A Hybrid Algorithm for the Examination Timetabling Problem

Liam T.G. Merlot[1], Natashia Boland[1], Barry D. Hughes[1], and Peter J. Stuckey[2]

[1] Department of Mathematics and Statistics,
The University of Melbourne, Victoria 3010, Australia
{merlot,natashia,hughes}@ms.unimelb.edu.au
[2] Department of Computer Science and Software Engineering,
The University of Melbourne, Victoria 3010, Australia
pjs@cs.mu.oz.au

Abstract. Examination timetabling is a well-studied combinatorial optimization problem. We present a new hybrid algorithm for examination timetabling, consisting of three phases: a constraint programming phase to develop an initial solution, a simulated annealing phase to improve the quality of solution, and a hill climbing phase for further improvement. The examination timetabling problem at the University of Melbourne is introduced, and the hybrid method is proved to be superior to the current method employed by the University. Finally, the hybrid method is compared to established methods on the publicly available data sets, and found to perform well in comparison.

1 Introduction

The difficulty of developing appropriate examination timetables for tertiary education institutions is increasing. Institutions are enrolling more students into a wider variety of courses including an increasing number of combined degree courses. For example, at the University of Melbourne, approximately 20 000 students have to be fitted into about 650 exams over a two and a half week period. For these 20 000 students, there exist approximately 8000 different individual examination timetables. Consequently, examination timetabling is a difficult combinatorial optimization problem and too complex an issue to be resolved by manual means. Appropriate algorithms are required to provide adequate examination timetables for universities.

The development of an examination timetable requires the institution to schedule a number of examinations ("exams") in a given set of exam sessions ("time slots", or simply "sessions"), so as to satisfy a given set of constraints. A common constraint for universities is that no student may have two exams scheduled at the same time. However, some universities allow a student to have two examinations scheduled at the same time (a "clash"), as long as an appropriate arrangement can be made (such as "quarantining" students between exams). This is the situation at the University of Melbourne, where every semester

E. Burke and P. De Causmaecker (Eds.): PATAT 2002, LNCS 2740, pp. 207–231, 2003.

students are scheduled with two exams in the same session. As quarantining is expensive and inconvenient, we propose a different examination timetabling method for the University of Melbourne that avoids these "clashes".

This paper discusses key features of examination timetabling problems and reviews existing methods for publicly available and other data sets in Section 2. A new hybrid exam scheduling method is presented in Section 3. This hybrid method seeks good-quality schedules, but attempts to avoid unnecessary clashes. The new method is a combination of constraint programming, simulated annealing and hill climbing (local search). Details of the problem for the University of Melbourne are given in Section 4, while benchmark problems in the literature are discussed in Section 5. The hybrid method is demonstrated in Section 4 to be superior to the current timetabling system used by the University of Melbourne, and in Section 5 to be superior or comparable to well-known existing methods, measured against established benchmarks.

2 Previous Work on Examination Timetabling

2.1 Examination Timetabling Problems

The primary form of the exam timetabling problem faced by educational institutions is to allocate a session and a room to every exam, so as to satisfy a given set of constraints. The result is a feasible exam timetable. However, each institution will have some unique combination of constraints, as policies differ from institution to institution. Furthermore, institutions may take different views on what constitutes the *quality* of an exam timetable. In some cases, any feasible timetable will do, while in other cases, timetables exhibiting desirable features are sought. This makes it difficult to give a universal definition of exam timetabling, but, although the exact nature of the constraints and quality measures tends to be unique to individual institutions, they tend to take on only a limited number of forms.

The most common forms of constraint are

1. Clashing: no student may have two exams in the same session.
2. Capacity: the total number of students sitting in all exams in the same session in the same room must be less than the capacity of the room.
3. Total Capacity: the total number of students sitting in all exams in the same session must be less than the total capacity for that session.
4. Exam Capacity: the total number of exams in the same session must be less than some specified number.
5. Exam Availability: some exams are preassigned to specific sessions or can only be held in a limited set of sessions.
6. Room Availability: some rooms are only available in specific sessions.
7. Pairwise Exam Constraints: some pairs of exams must satisfy pairwise scheduling constraints (e.g., one must be held before the other).
8. Exam/Room Compatability: some exams may require specific rooms.

9. Student Restrictions: there may be restrictions on students' individual examination timetables (e.g., no student can have two exams scheduled in three consecutive sessions).
10. Large Exams: large exams should be held earlier in the exam period (e.g., exams with more than 500 students must be held in the first 10 sessions).

Each institution will apply some or all of these constraints. The exact form will be dependent on the institution, and some may be treated as soft constraints (constraints that hold where possible, but can be violated). For example the University of Melbourne treats Clashing and Large Exams as soft constraints. Quality measures (or objectives) of a solution are usually derived from soft constraints, most frequently from Student Restrictions. For example, the number of clashes (instances of a student with two exams scheduled in the same session) is a quality measure for the University of Melbourne, as is the number of instances of a student with an exam scheduled in both the morning and afternoon sessions of the same day. If several different quality measures are used simultaneously, the objective is a linear combination of these measures, with relative weights that reflect their perceived importance.

For some institutions (including the University of Melbourne), the allocation of rooms to the exams in a given session is a secondary problem: exam rooms may be large, or exams easily split between rooms. In these cases, the assignment of sessions to exams has only to respect the total capacity constraint for each session, and the assignment of exams to specific rooms can be done later as a separate activity. Not all institutions are so fortunate.

2.2 Previous Methods

In this section, we review some influential and recent methods for solving exam timetabling problems. Often, different methods have addressed somewhat different versions of the exam timetabling problem, with different constraints and quality measures. Quality measures are often combined to form a mathematical *objective* for the problem, and methods which optimize with respect to that objective developed. We discuss variations of the problem encountered in the literature in more detail in Section 5; here we focus on the methods that have been applied.

Surveys of different methods for exam timetabling by Burke et al. [2] and Carter and Laporte [8] classify the different approaches as cluster methods, sequential construction heuristics, constraint programming, and local search (genetic algorithms, memetic algorithms, simulated annealing and tabu search). In recent years, Carter[1] and Burke[2] have made data sets for exam timetabling publicly available via the internet. Only three different approaches, that we are aware of, have been applied to these publicly available data.

Sequential construction heuristics have been applied to the publicly available data in a variety of forms by Burke et al. [6], Carter et al. [9,10] and Caramia

[1] ftp://ftp.mie.utoronto.ca/pub/carter/testprob
[2] ftp://ftp.cs.nott.ac.uk/ttp/Data

et al. [7]. Sequential construction heuristics order the exams in some way (for example, largest exam first), and attempt to allocate each exam to a session in order, while satisfying all the constraints. The different heuristics feature different orders. They also have other differences: Carter et al. [10] allow limited backtracking (deallocation of exams), Burke et al. [6] select exams from a randomly chosen subset of all exams, and Caramia et al. [7] include an optimization step after each exam allocation.

Burke et al. use memetic algorithms for exam timetabling [3,4]. In Burke et al. [4] an initial pool of timetables is generated via a random technique, which attempts to group together exams with similar sets of conflicting exams. Then timetables are randomly selected from the pool, weighted by their objective value, and mutations are applied by rescheduling randomly chosen exams, or all exams in a randomly chosen session. Finally, hill climbing (local search) is applied to the mutated timetable to improve its quality. The process continues with the new pool of timetables. Burke and Newall [3] improve upon their earlier work by applying the memetic algorithm only to the first k exams as defined by a sequential construction method ordering. After the best timetable for the first k exams is found, the exams are fixed in place, and the memetic algorithm applied for the next k exams, until all are fixed.

White and Xie [19] and Di Gaspero and Schaerf [13] use tabu search methods. White and Xie keep two tabu lists, the usual short-term tabu list, and a long-term tabu list which keeps track of the most-moved exams. Di Gaspero and Schaerf [13] use a single tabu list, but when exams are added to this list it is for a randomly determined number of iterations. They also modify the objective function as the algorithm progresses.

There is a considerable body of work on exam and other timetabling problems, which has not been applied to the publicly available data sets. The most closely related to our work appear to be the constraint programming approach used by Boizumault et al. [1] and the simulated annealing approaches explored by Dowsland and Thompson [11,12,16,17,18]. The principal innovation in our work, compared to these others, is the sequential use of these two methods as the first two stages of a total strategy. A similar sequential approach has been taken in work on other problems: White and Zhang [20] use constraint programming to find a starting point for tabu search in solving course timetabling problems, and for high school timetabling Yoshikawa et al. [21] test several combinations of two-stage algorithms, including a greedy algorithm followed by simulated annealing and a constraint programming phase followed by a randomized greedy hill climbing algorithm (which is deemed to be the best combination of those used). In a similar vein, Burke et al. [5] use their work on sequential construction heuristics [6] to generate initial solutions for their memetic algorithm [4].

Our approach is to apply constraint programming, simulated annealing and hill climbing in turn. Although each of these stages is closely related to an existing method, the combination we develop is new, and appears to be particularly effective in practice, as we show later. Here, we motivate the method we use at

each stage and comment on the similarities of each to methods described in the published literature.

Firstly, our constraint programming approach is similar to that of Boizumault et al. [1]; in fact our approach can be viewed as a simplification, tailored to finding a feasible solution very quickly rather than solving the entire problem. Other researchers, notably Thompson and Dowsland [16,17,18] have used alternative first-stage methods. Thompson and Dowsland use simulated annealing with a simple neighbourhood to solve a first-stage problem. As in our first stage, overall seating capacity and exam availability (time window) constraints are satisfied. However, we do not permit clashes, but allow some exams to remain unscheduled, if needed, whereas Thompson and Dowsland schedule all exams and seek to minimize the number of clashes. Whilst a direct comparison of these two approaches would be interesting, we have found constraint programming to be very effective. As we discuss in Sections 4 and 5, it is highly successful at providing suitable starting points for our simulated annealing stage and tends to provide compact solutions, using near-minimal number sessions. Running times have never been more than a few seconds on any data set tested.

Secondly, our simulated annealing stage is very similar to the later-stage simulated annealing method of Thompson and Dowsland [16,17,18] which they demonstrated to be very effective on problems similar to our problem class P4, discussed in Section 5.5. Both simulated annealing methods use a Kempe chain neighbourhood and geometric cooling (for more details see Section 3.2). Thompson and Dowsland select a neighbour by choosing two sessions and one exam from the first of these sessions at random and seek to move the selected exam to the second session, but have also experimented with our approach: we choose an exam at random and select a new session from those available for that exam. Thompson and Dowsland observe that when there is little slack in the total seating capacity constraint, the two sampling methods perform similarly. We believe that in cases where the set of sessions available for each exam is quite limited, (as occurs at the University of Melbourne), the latter choice will be more efficient; it will use less time testing infeasible options. We also fix the number of iterations performed at each temperature to a constant. This is one of many cooling schedules tested by Thompson and Dowsland.

Our third stage uses hill climbing to improve the final solution from the simulated annealing stage. The idea of a final hill climbing solution refinement stage is not new. For example, the approach used by Schaerf [15] for high school timetabling combines a metaheuristic with hill climbing: tabu search based on a small neighbourhood employs a (randomized) hill climbing method[3], based on a larger neighbourhood, to improve any local minimum (with respect to the smaller neighbourhood) encountered by the tabu search. Our approach is in some sense the reverse of this: our hill climbing method exhaustively searches a neighbourhood which is in some sense smaller than that used by our simulated annealing stage; the neighbourhood used in simulated annealing is actually very

[3] A neighbour is generated at random. It is accepted if it is at least as good as the current solution.

large, but of course simulated annealing only selects a neighbour at random and does not search the entire neighbourhood.

3 A Three-Stage Method

We consider the exam timetabling problem in which a session must be allocated to each exam. We apply three of the constraints defined in Section 2.1: Clashing, Exam Availability and Total Capacity. We also treat the Large Exams constraint as a hard constraint, modelled via the Exam Availability constraint. Room allocation is not critical for the application of most interest to us (see Section 4), so we neglect it here. The approach we take to the exam timetabling problem consists of three stages, yielding a hybrid method:

1. Constraint programming: to obtain a feasible timetable;
2. Simulated annealing: to improve the quality of the timetable;
3. Hill climbing: for further refinement of the timetable.

The first stage of the hybrid method is used primarily to obtain an initial timetable satisfying all the constraints. As we shall see below, our approach aims to achieve this using as few sessions as possible, and although all constraints will be satisfied by the resulting timetable, in some cases, some exams may remain unscheduled.

The second and third stages aim to improve the quality of the timetable, and to schedule as many exams as possible. The methods used in both these stages are optimization methods, which will seek to optimize a given objective function. For this purpose, we formulate an objective function which takes into account both aims. The measures of quality, and hence the objective function formulated, differ according to the particular data set used, but typically the objective function is some combination of penalties applied for proximity of exams (time-wise) in students' timetables. We discuss the objective functions, which we call *objective scores*, used in detail in the context of the data sets considered, in Sections 4 and 5.

It is possible (although rare for data sets we tested) that some exams remain unscheduled after all three stages. In this case, we employ a simple greedy heuristic to schedule the remaining exams. We discuss this heuristic in Section 3.4.

3.1 Constraint Programming

Constraint programming is used to find a first feasible timetable in our hybrid method. Our constraint programming model is defined as follows. We use the notation below:

- $E = \{1, \ldots, n\}$ denotes the given set of n exams,
- s_i is the number of students in exam i, for all $i \in E$,
- $T = \{1, \ldots, v\}$ denotes the set of v sessions,
- $R \subset E \times T$ represents the set of given exam-session restrictions, so that $(a, b) \in R$ indicates that session b cannot be allocated to exam a,

- C_t is the total capacity of session t: that is, the total number of students who can sit exams in session t, for all $t \in T$,
- D_{ij} is the number of students enrolled in both exams i and j, for all $i, j \in E$,
- variable $x_i \in T$ indicates the session allocated to exam i, for all $i \in E$.

Note that the sessions are assumed to be time ordered, so $t_1 < t_2$ for $t_1, t_2 \in T$ if and only if session t_1 occurs before session t_2. We also define the *domain* of each variable x_i to be the set of all sessions that can feasibly be allocated to exam i. Initially, the domain of x_i is the set of sessions $t \in T$ such that $(i, t) \notin R$. During a constraint programming search, the domain of variables may be reduced by the removal of sessions, so as to ensure some form of consistency with respect to the constraints. The initialization of the variable domains ensures that the Exam Availability constraint is satisfied. The Clashing constraint is modelled in our constraint program as follows:

$$x_i \neq x_j \text{ for all exams } i, j \in E \text{ with } i \neq j \text{ and } D_{ij} > 0\,.$$

The Total Capacity constraint is modelled as

$$\sum_{i \in E} s_i(x_i = t) \leq C_t \text{ for all sessions } t \in T\,.$$

Here $(x_i = t)$ is a Boolean switch, taking on value 1 if $x_i = t$, and 0 otherwise.

A typical constraint programming method, applied to the above exam timetabling model, will operate as follows. An exam is chosen, and a session in its domain is allocated to it: that is, all sessions except one are removed from the exam's domain. In this case we say the exam has been *scheduled*; otherwise it is *unscheduled*. Consistency of the domains of all variables with respect to the constraints is then checked. For example, any exam which clashes with the chosen exam should have the session allocated to the chosen exam removed from its domain. After consistency is checked, another exam is chosen, and is allocated a session from its domain. This process is repeated until either all exams have been allocated a session, or until infeasibility has been detected, usually as a result of the domain of some variable becoming empty. In this case, the method backtracks and re-allocates sessions to some exams. Clearly, there is a lot of flexibility in this method: which exam is chosen at each step, and which session in its domain is chosen to be allocated to it? The precise form of these choices determines the *search strategy* of the constraint programming method.

Our experiments revealed that the best search strategy for our exam timetabling problem is to choose the unscheduled exam with the smallest current domain size, that is, the exam with the smallest domain size greater than one, and to allocate it the earliest session in its domain. Other search strategies explored were: choosing exams based on the number of students in each exam, choosing exams based on the number of conflicting exams, and choosing sessions based on the total space remaining given the current allocations (most or least)[4].

[4] Boizumault et al. [1] found for their data that the best search strategy was to choose the exam with the largest number of students and the session with the most available space.

This constraint programming model and search strategy were implemented using the ILOG package OPL [14]. As the sessions were chosen in order (exams were scheduled as early as possible), the timetables found tended to use a number of sessions close to the minimum number required (and often less than the maximum number allowed). In order to take advantage of this result, and consequently speed up the overall algorithm, additional "dummy" sessions are included (with a capacity equal to the maximum used for all other sessions), which are feasible for all exams, and occur "after" the final session v. Any exams allocated these dummy sessions by OPL are interpreted as unscheduled. A component of the objective in the simulated annealing phase of the algorithm encourages scheduling of unscheduled exams. In Section 5.3, 12 different data sets are used, and for these, two data sets required one dummy session and one required two dummy sessions, but after the simulated annealing phase there were no exams remaining in these dummy sessions.

Our constraint programming approach thus generates timetables satisfying the given constraints, with the proviso that in some cases, some exams may remain unscheduled (although the search strategy we use ensures this is rare).

As the timetables produced by this process are not subject to any quality measures, they are usually of very poor quality. Optimization of exam timetables using constraint programming proved impractical: our experiments showed that the large search space and weak pruning of the minimization constraints made it difficult for a constraint programming method to significantly improve timetable quality. However, as we shall see later, a simulated annealing method *is* able to make substantial improvements.

3.2 Simulated Annealing

The timetable produced by the constraint programming algorithm is used as the starting point for the simulated annealing phase of the hybrid method. This phase is used to improve the quality of the timetable.

In what follows, we refer to a candidate timetable, together with any unscheduled exams (exams scheduled in dummy sessions), as a *solution*. In all cases, all scheduled exams in any solution will satisfy the Clashing, Total Capacity, and Exam Availability constraints, so all exams allocated a given session will not conflict with each other, the total number of students sitting exams allocated the same session will not exceed the capacity of the session, and no exam will be allocated a session for which it is not available.

The number of sessions used for the solutions in the simulated annealing phase (and hill climbing phase) is equal to the number of sessions specified by the problem plus the number of dummy sessions used in the solution generated by the constraint programming phase.

A key component of any simulated annealing method is the *neighbourhood* of a solution. The neighbourhood we use is a slight variant on the Kempe chains neighbourhood used by Thompson and Dowsland [18]. A Kempe chain is determined by an exam i currently allocated session t, and another session $t' \neq t$. Let G be the set of all exams allocated session t, and G' be the set of exams

allocated session t'. Note that our definition of a solution ensures that both G and G' are conflict-free sets of exams. The Kempe chain can be thought of as the (unique) minimal pair of sets of exams, $F \subseteq G$ and $F' \subseteq G'$, such that $i \in F$ and both $(G \setminus F) \cup F'$ and $(G' \setminus F') \cup F$ are conflict-free sets of exams. For a given exam i in session t and other session t', the Kempe chain (F and F') can easily be constructed by a simple iterative procedure. We call the timetable obtained by (re-)allocating session t to all exams in F' and (re-)allocating session t' to all exams in F a *neighbour* of the solution. The *neighbourhood* of a solution is defined to be the set of all such neighbours.

In our simulated annealing algorithm, a current solution is maintained, and a neighbour of the current solution chosen at random. We choose a neighbour by selecting an exam i at random from the set of all exams, and selecting a session t' at random from the set of sessions available for i, i.e. with $(i, t') \notin R$, such that $t' \neq t$, where t is the session allocated to exam i in the current solution. Together i, t and t' induce a Kempe chain, and hence a neighbour of the current solution. (Thompson [18] selects two sessions at random, and randomly chooses an exam allocated the first session.)

Our simulated annealing method is quite standard. Once a neighbour has been generated (each generation of a neighbour is defined as an iteration), it is tested for feasibility (of Total Capacity and Exam Availability), and if feasible, the objective function value, or *score*, for this neighbour is calculated. If this neighbour is superior in quality to (has a lower score than) the current solution then the current solution is replaced by the neighbour. Otherwise the current solution may or may not be replaced by the neighbour, depending on the difference in scores and the current *temperature* (as is standard in simulated annealing). Let $o(p)$ denote the objective score for a solution p. Let x be the current solution and let q be the neighbouring solution to x selected at random at the current iteration. Let u be the current temperature. Then if $o(q) > o(x)$, the probability that q will replace x as the current solution is $e^{(o(x)-o(q))/u}$.

The score used varied by problem class (different problem classes have different objectives), so we define the scores used in Sections 4 and 5, where we document each problem class tested.

Thompson and Dowsland [16] discuss different cooling schedules for simulated annealing processes applied to exam scheduling problems. They find that slow cooling schedules are generally more effective, but that no cooling schedule is markedly better than any other. We use a geometric cooling schedule, in which at every a iterations, the temperature, u, is multiplied by α, where a and α are given parameters of the algorithm. Initial experimentation revealed that the parameter settings of a starting temperature of 30 000, $\alpha = 0.999$, $a = 10$ and a stopping temperature of 10^{-11}, giving approximately 350 000 iterations, were appropriate for all problems discussed in this paper.

Throughout the simulated annealing phase, the algorithm keeps the best solution found so far, and yields as output this best solution, at the conclusion of the algorithm. As with most heuristics, simulated annealing solutions will vary in quality depending on how long the algorithm is permitted to run. It was found

that solutions using the standard parameters, generated in under a minute of CPU time on an Alphastation XP900 (466 MHz), were of good quality for the University of Melbourne's examination timetabling problem.

We use simulated annealing to improve the solution determined by the constraint programming phase, and to schedule any remaining unscheduled exams. We could have used simulated annealing starting from a random timetable to satisfy all the constraints. Experiments with a purely simulated annealing approach performed poorly, since finding a feasible solution was in many cases quite difficult, and this overwhelms the search for a good solution. Indeed for one of our data sets there was no solution to the hard constraints as given, and we would never determine this using simulated annealing alone.

3.3 Hill Climbing

The hill climbing algorithm starts with an initial solution, x, called the "current" solution, with objective score $o(x)$. The algorithm processes each exam in some order, determined a priori: we chose the exam subject code order. For each exam, e, in this order, the algorithm considers every neighbour of x (using the Kempe chain definition of neighbour given in Section 3.2) in which e is allocated a different session to the session allocated to it in x. For all such neighbours feasible with respect to the Exam Availability and Total Capacity constraints, the objective score is calculated and the neighbour q with minimum objective score is found. If there are several choices for q, select q to move e to the earliest time. If $o(q) \leq o(x)$, the current solution x is replaced by q: that is, the algorithm sets $x := q$; otherwise x is unchanged. In either case, this completes the processing of e, and the algorithm moves on to the next exam in the given order.

The hill climbing algorithm does not necessarily stop once all exams have been processed: at this point we say that one "iteration" of the hill climbing algorithm has been completed. The hill climbing algorithm may well go on to again process all exams in the given order, several times over; it may perform many iterations. The hill climbing algorithm stops when either all neighbours generated during the last iteration had strictly larger objective scores than the current solution, or after a specified number of iterations. On the data sets we experimented with, there was usually no improvement in the objective score of the current solution after around seven iterations, and consequently 10 hill climbing iterations were deemed to be sufficient; we specified a limit of 10 iterations for the hill climbing algorithm.

Normally the hill climbing stage does not significantly improve the quality of solution produced by the simulated annealing phase. However, in approximately 5–10% of cases, the simulated annealing algorithm had not sufficiently explored the neighbourhood of its best solution. For example, the best solution encountered may have been found early when the temperature was high, and moves that increase the objective are more frequently accepted. The algorithm may move away and settle in an inferior local minimum with a higher objective as the temperature is reduced. When this happened, the hill climbing stage produced a significant improvement on the simulated annealing solution. The

overall effect of the hill climbing phase of the algorithm is to make the results more consistent between runs of the three-stage algorithm by improving the less satisfactory simulated annealing solutions.

3.4 A Greedy Heuristic for Scheduling Unscheduled Exams

The constraint programming stage may allocate dummy sessions to some exams (the exams remain unscheduled). Normally the simulated annealing stage will reallocate normal sessions to these exams. However, it is possible for unscheduled exams to remain after all three stages of the algorithm.

In all tests we have performed, the only cases where this occurred were in the University of Melbourne problem and then only in those data sets where a clash-free solution is impossible. Although the University of Melbourne does allow clashes, when clashes occur, another constraint, known as the "Three-in-a-Day" constraint, must be satisfied; we discuss that constraint in detail below. For the purpose of describing our heuristic, we simply assume that there is some set of *essential constraints* that must be satisfied even when clashes are allowed.

In the event that exams remain unscheduled at the conclusion of the three-stage method, we apply a greedy heuristic to schedule these exams. This heuristic will, of necessity, introduce clashes into the timetable, however it attempts to minimize these. Furthermore, it will ensure the timetable satisfies the essential constraints, and will leave an exam unscheduled rather than violate these constraints.

The heuristic proceeds as follows, where the unscheduled exams are considered in order of subject code. For each exam, e, left in a dummy session (that is, unscheduled), a normal session $t \in \{1, \ldots, v\}$ is chosen so that if exam e was scheduled in session t the essential constraints would hold, and so that the number of clashes introduced by scheduling exam e in session t is minimized over all such sessions. If there are no such sessions, then the exam remains unscheduled (though this never occurred in our tests).

4 The University of Melbourne Problem

At the University of Melbourne, there are 600–700 exams to be scheduled for each of two teaching semesters. The June exam period at the end of semester 1 normally has about 600 exams, to be scheduled in 26–30 sessions (13–15 days with morning and afternoon sessions). In the November exam period at the end of semester 2 there are more exams (at least 650), and these have to be scheduled in 30–34 sessions (15–17 days). There are five rooms in which exams can be held, two of these on campus (with capacities 135 and 405) and three at an off-campus venue (with capacities 540, 774 and 1170).

The constraints that the university imposes can be defined as Capacity, Total Capacity, Exam Availability, Room Availability and Large Exams. It is a matter of university policy that Large Exams is a soft constraint to be respected if possible, but in practice the university's current procedure takes Large Exams

as a hard constraint. The Large Exam constraint can thus be treated as an Exam Availability constraint. The university also has some Pairwise Exam Constraints, in that some pairs (or sets) of exams have common content and are required to be held at the same time. We simply combine such exams into a single exam.

As there are only five available rooms, and splitting exams between the off-campus rooms does not present any difficulty, room allocation is not considered a serious issue at the University of Melbourne. Consequently, the University of Melbourne's current software, as well as our algorithm, initially only attempts to allocate a session to every exam, and does not allocate exams to rooms. Thus we only use the Capacity and Room Availability constraints to determine the total capacity for each session, needed for the Total Capacity constraint.

Unlike most educational institutions, the University of Melbourne does not enforce the Clashing constraint. Instead the university enforces a Student Restrictions constraint, called *Three-in-a-Day*, that students cannot have three exams scheduled in the same day (recall there are two sessions per day). By appropriate quarantining, students with two exams scheduled concurrently can be accommodated. This does, however, incur inconvenience and expense. The authors believed that allowing clashes and resolving them by quarantining students was unnecessary, and implemented the tighter Clashing constraint that no student may have two exams scheduled at the same time (as is required by our hybrid algorithm). If, at the end of the three stages of the hybrid algorithm, some exams remain unscheduled, we use the greedy heuristic described in Section 3.4, with essential constraints Total Capacity, Exam Availability and Three-in-a-Day, to allocate sessions to these exams. In all our test cases, the greedy heuristic successfully scheduled all remaining exams.

Data from two different semesters were examined[5]. These were for semesters 1 and 2 of the 2001 academic year at the University of Melbourne (mel01s1 and mel01s2). The semester 1 data set (mel01s1) required 609 exams to be scheduled into 28 sessions (30 sessions, with two sessions lost in the middle to a public holiday), and the semester 2 data set (mel01s2) required 657 exams to be scheduled into 31 sessions. To cope with the Pairwise Exam constraint that some exams had to be held at the same time, such exams were combined into a single exam. This left 521 exams in mel01s1 and 562 exams in mel01s2.

The current University of Melbourne software (which we refer to for brevity as UM) is an unnamed and scantily documented VMS executable code specially written for the university about 10 years ago. Little is known about how the UM software works, but the authors conjecture from running it that it may use some form of simulated annealing algorithm.

The objective score used for University of Melbourne problems in our simulated annealing and hill climbing stages is calculated as follows. Recall that these stages maintain clash-free solutions, and that there are two sessions on each day, so no student can be sitting more than two exams allocated sessions on the same day. Using the notation of Section 3.1, we have normal exam sessions $T = \{1, \ldots, v\}$ ordered chronologically, and introduce dummy sessions

[5] The data sets are available at http://www.or.ms.unimelb.edu.au/timetabling

$T' = \{v+1, \ldots, v'\}$. For a normal session t, we use Boolean $w(t)$ true if t is the last session before a weekend or public holiday. For a given solution x, we calculate the objective score $o(x)$ as follows, where U is the penalty per student for unscheduled exams, w_{sd} is the penalty per student for two exams scheduled on the same day, w_{am} is the penalty per student for an exam scheduled in the afternoon followed by one scheduled in the morning of the next day, w_{g2} is the penalty per student for two exams scheduled with one session between them, and w_{ma} is the penalty per student for an exam scheduled in the morning of one day and an exam scheduled in the afternoon of the next day:

initialize $o(x) := \sum_{\{i \in E \,:\, x_i \in T'\}} U s_i$
for each exam $i \in E$ with $x_i \in T$ and $w(x_i)$ not true **do**
  for each exam $j \in E$ with $x_j \in T$, $x_i < x_j \leq x_i + 3$ and $D_{ij} > 0$ **do**
    if $x_j = x_i + 1$ **then**
      if x_i is a morning session **then** $o(x) := o(x) + w_{sd} D_{ij}$
      else (x_i must be an afternoon session) $o(x) := o(x) + w_{am} D_{ij}$
    else if $w(x_i + 1)$ not true **then**
      if $x_j = x_i + 2$ **then** $o(x) := o(x) + w_{g2} D_{ij}$
      else (x_j must equal $x_i + 3$)
        if $w(x_i + 2)$ not true **then** $o(x) := o(x) + w_{ma} D_{ij}$
      endif
    endif
  endfor
endfor

In all tests on University of Melbourne data reported here[6] we used $U = 10\,000$. After trying a number of different values for the penalty parameters, we found those that gave best results were $w_{sd} = 2$, $w_{am} = 1$, $w_{g2} = 0$, and $w_{ma} = 0$. So in fact we found that a much simpler score would have sufficed for this data.

In addition to the exam and student data sets, the university also provided the data for Exam Availability and Large Exams constraints. As the Large Exams constraint is treated as a hard constraint by the university's timetabling software, this was combined with the Exam Availability constraint in order to restrict the large exams to earlier times. Upon initial examination, it was discovered that due to the Large Exams being treated as a hard constraint, it was impossible to avoid clashes in the timetable for the mel01s2 data set. For example, exams with more than 500 students were required to be held in the first 14 exam sessions, and because of the number of exams in this category, it was not possible to find a clash-free solution. However, if these exams were allowed to be scheduled in the first 15 exam sessions, (one more than previously allowed), it was possible to produce a clash-free solution.

[6] If the parameter U is too small, feasible solutions are rarely produced. In experiments conducted for a range of suitably large values of U ($10 \leq U \leq 10^6$) on both University of Melbourne data and other data, we found that the actual size of this parameter was most likely not a significant factor (probability of significance: 0.17), but to ensure that feasible solutions are found often enough one should select $U > 100$.

Table 1. *Comparison between the University of Melbourne current software (UM) and our hybrid algorithm.* The best timetable was deemed to be the one with the fewest clashes, with ties resolved by choosing the lowest objective score. The corresponding score for the best timetable can exceed the average. Note that it was not possible to compare directly to the UM program for the mel01s1 data set, and consequently the hybrid method was only compared to the timetable produced by the university for that semester. This timetable had been manipulated afterwards to satisfy constraints not known when the timetable was produced, and was not minimizing the same objective

Data set			Hybrid	UM
mel01s1	521 exams	Best clashes	**0**	8
	28 sessions	Corresponding score	**1072**	2085
		Average clashes	0	–
		Average score	1175.0	–
		Average time (s)	56	–
mel01s2a	562 exams	Best clashes	**1**	**1**
	31 sessions	Corresponding score	**1448**	2384
		Average clashes	3.2	1.4
		Average score	1634.8	2298
		Average time (s)	74	1008
mel01s2b	562 exams	Best clashes	**0**	**0**
	31 sessions	Corresponding score	**1115**	3373
		Average clashes	0	1
		Average score	1300	2632.2
		Average time (s)	73	1008

In order to test the hybrid algorithm under the assumption that clash-free solutions were possible, two different data sets for this semester were used: one used the Exam Availability and Large Exams data sets that the university used for that semester, which forced clashes due to the Large Exams constraint (mel01s2a), and the second modified the Large Exams constraint slightly so that clash-free solutions were possible (mel01s2b). Five different runs were made by the hybrid algorithm on all data sets (with the standard simulated annealing parameters and 10 hill climbing iterations) and five different runs were made by the UM program on the mel01s2a and mel01s2b data sets using the university's parameters. The results are shown in Table 1.

It can be seen that the hybrid method is superior in terms of the objective and time. The algorithms were run on different machines: the hybrid algorithm on an XP900 Alphastation (466 Mhz CPU) and the UM algorithm on a DEC Alphastation 8200 (dual 300 Mhz CPU), but the time difference is still significant. The hybrid method was not designed to cope with clashes, and thus it is not as good at minimizing the number of clashes (on average) compared to the UM program when clashes are unavoidable. Conversely, in the potentially clash-free data sets, the university's program does not always provide a clash-free solution.

Table 2. *Papers on examination timetabling for publicly available data sets.* B1: Burke et al. [4], B2: Burke et al. [6], B3: Burke and Newall [3], Ca: Caramia et al. [7], C: Carter et al. [10], D: Di Gaspero and Schaerf [13] and W: White and Xie [19]. *Note that Burke et al. [6] (B2) use a slightly different version of the KFU-S-93 data set to that of the other authors; they use a different number of sessions

Data Set	Exams	P1	P2	P3	P4
CAR-F-92	543	C,Ca	C,Ca,D,W	B1,D,Ca	B2,B3,D
CAR-S-91	682	C,Ca	C, Ca,D	B1,D,Ca	–
EAR-F-83	190	C,Ca	C,Ca,D	–	–
HEC-S-92	81	C,Ca	C,Ca,D	–	–
KFU-S-93	461	C,Ca	C,Ca,D	B1,D,Ca	B2*,B3,D
LSE-F-91	381	C,Ca	C,Ca,D	–	–
PUR-S-93	2419	C,Ca	C,Ca	–	B3,D
RYE-S-93	486	C,Ca	C,Ca	–	–
STA-F-83	139	C,Ca	C,Ca,D	–	–
TRE-S-92	261	C,Ca	C,Ca,D	B1,D,Ca	–
UTA-S-92	622	C,Ca	C,Ca,D,W	B1,D,Ca	B2
UTE-S-92	184	C,Ca	C,Ca,D	–	–
YOR-F-83	181	C,Ca	C,Ca,D	–	–
NOTT	800	–	–	B1,D,Ca	B3,D

5 Benchmarks

5.1 Publicly Available Data

As stated in Section 2.2, Carter and Burke have made data sets for exam timetabling publicly available. For each of these data sets, attempts have been made to solve up to four different problems. The solutions to these problems provide benchmarks to compare different exam timetabling methods. The problem definitions, and consequently the initial benchmarks for these problems, are presented in the papers of Carter et al. [10] (problems P1 and P2), Burke et al. [4] (problem P3), and Burke and Newall [3] (problem P4). Below we discuss in detail each of these problems and the results of solving them with our method.

Four other papers compared alternative methods to the original benchmarks established by Burke et al. [3,4] and Carter et al. [10]: Burke et al. [6], Di Gaspero and Schaerf [13], Caramia et al. [7], and White and Xie [19]. In Table 2 the publicly available data sets are listed, along with the papers that have tested methods on each problem. It should be noted that the computing resources used differ. We have not been able to convert the reported run times in seconds to equivalent run times for a common standard, so all tabulated times reported below should be taken as indicative only.

5.2 Problem P1: Graph Colouring Benchmarks

Carter et al. [10] look at several of the publicly available data sets, and attempt to find the minimum number of sessions required for a feasible timetable sub-

Table 3. *Problem P1: graph colouring benchmarks.* The constraint programming stage of the hybrid algorithm yields timetables with close to the minimum number of sessions. The time reported for the methods of Carter et al. [10] and Caramia et al. [7] is the time for the run producing the best result. The time for the hybrid algorithm is the average time per run. Reported times have not been converted to account for different computing resources. Data sets are listed in order of number of exams to give an indication of order of difficulty

Data set	No of exams		Hybrid stage 1	Carter et al.	Caramia et al.
HEC-S-92	81	Best	18	**17**	**17**
		Range	–	17–18	17–18
		Time (s)	0.36	0.5	10.6
STA-F-83	139	Best	**13**	**13**	**13**
		Range	–	13–13	13–13
		Time (s)	0.62	2.7	10.2
YOR-F-83	181	Best	23	**19**	**19**
		Range	–	19–21	19–21
		Time (s)	1.05	190.4	226.2
UTE-S-92	184	Best	11	**10**	**10**
		Range	–	10–10	10–10
		Time (s)	0.75	1.6	24.3
EAR-F-83	190	Best	24	**22**	**22**
		Range	–	22–24	22–23
		Time (s)	1.2	8.7	86.3
TRE-S-92	261	Best	21	**20**	**20**
		Range	–	20–23	20–23
		Time (s)	1.03	32.8	214.7
LSE-F-91	381	Best	18	**17**	**17**
		Range	–	17–18	17–18
		Time (s)	2.45	78.0	9.6
KFU-S-93	461	Best	21	**19**	**19**
		Range	–	19–20	19–20
		Time (s)	3.4	97.2	159.6
RYE-S-93	486	Best	22	**21**	**21**
		Range	–	21–23	21–23
		Time (s)	3.95	343.8	225.9
CAR-F-92	543	Best	31	**28**	**28**
		Range	–	28–32	28–32
		Time (s)	5.37	227.2	559.2
UTA-S-92	622	Best	32	32	**30**
		Range	–	32–35	30–34
		Time (s)	6.39	272.3	1023.5
CAR-S-91	682	Best	30	**28**	**28**
		Range	–	28–35	28–32
		Time (s)	7.34	75.1	86.3

ject only to the Clashing constraint. They set no Capacity or Total Capacity constraints, and thus the problem reduces to a graph colouring problem. The sequential construction heuristic of Carter et al. [10] produces a variety of solutions, depending on the order in which the exams were processed. Using a different sequential construction heuristic, Caramia et al. [7] managed to produce equal (and superior in one case) results to Carter et al. [10].

The constraint programming stage of the hybrid method produces solutions that tend to use a relatively small number of sessions, as we demonstrate in Table 3. Here we compare the number of sessions used in the solution produced by the constraint programming stage with the results of Carter et al. (Table 3 in [10]) and Caramia et al. (Table 1 in [7]). The results of the comparison can be seen in Table 3.

The constraint programming stage of the hybrid method only produced one solution with as few sessions as the best of the Carter et al. [10] and Caramia et al. [7] methods. However, in all but one data set, the constraint programming phase produced a solution with only one or two more sessions than the minimum attained by the other methods, and never used more than four additional sessions. These results confirm that the constraint programming stage produces timetables with close to the minimum number of sessions.

5.3 Problem P2: Uncapacitated Benchmarks

In addition to their work on graph colouring benchmarks, Carter et al. [10] developed a second uncapacitated version of the examination timetabling problem. They set a maximum number of sessions, and devise an objective function designed to favour timetables which space out students' exams. The objective function applies a penalty w_t to a timetable whenever a student has to sit two exams scheduled t periods apart, with $w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$ and $w_5 = 1$. The total penalty is divided by the number of students to get an average penalty per student; this is the value of the objective function for the given timetable. No account was taken of weekends and there was no differentiation between consecutive exam periods within the same day, versus overnight.

For this problem class, our hybrid algorithm uses an objective score identical to the objective function defined by Carter et al. [10], calculated as follows:

initialize $o(x) := \sum_{\{i \in E \,:\, x_i \in T'\}} Us_i$
for each exam $i \in E$ with $x_i \in T$ **do**
  for each exam $j \in E$ with $x_j \in T$, $x_i < x_j \leq x_i + 5$ and $D_{ij} > 0$ **do**
    $o(x) := o(x) + w_{(x_j - x_i)} D_{ij}$
  endfor
endfor

Caramia et al. [7], Di Gaspero and Schaerf [13] and White and Xie [19] compared their algorithms to that used by Carter et al. [10], using the same problem definition, on some or all of the P2 data sets. The method of Caramia et al. [7] proved to be superior on 10 of the 13 data sets, equal to Carter et al. [10]

Table 4. *Problem P2: uncapacitated benchmarks.* The number of sessions is restricted, but there are no capacity constraints. Scores listed under "best" and "average" are per student scores. For the hybrid method, this is obtained by dividing the final score by the number of students. Times reported are average times per run for the hybrid method, and times of the best run for Carter et al. [10] and Caramia et al. [7]. Reported times have not been converted to account for different computing resources. Data sets are listed in order of number of exams to give an indication of order of difficulty

Data set			Hybrid	Carter et al.	Caramia et al.	Di Gaspero and Schaerf	White and Xie
HEC-S-92		Best	10.6	10.8	**9.2**	12.4	–
Exams	81	Average	10.7	15.04	–	12.6	–
Sessions	18	Time (s)	5.4	7.4	11.0	–	–
STA-F-83		Best	**157.3**	161.5	158.2	160.8	–
Exams	139	Average	157.4	167.14	–	166.8	–
Sessions	13	Time (s)	5.1	5.7	6.5	–	–
YOR-F-83		Best	37.4	41.7	**36.2**	41.0	–
Exams	181	Average	37.9	45.6	–	42.1	–
Sessions	21	Time (s)	30	271.4	125.4	–	–
UTE-S-92		Best	25.1	25.8	**24.4**	29.0	–
Exams	184	Average	25.2	30.78	–	31.3	–
Sessions	10	Time (s)	8.6	9.1	5.0	–	–
EAR-F-83		Best	35.1	36.4	**29.3**	45.7	–
Exams	190	Average	35.4	40.92	–	46.7	–
Sessions	24	Time (s)	26	24.7	29.3	–	–
TRE-S-92		Best	**8.4**	9.6	9.4	10.0	–
Exams	261	Average	8.6	10.78	–	10.5	–
Sessions	18	Time (s)	39	107.4	102.8	–	–
LSE-F-91		Best	10.5	10.5	**9.6**	15.5	–
Exams	381	Average	11.0	12.36	–	15.9	–
Sessions	18	Time (s)	35	48.0	92.8	–	–
KFU-S-93		Best	**13.5**	14.0	13.8	18.0	–
Exams	461	Average	14.0	18.76	–	19.5	–
Sessions	20	Time (s)	40	120.2	112.8	–	–
RYE-S-93		Best	8.4	7.3	**6.8**	–	–
Exams	486	Average	8.7	8.68	–	–	–
Sessions	23	Time (s)	70	507.2	89.4	–	–
CAR-F-92		Best	**4.3**	6.2	6.0	5.2	–
Exams	543	Average	4.4	7.04	–	5.6	4.7
Sessions	32	Time (s)	171	47	142.7	–	–
UTA-S-92		Best	**3.5**	**3.5**	**3.5**	4.2	–
Exams	622	Average	3.6	4.8	–	4.5	4.0
Sessions	35	Time (s)	233	664.3	589.4	–	–
CAR-S-91		Best	**5.1**	7.1	6.6	6.2	–
Exams	682	Average	5.2	8.38	–	6.5	–
Sessions	35	Time (s)	296	20.7	34.7	–	–

on the UTA data set, with Di Gaspero and Schaerf [13] superior on the other two data sets.

The hybrid method described in this paper was run on 12 of the data sets, and the results are summarized in Table 4 (compared to results from Table 5 in Carter et al. [10], Table 3 in Caramia et al. [7], Table 1 in Di Gaspero and Schaerf [13] and Table 9 in White and Xie [19]). Note that for all these data sets, the hybrid algorithm produced a clash-free timetable within the maximum allowed number of sessions, without recourse to the fourth greedy heuristic stage. The hybrid method is superior to that of Di Gaspero and Schaerf [13] and to that of White and Xie [19], and better than Carter et al. on nine of 12 data sets (with two ties). However, the method of Caramia et al. [7] produces the best results with superior results to the hybrid method in six of the data sets (again with a tie on the UTA-S-92 data set).

5.4 Problem P3: Capacitated Benchmarks (Set 1)

Burke et al. [4] created a new class of capacitated problem for the publicly available data sets, with three sessions per weekday and a Saturday morning session. It was assumed that the exam period starts on a Monday. They set a maximum number of exam sessions and imposed Clashing and Total Capacity constraints. For the Nottingham data sets (NOTT), an Exam Availability constraint was also applied: exams over two hours in length had to be held in the first session of the day. The objective was to minimize the number of instances of a student having two exams in a row on the same day. For this problem class, our hybrid algorithm uses an identical objective score, calculated as follows:

initialize $o(x) := \sum_{\{i \in E \,:\, x_i \in T'\}} Us_i$
for each exam $i \in E$ with $x_i \in T$ and x_i not the last session of the day **do**
  for each exam $j \in E$ with $x_j = x_i + 1$ and $D_{ij} > 0$ **do**
    $o(x) := o(x) + D_{ij}$
  endfor
endfor

Di Gaspero and Schaerf [13] and Caramia et al. [7] applied their methods to this problem class. The results of Burke et al. [4] were bested by either Di Gaspero and Schaerf [13] or Caramia et al. [7] in every case, each with approximately half the best results on the data sets examined. Our hybrid method, with the objective score calculated as above, was run five different times for each data set. Note that for all these data sets, the hybrid method produced a clash-free timetable within the maximum allowed number of sessions, without recourse to the fourth greedy heuristic stage. The results can be seen in Table 5 (compared to results in Tables 1 and 5 in Burke et al. [4], Table 4 in Caramia et al. [7] and Table 2 in Di Gaspero and Schaerf [13]).

Clearly, the hybrid method is the superior method when applied to these capacitated versions of the data sets, providing the best solution in all instances. The method consistently provides lower scores than Di Gaspero and Schaerf [13],

Table 5. *Problem P3: capacitated benchmarks, set 1.* The objective to be minimized is the number of students with two consecutive exams on the same day. Times reported are average times per run for the hybrid method, times of the best run for Caramia et al. [7] and approximate run times for Burke et al. [4]. Reported times have not been converted to account for different computing resources. Data sets are listed in order of number of exams to give an indication of order of difficulty

Data Set			Hybrid	Burke et al.	Caramia et al.	Di Gaspero and Schaerf
TRE-S-92						
Exams	261	Best	**0**	3	2	4
Sessions	35	Average	0.4	–	–	5
Capacity	655	Time (s)	16	10800	222.4	–
KFU-S-93						
Exams	461	Best	**247**	974	912	512
Sessions	20	Average	282.8	–	–	597
Capacity	1995	Time (s)	45	24240	118.2	–
CAR-F-92						
Exams	543	Best	**158**	331	268	424
Sessions	31	Average	212.8	–	–	443
Capacity	2000	Time (s)	96	24240	80.4	–
UTA-S-92						
Exams	622	Best	**334**	772	680	554
Sessions	38	Average	393.4	–	–	625
Capacity	2800	Time (s)	173	24000	265.1	–
CAR-S-91						
Exams	682	Best	**31**	81	74	88
Sessions	51	Average	47	–	–	98
Capacity	1550	Time (s)	125	21120	31.4	–
NOTT						
Exams	800	Best	**2**	53	44	11
Sessions	26	Average	15.6	–	–	13
Capacity	1550	Time (s)	44	24240	359.1	–
NOTT						
Exams	800	Best	**88**	269	–	123
Sessions	23	Average	104.8	–	–	134
Capacity	1550	Time (s)	42	18000	–	–

which we believe is due to the choice of Kempe chain neighbourhood. The results provided by Burke et al. [4] are no longer competitive. However, it is important to note that on the Nottingham data sets, Burke et al. [4] set a Capacity constraint and allocated rooms to the exams. The hybrid method did not do this, whilst Di Gaspero and Schaerf [13] and Caramia et al. [7] do not state whether they do this or not. The extra constraints applied by Burke et al. [4] may be part of the reason for their relatively poor results on these data sets.

The major difference between the quality of the results produced in the uncapacitated version and the capacitated versions of these problems is between

our hybrid method and the algorithm of Caramia et al. [7]. In the uncapacitated version, the algorithm of Caramia et al. [7] is superior in seven of the data sets, and the hybrid method in four, but with the capacitated data sets, the hybrid method is superior in all six instances (Caramia et al. [7] do not attempt the Nottingham data set with 23 sessions). Of these six instances, three were data sets that the hybrid method was superior for the uncapacitated version and two had almost equal results (the Nottingham data set was not attempted as uncapacitated). Therefore, one would expect that the hybrid method would fare better with this comparison. However, the improvement cannot be explained by simply favourable data sets alone, as there is a significant difference in this problem for data sets that were comparable for problem P2.

5.5 Problem P4: Capacitated Benchmarks (Set 2)

Burke and Newall [3] build on previous work undertaken in Burke et al. [4], using a modified version of their memetic algorithm. They looked at some of the publicly available data sets, and optimized these with a different objective function. They consider a situation with three exam sessions per day on weekdays (morning, lunchtime, afternoon) and one on Saturday morning. The objective function considers only students with two exams in two consecutive sessions. They give a penalty of three per student for two exams in a row in the same day, and one per student for two exams in a row overnight. For this problem class, our hybrid algorithm uses an identical objective score, calculated as follows:

initialize $o(x) := \sum_{\{i \in E \,:\, x_i \in T'\}} Us_i$
for each exam $i \in E$ with $x_i \in T$ and x_i not on a Saturday **do**
  for each exam $j \in E$ with $x_j = x_i + 1$ and $D_{ij} > 0$ **do**
    if x_i is the last session of the weekday **then**
      $o(x) := o(x) + D_{ij}$
    else (x_i must be a morning or lunchtime session of a weekday)
      $o(x) := o(x) + 3D_{ij}$
    endif
  endfor
endfor

The Nottingham data set has the Exam Availability constraint that any exam over 2 hours in length must be held in a morning session.

Di Gaspero and Schaerf [13] compared their algorithm to the results produced by Burke and Newall [3]. Burke et al. [6], independent of their work on memetic algorithms, wrote another paper on sequential construction heuristics using the publicly available data. This method was not compared directly to any other work, though, as it was run on the CAR-F-92 data set with the same problem definition, it can be compared for this problem instance. Unfortunately, the other data this paper used were the KFU-S-93 set with a different number of sessions, and the UTA-S-92 data set, which have not been tested by other authors.

The hybrid method was run on the data sets used in Burke and Newall [3]. The results, together with the results of Burke and Newall [3], Burke et al. [6]

Table 6. *Problem P4: capacitated benchmarks, set 2.* The objective function is based on students with exams in consecutive sessions. Times reported are average times per run. Reported times have not been converted to account for different computing resources. Data sets are listed in order of number of exams to indicate order of difficulty

Data Set			Hybrid	Burke and Newall	Burke et al.	Carter et al.	Di Gaspero and Schaerf
KFU-S-93							
Exams	461	Best	**1337**	1388	–	2700	1733
Sessions	21	Average	1487.8	1608	–	–	1845
Capacity	1995	Time (s)	39	105	–	–	–
CAR-F-92							
Exams	543	Best	2188	**1665**	2555	2915	3048
Sessions	36	Average	2267.6	1765	–	–	3377
Capacity	2000	Time (s)	106	186	–	–	–
NOTT							
Exams	800	Best	731	**498**	–	918	751
Sessions	23	Average	841.2	544	–	–	820
Capacity	1550	Time (s)	44	467	–	–	–

(for the CAR-F-92 problem only), Carter et al. [10] and Di Gaspero and Schaerf [13], can be seen in Table 6 (using data from Tables 2–4 in Burke and Newall [3] and Table 3 in Di Gaspero and Schaerf [13]).

Clearly, the algorithm of Burke and Newall [3] is superior: for two of the data sets their results are clearly the best, and for the third they are only just behind our hybrid algorithm. It should be noted that the values for the memetic algorithm for Burke and Newall [3] in this table have been chosen as the best from five different runs with 18 different parameter settings. This is a total of 90 different runs for each data set, compared to five for the hybrid method. However, if the hybrid method is run many times, the scores do not improve significantly, as the method is tied to the solution produced deterministically by the constraint programming stage, which does not change with multiple runs. Instead, the hybrid method requires more simulated annealing and hill climbing iterations, to allow it to move further away from the initial solution, to improve the solution. To test this theory a series of longer runs were undertaken on these data sets. In addition to the standard run ($\alpha = 0.999$, $a = 10$: approximately 350 000 simulated annealing iterations and 10 hill climbing iterations), a medium-length run ($\alpha = 0.999$, $a = 100$: approximately 3500 000 simulated annealing iterations and 30 hill climbing iterations) and a long run ($\alpha = 0.999$, $a = 1000$: approximately 35 000 000 simulated annealing iterations and 100 hill climbing iterations) were performed on the data sets. The results can be seen in Table 7.

It is quite clear that the longer the hybrid algorithm is allowed to run, the better the quality of the solution. However, with the CAR-F-92 data set, even though the hybrid algorithm is allowed to run about 50 times longer than the memetic algorithm of Burke and Newall [3], the solution remains slightly inferior

Table 7. *Problem P4: longer hybrid runs.* Our experiments and those of Burke and Newall [3] are on similar but not identical machines

Data set			Hybrid standard	Hybrid medium	Hybrid long	Burke and Newall
KFU-S-93						
Exams	461	Best	1337	1182	**1082**	1388
Sessions	21	Average	1487.8	1255.6	1214.4	1608
Capacity	1995	Time (s)	39	348	3202	105
CAR-F-92						
Exams	543	Best	2188	1809	1744	**1665**
Sessions	36	Average	2267.6	1970.6	1801.4	1765
Capacity	2000	Time (s)	106	828	6671	186
NOTT						
Exams	800	Best	731	481	**401**	498
Sessions	23	Average	841.2	558.8	431.6	544
Capacity	1550	Time (s)	44	317	2818	467

(less than 5% worse). While it looks plausible that if the hybrid algorithm was allowed to perform an even longer run it would provide a superior solution, it is clearly an inferior method for this data set. However, for the other two data sets, the hybrid method does provide superior solutions in less time (a short run for KFU-S-93, and a medium run for NOTT). Unfortunately, with only three different comparisons between the methods, it is difficult to conclude definitively which method is superior.

6 Conclusions

The hybrid method for examination timetabling described in this paper is superior to the method currently used by the University of Melbourne, and performs well in comparison to other, well known methods that have been applied on the publicly available data sets. However, too few of these data sets have had benchmarks established on them (problem P4 has only been developed for five of the 14 data sets), and only three comparisons have been made between the hybrid method and the Burke and Newall [3] memetic algorithm with sequential construction. It is therefore not possible to make a definitive assessment of the relative quality of solutions found by the hybrid method and by existing methodologies. In addition, it is desirable to perform comparisons of the algorithms using the same computer resources.

In spite of the shortcomings of the comparisons, the hybrid method is still proven as a worthwhile algorithm, among the best currently in use for examination timetabling. The constraint programming stage provides a fast route to a first feasible solution. This solution is improved by the simulated annealing stage, with the Kempe chain neighbourhoods proving effective at diversifying

solutions (though occasionally more time is needed). The hill climbing stage further improves the solutions, and reduces the effect of unfavourable fluctuations in the simulated annealing stage.

We suggest that the dominant methods of the future for the examination timetabling problem will combine solution construction with local search. The stages of the hybrid method may be integrated more fully, to produce a still more powerful algorithm.

Acknowledgements. The authors would like to thank Aiden Tran and Gerry Barretto for providing the University of Melbourne data, and Michael Carter, Luca Di Gaspero and James Newall for help they gave in understanding their previous work.

References

1. Boizumault, P., Delon, Y., Peridy, L.: Constraint Logic Programming for Examination Timetabling. J. Logic Program. **26** (1996) 217–233
2. Burke, E.K., Jackson, K., Kingston, J., Weare, R.F.: Automated University Timetabling: the State of the Art. Comput. J. **40** (1997) 565–571
3. Burke, E.K., Newall, J.: A Multistage Evolutionary Algorithm for the Timetable Problem. IEEE Trans. Evolut. Comput. **3** (1999) 63–74
4. Burke, E.K., Newall, J., Weare, R.F.: A Memetic Algorithm for University Exam Timetabling. In: Burke, E, Ross, P. (Eds.): Practice and Theory of Automated Timetabling I (PATAT 1995, Edinburgh, Aug/Sept, selected papers). Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg New York (1996) 241–250
5. Burke, E.K., Newall, J., Weare, R.F.: Initialisation Strategies and Diversity in Evolutionary Timetabling. Evolut. Comput. J. (Special Issue on Scheduling) **6** (1998) 81–103
6. Burke, E.K., Newall, J., Weare, R.F.: A Simple Heuristically Guided Search for the Timetable Problem. Proc. Int. ICSC Symp. Eng. Intell. Syst. (1998) 574–579
7. Caramia, M., Dell'Olmo, P., Italiano, G.F.: New Algorithms for Examination Timetabling. In: Näher, S., Wagner, D. (Eds.): Algorithm Engineering 4th Int. Workshop, Proc. WAE 2000 (Saarbrücken, Germany, September) Lecture Notes in Computer Science, Vol. 1982. Springer-Verlag, Berlin Heidelberg New York (2001) 230–241
8. Carter, M., Laporte, G.: Recent Developments in Practical Examination Timetabling. In: Burke, E, Ross, P. (Eds.): Practice and Theory of Automated Timetabling I (PATAT 1995, Edinburgh, Aug/Sept, selected papers). Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg New York (1996) 373–383
9. Carter, M., Laporte, G., Chinneck, J.: A General Examination Scheduling System. Interfaces **24** (1994) 109–120
10. Carter, M., Laporte, G., Lee, S.T.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **47** (1996) 373–383
11. Dowsland, K.: Using Simulated Annealing for Efficient Allocation of Students to Practical Classes. In: Vidal, R.V.V. (Ed.): Applied Simulated Annealing. Lecture Notes in Economics and Mathematical Systems, Vol. 396. Springer-Verlag, Berlin Heidelberg New York (1993) 125–150

12. Dowsland, K.: Off-the-Peg or Made-to-Measure? Timetabling and Scheduling with SA and TS. In: Burke, E, Carter, M. (Eds.): Practice and Theory of Automated Timetabling II (PATAT 1997, Toronto, Canada, August, selected papers). Lecture Notes in Computer Science, Vol. 1408. Springer-Verlag, Berlin Heidelberg New York (1998) 37–52
13. Di Gaspero, L., Schaerf, A.: Tabu Search Techniques for Examination Timetabling. In: Burke, E, Erben W. (Eds.): Practice and Theory of Automated Timetabling III (PATAT 2000, Konstanz, Germany, August, selected papers). Lecture Notes in Computer Science, Vol. 2079. Springer-Verlag, Berlin Heidelberg New York (2001) 104–117
14. Van Hentenryck, P.: The OPL Optimization Programming Language. MIT Press, Cambridge, MA (1999)
15. Schaerf, A.: Tabu Search Techniques for Large High-School Timetabling Problems. Proc. Natl Conf. Am. Assoc. Artif. Intell. AAAI Press, Menlo Park, CA (1996) 363–368
16. Thompson, J., Dowsland, K.: General Cooling Schedules for a Simulated Annealing Timetabling System. In: Burke, E, Ross, P. (Eds.): Practice and Theory of Automated Timetabling I (PATAT 1995, Edinburgh, Aug/Sept, selected papers). Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg New York (1996) 345–363
17. Thompson, J., Dowsland, K.: Variants of Simulated Annealing for the Examination Timetabling Problem. Ann. Oper. Res. **63** (1996) 105–128
18. Thompson, J., Dowsland, K.: A Robust Simulated Annealing Based Examination Timetabling System. Comput. Oper. Res. **25** (1998) 637–648
19. White, G.M., Xie, B.S.: Examination Timetables and Tabu Search with Longer-Term Memory. In: Burke, E, Erben W. (Eds.): Practice and Theory of Automated Timetabling III (PATAT 2000, Konstanz, Germany, August, selected papers). Lecture Notes in Computer Science, Vol. 2079. Springer-Verlag, Berlin Heidelberg New York (2001) 85–103
20. White, G.M., Zhang, J.: Generating Complete University Timetables by Combining Tabu Search with Constraint Logic. In: Burke, E, Carter, M. (Eds.): Practice and Theory of Automated Timetabling II (PATAT 1997, Toronto, Canada, August, selected papers). Lecture Notes in Computer Science, Vol. 1408. Springer-Verlag, Berlin Heidelberg New York (1998) 187–198.
21. Yoshikawa, M., Kaneko, K., Nomura, Y., Watanabe, M.: A Constraint-Based Approach to High-School Timetabling Problems: a Case Study. Proc. Natl Conf. Am. Assoc. Artif. Intell. AAAI Press, Menlo Park, CA (1994) 1111–1116