{1, \ldots, n\}$ is the set of events, and $V_2 = \{\langle 1, 1\rangle, \ldots, \langle 45, m\rangle\}$ is the set of all pairs $\langle$timeslot, room$\rangle$. An edge $(i, \langle j, k\rangle)$ is present iff event i can be assigned to timeslot j in room k. A maximal matching of G thus represents an assignment of events to rooms satisfying constraints (2) and (3). We introduce n variables to link this matching with the colouring: $event_i \in \{\langle 1, 1\rangle, \ldots, \langle 45, m\rangle\}$ denotes the timeslot and room, represented by a pair, to which event i is assigned. An ALLDIFF($\{event_1, \ldots, event_n\}$) constraint (Régin 1994) ensures that the graph G admits a matching of cardinality n. Notice that during search, arc-consistency is achieved for all matching problems at once, giving stronger inference than considering the matchings independently. Notice that we could post other constraints directly on variables *events*. However, this can be done more easily on the *eventTime* variables. They also provide a naturally good branching scheme, since rooms have been factored out of the search space. We thus define a second model, where we substitute the variable $eventRoom_i$ with $event_i$ and channel it to $eventTime_i$ using a simple binary constraint.

Model 2

$$\forall i, j \leq n \text{ such that } student(i) \cap student(j) \neq \emptyset \quad eventTime_i \neq eventTime_j \quad (1)$$
$$\forall i \leq n \quad eventTime_i \in T_i \quad (4)$$
$$\forall (i, j) \in prec \quad eventTime_i < eventTime_j \quad (5)$$
$$\forall i \leq n \quad event_i \in \langle R_i \times T_i \rangle \quad (6)$$
$$\forall i \leq n \quad eventTime_i = event_i[0] \quad (7)$$
$$\textsc{AllDiff}(\{event_1, \ldots, event_n\}) \quad (8)$$

Constraint (7) channels the variables *eventTime* to *event* by projecting on the first element of the pair. Notice that since arc consistency is achieved in polynomial time on the ALLDIFF constraint, an assignment of *eventTime* satisfying Model 2 can always be extended to *event* in a backtrack-free manner.

4.3 Alternate colourings and matchings

Constraint (8) of Model 2 is very costly to maintain. Therefore, we consider a decomposition similar to a logic-based Benders decomposition scheme (Hooker and Ottosson 2003). We adopt the colouring strategy of the local search (Sect. 3.3), and first solve the colouring relaxation and delay the resolution of the matchings until a colouring has been found. If the matching is infeasible, we seek another solution for the colouring, and iterate in this way until a full solution is found. Clearly, solving the colouring part alone allows for a far more optimised and sleeker model, however, reaching a fixed point might not be easy. We first describe the lighter model restricted to the colouring and precedence constraints, and where the room allocation constraints are relaxed to a simpler cardinality constraint. Then, we show how Benders cuts can be inferred when failing to solve a matching in order to tighten the colouring sub-problem.

The *eventRoom* variables are ignored as in the previous model and a single global cardinality constraint (GCC) (Régin 1996) is added to ensure that every timeslot is used at most m times. This constraint eliminates trivially infeasible matchings where the number of events assigned to a timeslot is greater than the number of rooms.

Model 3

$$\forall i, j \leq n \text{ s.t. } student(i) \cap student(j) \neq \emptyset \quad eventTime_i \neq eventTime_j \quad (1)$$
$$\forall i \leq n \quad eventTime_i \in T_i \quad (4)$$
$$\forall (i, j) \in prec \quad eventTime_i < eventTime_j \quad (5)$$
$$\forall i \leq n \quad \textsc{Gcc}(\{eventTime_i \mid i \leq n\}, [[0..m], \ldots, [0..m]]) \quad (9)$$

A solution of this model is not guaranteed to be a feasible solution of the original problem. Indeed, a matching problem can be inconsistent once the colouring is fixed. We, thus, iteratively solve the colouring part until we find a feasible room allocation, as depicted in Algorithm 1. If a matching problem fails, a minimal conflict corresponds to a set of events that cannot be assigned together in any timeslot. We use an algorithm for finding minimal conflicts (de Siqueira and Puget 1988) to extract such a set of events (line 3). In order to rule out this conflicting assignment in future resolutions of the colouring sub-problem, we post a NOTALLEQUAL constraint to the model (line 4). The constraint $\textsc{NotAllEqual}(x_1, \ldots x_k)$ ensures that there exists $i, j \in [1..k]$ such that $x_i \neq x_k$. This acts as a Benders cut and prevents the same assignment from being met repeatedly. Observe that since we extract minimal sets of conflicting events (Jain and Grossmann 2001; Cambazard et al. 2004), entire classes of assignments that would fail for the same reason are

Algorithm 1: `Decomposition`

```
1 repeat
2     solve Model 3;
      matched ← true;
      foreach 1 ≤ j ≤ 45 do
          G ← (V1 = {i | eventTime_i = j}, V2 = ⋃_{i∈V1} R_i, E = {(i,k) | i ∈ V1, k ∈ R_i});
          if cannot find a matching of G then
              matched ← false;
3             cut ← Extract-min-conflict(G);
4             add NotAllEqual(eventTime_k | k ∈ cut) to Model 3;
  until matched;
```

ruled out. Notice also that although this constraint is inferred from a particular timeslot, it holds for every timeslot.

We explored further improvements of this model based on the analysis of the colouring graph described in Sect. 2. Conflicts between events are organised into large cliques, one for each student and even larger cliques can be inferred by taking room conflicts into account. This information can be used to obtain stronger filtering from the model. One possibility is to replace the constraints (1) by ALLDIFF constraints. Each of the aforementioned "final cliques" implies an ALLDIFF constraint between a set of $eventTime_i$ variables. In this manner, all the binary differences (1) are covered by at least one clique and can thus be removed. We can expect to achieve a stronger level of propagation as a result. On the other hand, ALLDIFF can be expensive to maintain. We can therefore choose to keep, amongst the final cliques, only the cliques obtained from a room clique, as a trade-off between the efficiency of binary differences and the additional reasoning brought by the cliques, as they are big and they contain additional conflicts. This leads to two variations of Model 3 that we assess empirically below.

4.4 Experimental results

We ran Model 2, Model 3, Model 3-cliques (Model 3 including all implied ALLDIFF constraints) and Model 3-rooms (Model 3 including only the ALLDIFF constraints representing room cliques). In Table 5, we give the number of iterations of Algorithm 1 (`Decomposition`), that is, the number of feasible colourings that were required to find a complete solution. This number is always 1 for Model 2. We also give the cumulative CPU time in seconds and number of nodes explored on solved instances.

Notice that no model could solve instances 1, 2, 9, 10, 13 and 14 within the time cutoff of 420 s, corresponding to the 10 min cutoff in the competition, on an Apple MacBook. Model 2 does not need to solve several colouring problems, however, the overhead due to the extra variables (*event*) and to the large ALLDIFF constraint, is too important. We also observe that in most cases, the ALLDIFF constraints on events sharing the same unique suitable room dramatically reduces the number of iterations required to solve the problems. On the other hand, using ALLDIFF constraints for representing the colouring problem seems to be detrimental. The best combination seems to be Model 3 using ALLDIFF only for rooms.

We believe that the main reason for Model 3 dominating Model 2 is that the difficult part of the problem lies primarily in the colouring for these instances. The very low number

Table 5 A comparison of the various CP models we studied

Inst.	Model 2			Model 3 (conflicts)			Model 3 (all)			Model 3 (room)		
	Iter.	Time (s)	Nodes	Iter.	Time (s)	Nodes	Iter.	Time (s)	Nodes	Iter.	Time (s)	Nodes
3	1	12.813	327	2	5.111	312	1	15.919	198	1	4.850	198
4	–	–	–	2	7.386	3789	1	18.154	351	1	4.814	351
5	–	–	–	–	–	–	4	23.428	5335	3	9.599	1977
6	–	–	–	22	28.878	26,049	3	93.895	42,753	2	25.619	22,137
7	–	–	–	9	17.608	21,410	1	17.459	2595	1	7.690	5626
8	1	119.190	2144	6	2.521	534	1	7.283	633	1	3.015	633
11	–	–	–	5	4.297	713	3	18.485	443	3	5.678	1896
12	–	–	–	8	160.732	178,437	2	271.381	29,291	1	75.705	78,666
15	–	–	–	10	2.528	601	2	6.096	191	2	2.783	191
16	1	4.883	213	12	2.477	1143	2	6.153	261	2	2.713	261

of colouring sub-problems solved when adding the implied ALLDIFF constraints provides further evidence of this. Any given colouring satisfying the implied ALLDIFF constraints is very likely to be extensible to a feasible matching. We also observed (but this is not apparent in the tables) that the extra GCC constraint used to approximate the matching part was almost unnecessary in most cases. That is, even without this constraint, the number of iterations to reach a complete solution remains relatively small. Notice, however, that this last observation does not stand for instances 1, 2, 9 and 10, which happen to be the hardest.

Next we compare three heuristics all using the best model: Model 3 (room). We used the good general purpose heuristics *minimum domain over future degree (dom/deg)* and *impact* (Refalo 2004) as benchmarks. The former was used successfully on list-colouring problems in the past, whilst the latter proved to be the best in our experiments. The third heuristic, *contention*, is based on computing the contention of events for a given timeslot. In scheduling, resource contention has been used as heuristic with considerable success (Sadeh and Fox 1996). In our case, timeslots can be viewed as resources, of capacity m (the number of rooms), required by events. The *contention* $C(j)$ of a timeslot j is

$$C(j) = \sum_{i \mid j \in D(eventTime_i)} 1/|D(eventTime_i)|.$$

Intuitively, this quantity describes the demand for timeslot j. It clearly induces a value ordering, since less contended for time slots are less likely to lead to a failure. Next we can compute a contention value for variables $C(eventTime_i)$, representing the "constrainedness" of a given variable and is equal to

$$C(eventTime_i) = \sum_{j \in D(eventTime_i)} \frac{1}{C(j)}.$$

The event i that minimises $C(eventTime_i)$ and the timeslot j that minimises $C(j)$ are explored first.

In Table 6, we give the number iterations of `Decomposition` (Algorithm 1) as well as the cumulative cpu time and number of nodes explored on solved instances. The results clearly show that *contention* dominates *dom/deg* and is itself dominated by *impact*. Notice that these two better heuristics also provide value orderings, whereas *dom/deg* does not. This

Table 6 Comparison of search heuristics for the CP models

Inst.	Impact			Contention			Dom./Deg.		
	Iter.	Time (s)	Nodes	Iter.	Time (s)	Nodes	Iter.	Time (s)	Nodes
3	1	4.850	198	1	3.455	182	1	3.183	228
4	1	4.814	351	–	–	–	–	–	–
5	3	9.599	1977	3	66.489	112,413	–	–	–
6	2	25.619	22,137	2	318.635	529,877	–	–	–
7	1	7.690	5626	–	–	–	–	–	–
8	1	3.015	633	2	1.958	413	3	3.021	3098
11	3	5.678	1896	4	3.165	342	–	–	–
12	1	75.705	78,666	–	–	–	–	–	–
15	2	2.783	191	1	6.224	6478	–	–	–
16	2	2.713	261	2	1.878	252	2	1.831	237

is important on these benchmarks, since they have a relatively large number of solutions whilst being hard for a complete method.

5 Constraint programming models for timetable optimisation

In this section we introduce three soft global constraints to reason about the costs, based on the soft constraints from the problem description, and specifically derive lower bounds for those. The main difficulty we encountered is that all three costs are defined in terms of students who are numerous and, thus, not represented explicitly in our CP model. In each case we tried to circumvent this issue by projecting the cost onto events and/or timeslots. All the pruning is done regarding the current upper bound (best known solution) denoted ub in the following. We say that a lower bound of a cost is *exact* if and only if it is the cost of an optimal solution given the current domains, and discarding all other constraints.

5.1 Last timeslot of each day

This soft constraint counts the number of students attending an event in the last timeslot of the day ($\{9, 18, 27, 36, 45\}$). We introduce, for each event i, a Boolean variable b_i such that $b_i = 0$ if event i is in a timeslot other than the last ones, and $b_i = 1$ if the event i is in one of the last timeslots. The cost can then be expressed as $cost_1 = \sum_i (b_i \times |student(i)|)$. The Boolean variables can be added to Model 3 and channelled with $eventTime_i$ or a simple dedicated global constraint can be implemented. We chose the latter option for efficiency reasons and to be able to augment it with stronger inference.

Lower bound Consider the bipartite graph $G = (V_1, V_2, E)$ described in Sect. 4.2 and captured by Constraint (8) of Model 2. We recall that $V_1 = \{1, \ldots, n\}$ is the set of events and $V_2 = \{\langle 1, 1\rangle, \ldots, \langle 45, m\rangle\}$ is the set of all pairs ⟨timeslot, room⟩. The existence of a maximum matching in this graph ensures a possible allocation of each event to a pair $\langle timeslot, room\rangle$. WG extends G by adding a weight w_{ij} to each edge of E defined as follows:

$$w_{ij} = \begin{cases} |student(i)| & \text{iff } j = \langle a, b\rangle \text{ with } a \in \{9, 18, 27, 36, 45\}; \\ 0 & \text{otherwise.} \end{cases}$$

Let us denote by W the value of a minimum weighted matching in WG. W is a lower bound on the minimum number of students that must go in the last timeslots. Adding these weights to the matching extends the ALLDIFF constraint to an ALLDIFF with costs.

Pruning The ALLDIFF with costs is a specific case of the global cardinality constraint with costs (Régin 1999). We refer the reader to Régin (1999) for details about the pruning.

Computational complexity The maximum weighted matching corresponds to an assignment problem and can be solved in polynomial time (in $O(n^2 45m)$ for our particular case (Régin 1999)). This improved bound has not yet been included in our current implementation. Note that this bound is exact when relaxing only Hard Constraints 1 and 5 of the problem description. We saw that the colouring sub-problem can however be tighter than the matchings so that reasoning on the colouring can improve this bound.

5.2 Consecutive events

This soft constraint counts the number of students attending more than two events in a row on a given day. We present the lower bound developed for this cost and show how it is maintained incrementally at a relatively low computational cost.

Lower bound The cost associated with the consecutive allocation of every possible triplet of events is pre-computed initially and stored in a large static table: $\textit{static-cost}(i_1, i_2, i_3) = |\textit{student}(i_1) \cap \textit{student}(i_2) \cap \textit{student}(i_3)|$. We first consider only events committed to a timeslot, i.e., instantiated variables. The lower bound, $lb(cost_2)$ takes into account the sum of these costs implied by instantiated events i_1, i_2, and i_3 to consecutive timeslots. This part of the bound is referred to as $lb_g(cost_2)$:

$$\textit{ground-cost}(i_1, i_2, i_3) = \begin{cases} \textit{static-cost}(i_1, i_2, i_3) & \text{if } \textit{eventTime}_{i_1/i_2/i_3} \text{ are} \\ & \text{assigned and consecutive;} \\ 0 & \text{otherwise.} \end{cases}$$

$$lb_g(cost_2) = \sum_{i_1 < i_2 < i_3} \textit{ground-cost}(i_1, i_2, i_3).$$

Then, for each unassigned event, a lower bound on the cost involved by its insertion in the current timetable is maintained. For a timeslot j, let $pairs(j)$ be the set of pairs of events assigned respectively to $j-2$ and $j-1$, or $j-1$ and $j+1$, or $j+1$ and $j+2$. The cost of assigning event i to timeslot j is the sum of all triplets formed by i and any existing pair p in $pairs(j)$. Then we define $\textit{pending-cost}(i, j)$ as: $\sum_{p \in pairs(j)} \textit{static-cost}(p \cup \{i\})$. The lower bound $lb(i)$ associated with allocating event i to one of its possible timeslots is equal to the minimum pending cost over all values $lb(i) = min_{j \in D(\textit{eventTime}_i)} \textit{pending-cost}(i, j)$. We use the following lower bound during search:

$$lb(cost_2) = lb_g(cost_2) + \sum_{|D(\textit{eventTime}_i)| > 1} lb(i).$$

Pruning We prune timeslot j for event i iff $lb(cost_2) + \textit{pending-cost}(i, j) - lb(i) > ub$.

Computational complexity The base lower bound $lb_g(cost_2)$ is maintained incrementally during search. It is updated only when a variable *eventTime* becomes assigned to some timeslot. In this case we increase the cost by the value of the *static-cost* of the newly formed triplets of events. There are at most $35m^3$ triplets in total, the amortised computational cost of maintaining this lower bound along one branch of the search tree is thus $O(m^3)$.

The pre-computation of the static-cost is key for efficiency. Computing *pending-cost*(i, j) can be done in $O(m^2)$ time since there are three sets of at most m^2 pairs to consider for each timeslot of each event. Since there are at most 45 possible timeslots for a given event, one can compute $lb(i)$ for all events in $O(nm^2)$. In practise, we update the values of $lb(i)$ only when event i loses some values, or when another variable gets assigned to some timeslot j and $D(eventTime_i) \cap \{j-2, j-1, j+1, j+2\} \neq \emptyset$. The pruning can be done in the same time complexity since we only need to go through at most $45n$ values.

5.3 Single events

This soft constraint counts students attending a single course in any day of the week. The non-monotonic nature of this cost makes it difficult to reason about. In fact, scheduling an event in a given day simultaneously increases the cost for students attending only this event, and decreases it for students attending another, until then unique, event. We show that even when we relax all other factors to an extreme case, computing an optimal lower bound for this cost is NP-hard.

Theorem 1 *Finding the exact lower bound for Soft Constraint* 3 *is NP-complete, even if all other constraints are relaxed.*

Proof We consider the problem of finding a lower bound for $cost_3$ for a given day, with no external constraint. We only assume that a set of events may already have been assigned to this day (event variables with a singleton domain), and that we have a finite set of extra events to choose from (yet unassigned event variables). Observe that, in this formulation, even the "domain constraints" are relaxed since it is possible for an event to account for the occupation of two slots in two different days. We analyse the corresponding decision problem, SINGLE-EVENT:

> *Data:* An integer k, a set R of events already assigned to one given day, and another set P that can possibly be assigned to this day.
> *Question:* Is there a set $R \subseteq S \subseteq P$ of events such that no more than k students have a single event in that day?

We reduce SET-COVER to SINGLE-EVENT. A SET-COVER instance is composed of a set $U = \{u_1, \ldots, u_n\}$, a set $S = \{S_1, \ldots, S_M\} \subseteq 2^U$ of subsets of U, an integer $k \leq M$. The problem consists of deciding whether there exists a set $C \subseteq S$ such that $\bigcup_{S_i \in C} S_i = U$ and $|C| \leq k$.

We build R with one event E, that contains $k+1$ students $e_i^1, \ldots, e_i^{k+1}$ per element u_i of U (the element-students). We build P with an event E_j for each subset $S_j \in S$. Each event E_j contains the element-students of each element in S_j, i.e. the element-students $e_i^1, \ldots, e_i^{k+1}$ for each $u_i \in S_j$, plus one unique student s^j (the set-student).

Each subset $R \subseteq S \subseteq P$ of cost k, i.e. such that no more than k students attend a single event in the day, corresponds to a set cover of U of size k, and vice-versa:

- For a given set cover C of size k, the set $S = \{E_j | S_j \in C\}$ is of cost k.

- Conversely, let S be a set of cost k. Clearly, S corresponds to a set cover: if any element of U is not covered, then the cost of S is at least $k+1$ (each uncovered element corresponds to $k+1$ element-students attending only E). Now, as all the element-students attend at least two events, the cost can only result from the set-students, which is simply the number of events (other than E) in S. □

Observe that solving a sequence of SINGLE-EVENT instances with decreasing values of k gives us a lower bound on this cost when all other constraints are relaxed, and *without even imposing each variable to take at least one value*. For instance if event i is in P, we can choose not to schedule it at all, whereas in effect, it will necessarily be assigned to some day of the week. This is, therefore, a much easier problem than finding the exact lower bound of Cost 3. However, even this relaxed problem is NP-complete.

Taking this fact into consideration, we only maintain this cost correctly in the computationally cheapest possible way. We consider each pair ⟨day, student⟩. As long as at least two events attended by this student can potentially happen this day, we do nothing. Otherwise, there are two choices, either this student has no course at all in this day, or has exactly one. In the latter case we increase the cost by one. This can be efficiently done with a system akin to the *watched literals* used in SAT unit propagation (Moskewicz et al. 2001). For every student and every day, we randomly pick two events to "watch" for this pair. When an event cannot be assigned to some day anymore, we update the list of students it watches for that day, finding a new available watcher. Notice that this is very cheap to do. For instance if this event was not watching any student for that day, it does not cost anything at all. When we do not find any replacement, we know that the given student is either attending no event in that day, or only a single one. We update the cost accordingly.

6 Large neighbourhood search

One weakness of the local search approach is the lack of flexibility when moving in the space of feasible solutions. The search space accessible from a given feasible solution might be very limited by the hard constraints and even disconnected. In such a case, the search can only reach the best solution connected with the initial one.

One solution would be to relax feasibility during search without any guarantees to find it again or to restart from different feasible solutions. We already investigated the first option with the colouring strategy. Another possibility is to design more complex moves that affect larger parts of the current assignment. Move *Hu* is one example of a complex move that remains polynomial. A more general kind of move can be performed using a complete solver. This is the central idea of Large Neighbourhood Search (LNS) (Shaw 1998). LNS is a local search paradigm where the neighbourhood is defined by fixing a part of an existing solution. The rest of the variables are said to be *released* and all possible extensions of the fixed part define the neighbourhood which is usually much larger than the one obtained from classical and elementary moves. Algorithm 2 presents the simple LNS scheme. An efficient systematic algorithm is needed to explore this large neighbourhood and the CP Model 3 presented earlier will be used for this.

Nature and size of the neighbourhood The selection of variables to release is a key element of a LNS scheme: we need to decide *which* events should be released (nature of the neighbourhood), and in *what number* (size of the neighbourhood). Previous work on LNS (Danna and Perron 2003; Perron et al. 2004; Perron and Shaw 2004) outlines the importance of

Algorithm 2: `LNS Scheme`

```
find a feasible solution;
while optimal solution not found or time limit not reached do
    choose a set of events to release;
    freeze the remaining events to their current position;
    if search for an improving solution succeeds then
        update the upper bound;
```

structured neighbourhoods specific to the problem. We have investigated neighbourhoods that release events per timeslots (all events contained in a given set of timeslots). It is critical to choose a neighbourhood that releases *related* variables, i.e. variables that are likely to be able to change and exchange their values. It is indeed very important that the neighbourhood contain more feasible solutions than the one we already had before releasing the variables. A promising neighbourhood should also be likely to contains feasible solutions of better cost. We, therefore, investigated a neighbourhood that releases kc conflicting and kr random timeslots (all events in the corresponding timeslots are released).

The size of the neighbourhood is difficult to set as the tradeoff between searching *more* versus searching *more often* is difficult to achieve. We choose to start from small sizes ($kc = 2$ and $kr = 2$) and to increase it when the search stagnates; in practise, after 100 non-improving iterations, the minimum of kc and kr is increased by 1. The reason is that the accurate size seems to vary a lot between timetabling instances. Much bigger sizes are typically needed for instances 1, 2, 9 and 10 where feasibility is tight. Sideways moves are again very important for diversification and are always accepted.

LNS as an intensification mechanism for the SA The LNS approach relies only on the CP solver as shown in Algorithm 2. Another idea is to use the LNS move at the low temperatures of the SA to help the very important and final phase of optimisation performed at the end of the cooling phase. In this mode, we do not accept sideways moves to speed up the solving process and look for improving solutions only. Diversification is ensured by other moves of the SA that continuously change the current assignment. The CP move is included in the neighbourhood of the SA at each iteration with a probability that increases while the temperature decreases:

$$p_{include\_lns}(\tau) = \frac{1}{200 * \tau}.$$

7 Experimental results comparing the various approaches

We summarise the results of our study with complete approaches (including the SA of the local search as a mean of comparison):

- CP: The Constraint Programming approach described in Sects. 4 and 5 based on Model 3 (room) using Impact-based search.
- LNS: The Large Neighbourhood Search approach relying on Model 3 (room) (the local search of Sect. 3.1 is used to provide an initial feasible solution).
- SA: The local search approach described in Sect. 3 which is based on Simulated Annealing for the optimisation stage.

Table 7 Overall results on 100 runs reporting the average, median, min and max cost for LNS, SA and SA_LNS. CP being deterministic, a single cost is given

Inst.	CP	LNS				SA				SA_LNS			
		Avg.	Med.	Min.	Max.	Avg.	Med.	Min.	Max.	Avg.	Med.	Min.	Max.
1	–	1977	1979	1566	2278	830	845	358	1313	827	839	349	1308
2	–	2206	2189	1789	2678	924	969	11	1965	930	968	0	1938
3	1930	860	843	515	1174	224	220	156	396	221	213	142	399
4	2097	948	942	667	1213	352	351	61	471	349	352	22	474
5	1767	912	931	660	1110	3	3	0	14	5	5	0	23
6	1681	989	996	766	1172	14	0	0	285	15	2	0	285
7	1450	518	524	258	735	11	8	5	83	1	0	0	50
8	1111	465	462	166	714	0	0	0	0	0	0	0	0
9	–	2334	2330	1989	2586	1649	1643	1049	2377	1634	1623	978	2313
10	–	2473	2469	2059	2916	2003	1999	773	2940	1985	1983	772	2845
11	2388	909	899	642	1170	311	309	157	456	302	299	149	456
12	2328	1163	1147	767	1498	408	420	0	782	402	413	0	748
13	–	1018	1017	784	1207	89	74	0	270	88	76	0	269
14	–	1053	1051	789	1258	1	1	0	14	2	2	0	6
15	1225	585	585	404	778	80	0	0	311	79	0	0	309
16	964	443	441	312	654	19	11	1	120	10	0	0	114

- SA_LNS: the SA approach augmented with LNS as an intensification mechanism at the end of the cooling (still using Model 3 (room)).

Table 7 reports the cost found by each technique on the 16 available timetabling instances from Track 2 of the 2007 ITC. LNS, SA and SA_LNS were run on 100 different seeds and the average, median, minimum and maximum cost found over the 100 runs are reported. The CP approach is entirely deterministic and a single run is therefore shown. Two computers were used, CP was run on a MacBook[6] within a time limit of 420 s and the others were run on a cluster[7] within a time limit of 324 s, selected using the ITC benchmarking utility that gives an approximation of 10 min of competition time on alternative hardware platforms.

Firstly, the LNS scheme outperforms CP alone (even by only looking at instances where CP does find a feasible solution) while being a very simple application of CP. Secondly, LNS is itself outperformed by the SA. We observed here that it remains stuck in local minima despite the large size of the neighbourhood. Finally, SA_LNS marginally improves over SA. The CP moves do not seem to bring much more flexibility to the SA to escape local minima in general. It allows, however, to find three new optimal solutions (Instances 2, 7, and 16) and improves the performance on two instances (7 and 16). Note that all the minimum costs are improved showing that LNS does play a role in the final intensification stage.

[6]Mac OS X 10.4.11, 2 GHz Intel Core 2 Duo, 2 GB 667 Mhz DDR2.

[7]A single thread on a Dual Quad Core Xeon CPU @ 2.66 GHz with 12 MB of L2 cache per processor and 16 GB of RAM overall, running Linux 2.6.25 × 64.

8 Conclusion

We have presented a comprehensive study of a university timetabling problem, comparing a variety of local search and constraint programming approaches. A central and very successful idea across the methods is the decomposition based on colouring and matching views of the problem. We designed a constraint programming approach that proceeds by decomposing the list-colouring and the matching subproblems and outperforms more classic CP models. Lower bounds were introduced to tackle soft constraints, leading to the first complete algorithm for this problem.

While our local search technique benefits from the experience of the 2003 competition, we have presented several improvements to deal with hard constraints. The results show that the local search technique is more mature than the CP technique, which in turn should have much more room for improvement. However, an LNS scheme integrating both our CP and LS approaches improved some of the best results. Any improvement of the CP approach should lead to improvements on the LNS thus opening additional avenues for the CP technique.

The structure of the list-colouring graph made of large and overlapping cliques was shown to be important for both CP and LS techniques. Improving the propagation we can achieve from a collection of ALLDIFF constraints is very important in this context. Arc-consistency on two overlapping ALLDIFF is already known to be NP-Complete (Beldiceanu et al. 2007) but a number of pragmatic filtering rules could be designed. This is an important topic for future work.

References

Abdullah, S., Burke, E. K., & McCollum, B. (2005). Using a randomised iterative improvement algorithm with composite neighbourhood structures for course timetabling. In *MIC 05: the 6th meta-heuristic international conference*.

Aggoun, A., & Beldiceanu, N. (1993). Extending chip in order to solve complex scheduling and placement problems. *Mathematical Computing and Modelling*, *17*(7), 57–73.

Beldiceanu, N., Carlsson, M., Demassey, S., & Petit, T. (2007). Global constraint catalogue: past, present and future. *Constraints*, *12*(1), 21–62.

Cambazard, H., Hladik, P. E., Déplanche, A. M., Jussien, N., & Trinquet, Y. (2004). Decomposition and learning for a real time task allocation problem. In *Proc. of CP* (pp. 153–167).

Carter, M. W., & Laporte, G. (1997). Recent developments in practical course timetabling. In *PATAT* (pp. 3–19).

Chiarandini, M., Birattari, M., Socha, K., & Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *J. Scheduling*, *9*(5), 403–432.

Danna, E., & Perron, L. (2003). Structured vs. unstructured large neighborhood search: a case study on job-shop scheduling problems with earliness and tardiness costs. In *CP* (pp. 817–821).

de Siqueira, J. L., & Puget, J. F. (1988). Explanation-based generalisation of failures. In *European conference on artificial intelligence (ECAI'88)* (pp. 339–344).

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

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.

Galinier, P., & Hertz, A. (2006). A survey of local search methods for graph colouring. *Computers Operating Research*, *33*(9), 2547–2562.

Hooker, J. N., & Ottosson, G. (2003). Logic-based benders decomposition. *Mathematical Programming*, *96*, 33–60.

Jain, V., & Grossmann, I. E. (2001). Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on Computing*, *13*, 258–276.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Kostuch, P. (2004). The university course timetabling problem with a three-phase approach. In *PATAT* (pp. 109–125).

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, *2*(1), 83–98.

Lewis, R., Paechter, B., & McCollum, B. (2007). Post enrolment based course timetabling: a description of the problem model used for track two of the second international timetabling competition (Technical report). Cardiff University.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th design automation conference (DAC'01)* (pp. 530–535).

Perron, L., & Shaw, P. (2004). Combining forces to solve the car sequencing problem. In *CPAIOR* (pp. 225–239).

Perron, L., Shaw, P., & Furnon, V. (2004). Propagation guided large neighborhood search. In *CP* (pp. 468–481).

Refalo, P. (2004). Impact-based search strategies for constraint programming. In *CP* (pp. 557–571).

Régin, J. C. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th national conference on artificial intelligence (AAAI-94)* (pp. 362–367).

Régin, J. C. (1996). Generalized arc consistency for global cardinality constraint. In *National conference on artificial intelligence (AAAI'96)* (pp. 209–215).

Régin, J. C. (1999). Arc consistency for a global cardinality constraints with costs. In *Principles and practise of constraint programming (CP'99)* (pp. 390–404).

Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L. M., Knowles, J. D., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., & Stützle, T. (2002). A comparison of the performance of different metaheuristics on the timetabling problem. In *PATAT* (pp. 329–354).

Sadeh, N., & Fox, M. S. (1996). Variable and value ordering heuristics for the job-shop scheduling constraint satisfaction problem. *Artificial Intelligence*, *86*(1), 1–41.

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

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *CP* (pp. 417–431).