OR Spectrum (2008) 30:167–190
DOI 10.1007/s00291-007-0097-0

REGULAR ARTICLE

A survey of metaheuristic-based techniques for University Timetabling problems

Rhydian Lewis

Published online: 11 July 2007

Abstract As well as nearly always belonging to the class of NP-complete problems, university timetabling problems can be further complicated by the often idiosyncratic requirements imposed by the particular institution being considered. It is perhaps due to this characteristic that in the past decade-or-so, metaheuristics have become increasingly popular in the field of automated timetabling. In this paper we carry out an overview of such applications, paying particular attention to the various methods that have been proposed for dealing and differentiating between constraints of varying importance. Our review allows us to classify these algorithms into three general classes, and we make some instructive comments on each of these.

Keywords Timetabling · Metaheuristics · Constraints

1 Introduction

University timetabling problems belong to the larger family of general timetabling and scheduling problems, where the aim is to assign a set of *entities* (such as tasks, public events, vehicles, or people) to limited number of *resources* over time, in such a way as to meet a set of pre-defined schedule requirements. Over the past decade-or-so there has been a large interest in applying *metaheuristic*-based algorithms to university

The author would like to give thanks to Ben Paechter at Napier University, Edinburgh, and Barry McCollum of Queens University, Belfast for providing the initial motivation for the production of this work. The author is also grateful to Peter Morgan, Bruce Curry, and Jonathan Thompson at Cardiff University and also an anonymous referee for their helpful comments and insights.

R. Lewis (✉)
Cardiff Business School, Pryfysgol Caerdydd/Cardiff University,
Colum Drive, Cardiff, CF10 3EU Wales, UK
e-mail: lewisR9@cf.ac.uk

timetabling problems, perhaps due to the fact that these problems can vary a great deal between institutions, and that metaheuristics are, by definition, types of "higher level" general-purpose algorithms that can be used with a wide range of different problem types. Yet, although standard metaheuristic methodologies define a basic overall behaviour of an algorithm, in this survey paper we will see that the various ways that timetabling algorithms can operate *within* these frameworks can also vary a great deal. Our intention is therefore to survey these differing features and to offer some useful comments about their relative strengths and drawbacks.

The generic definition of a university timetabling problem can be considered the task of assigning a number of *events*, such as lectures, exams, meetings, and so on, to a limited set of timeslots (and perhaps rooms), in accordance with a set of *constraints*. It is suggested by Corne et al. (1995) that the different sorts of timetabling constraints (of which there many) can be categorised into five main classes:

Unary constraints	that involve just one event, such as the constraint "event *a* must not take place on a Tuesday", or the constraint "event *a* must occur in timeslot *b*".
Binary constraints	that concern *pairs* of events, such the constraint "event *a* must take place before event *b*", or the *event clash* constraint, which is considered in more detail below. (Note that *chains* of binary constraints can also be applied, such as the constraint "event *a* must take place before *b*, which must take place before *c*", etc...)
Capacity constraints	that are governed by room capacities, etc. For example "All events should be assigned to a room which has a sufficient capacity".
Event spread constraints	that concern requirements such as the "spreading-out" or "clumping-together" of events within the timetable in order to ease student/teacher workload, and/or to agree with a university's timetabling policy.
Agent constraints	that are imposed in order to promote the requirements and/or preferences of the people who will use the timetables, such as the constraint "lecturer *x* likes to teach event *a* on Mondays", or "lecturer *y* must have have *n* free mornings per week".

A fundamental constraint in timetabling is the "event-clash" binary constraint. This specifies that if a person (or some other resource of which there is only one) is required to be present in a pair of events, then these must not be assigned to the same timeslot, as such an assignment will result in this person/resource having to be in two places at once. This particular constraint can be found in almost all university timetabling problems, and its presence allows parallels to be drawn with the well-known graph colouring problem (which we will look at in Sect. 2). Beyond this near-universal constraint, however, timetabling problem-formulations will usually vary widely from place-to-place due to the fact that different universities will usually have their own individual needs and timetabling policies (and therefore set of constraints) that they need satisfied. From an Operations Research point-of-view, this idiosyncratic

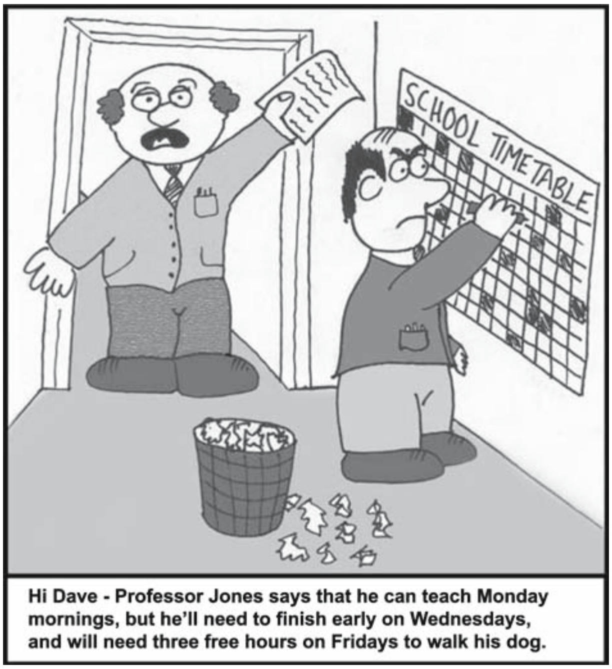

SCHOOL TIMETABLE
Hi Dave - Professor Jones says that he can teach Monday mornings, but he'll need to finish early on Wednesdays, and will need three free hours on Fridays to walk his dog.

Fig. 1 When constructing a timetable, meeting the needs of all concerned may not always be possible!

nature can often make it difficult when trying to formulate meaningful and universal generalisations about timetabling in general.

One convention in automated timetabling is to group the constraints being considered into two categories: the *hard* constraints and the *soft* constraints. Hard constraints have a higher priority than soft, and their satisfaction is usually mandatory. Indeed, timetables will normally only be considered "feasible" if *all* of the imposed hard constraints have been satisfied. Soft constraints, meanwhile, are those that we want to obey *if possible*, and more often than not they will describe the criteria for a timetable to be "good" with regards to the timetabling policies of the university concerned, as well as the experiences of the people who will have to use it (see Fig. 1).

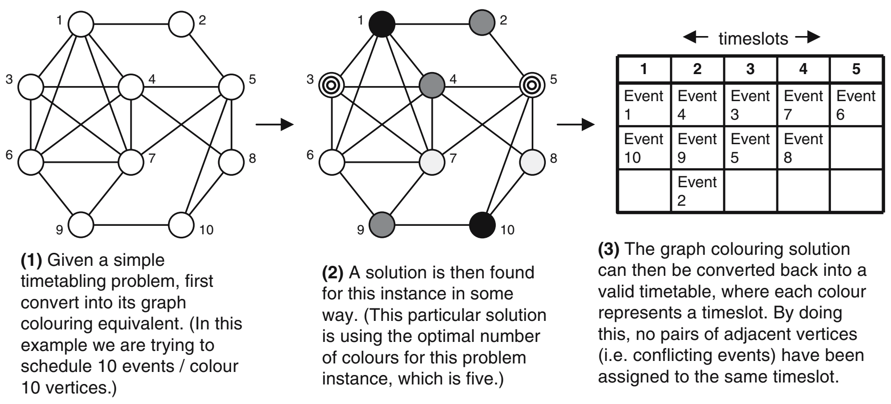
It is also widely accepted that university timetabling problems can be arranged into two main categories: exam timetabling problems and course timetabling problems. In reality, and depending on the university involved, both types of problem can often exhibit similar characteristics, but the main difference is that in exam timetabling, multiple events can be scheduled in the same room at the same time (providing seating-capacity constraints are not exceeded), whilst in course timetabling, we are generally only allowed one event per room, per timeslot. A second difference also sometimes occurs concerning the timeslots: in course timetabling we will generally be looking to assign all of the events to a *fixed* number of timeslots (e.g. those occurring in exactly one week); in exam timetabling though, we might have some flexibility in the number of timeslots that are available. (However, as we will see later, this is not always the case.)

A further generalisation that we can make about timetabling problems at universities is that they are NP-complete in almost all variants. What this means is that we cannot hope to find a polynomially bounded algorithm for solving them in general (unless, of course, P = NP, which is not generally considered to be the case—see the work of Garey and Johnson (1979) for more information).[1] Cooper and Kingston (1996), for example, have proved that NP-completeness exists for a number of different problem interpretations that can arise in practice. This, they achieve, by providing polynomially bounded transformations from various well-known NP-complete problems such as graph colouring (see Sect. 2), bin packing, and three-dimensional matching to a number of different timetabling problem variants. Even et al. (1976) have also shown a transformation of the NP-complete 3-SAT problem into a timetabling problem. Of course, this general NP-completeness of timetabling problems (or NP-hardness, if we are considering timetabling from an optimisation perspective) tells us that if we wish to obtain anything that might be considered a workable timetable in any sort of reasonable time, then this will depend very much on the nature of the problem instance being tackled. If, for example, an instance is encountered where the requirements are fairly "loose" (perhaps rooms are in abundance, or only a very small number of events actually need to be scheduled), then it might be that many acceptable timetables exist within the problem's search space, of which one or more can be discovered quite easily. On the other hand, some universities' requirements might be much more demanding, and perhaps only a very small proportion of the search space (if any) will be occupied by workable timetables. (It should be noted that, in practice, the combination of constraints that are imposed by timetabling administrators could result in problems that are *impossible* to solve unless some of the constraints are relaxed). Thus, in cases where "harder" timetabling problems are encountered, there seems an implicit need for powerful and robust methods for tackling these sorts of problems.

It is widely accepted that one way of dealing with NP-hard problems is to make use of *approximation algorithms* which, while not being able to guarantee an optimal solution, will hopefully produce a solution that is "good enough" for practical purposes in the majority of cases. In this paper we will conduct a review of works that have proposed such algorithms, paying particular attention to those approaches that have used *metaheuristic*-based techniques. We will argue that metaheuristics are indeed quite appropriate for this task, and will examine some of the different problem formulations that have been put forward in the literature, together with the various ways that metaheuristics have been proposed for solving them.

Note that other surveys on automated timetabling have also appeared in the literature in the past, including those concentrating on exam timetabling by Carter (1986b), Carter et al. (1996), Burke et al. (1995a, 1996a), and Burke and Petrovic (2002). Other, more general timetabling surveys—including those that give information on the related problem of timetabling in *schools* have also been contributed by Carter and Laporte (1998), De Werra (1985), Ross et al. (2003), and Schaerf (1999b).

[1] We use the word "solve" here in the strong sense: an algorithm only "solves" a problem when it is able to indicate the presence (or lack of) an optimal solution for *all instances* of a given problem.



1	2	3	4	5
Event 1	Event 4	Event 3	Event 7	Event 6
Event 10	Event 9	Event 5	Event 8	
	Event 2			

(1) Given a simple timetabling problem, first convert into its graph colouring equivalent. (In this example we are trying to schedule 10 events / colour 10 vertices.)

(2) A solution is then found for this instance in some way. (This particular solution is using the optimal number of colours for this problem instance, which is five.)

(3) The graph colouring solution can then be converted back into a valid timetable, where each colour represents a timeslot. By doing this, no pairs of adjacent vertices (i.e. conflicting events) have been assigned to the same timeslot.

Fig. 2 Demonstrating the relationship between the graph colouring problem and a simple timetabling problem where only the event-clash constraint is considered

2 Comparison to graph colouring

Before conducting this review, it is first useful (as it will certainly assist with various explanations within this paper) to provide a comparison between the timetabling problem and the well-known problem of graph colouring. Given a simple and undirected graph G comprising a set of n vertices $V = v_1, \ldots, v_n$ and a set of edges E which join various pairs of different vertices, the NP-hard graph colouring problem involves finding an assignment of "colours" for each vertex in V such that (a) no pair of vertices with a common edge is assigned the same colour, and (b) the number of colours being used is minimal.[2]

It is straightforward to convert the most simple of timetabling problems into a graph colouring problem (and vice-versa) by considering each event as a vertex, and then simply adding edges between any pair of vertices that correspond to pairs of events that cannot be assigned to the same timeslot. Each timeslot that is available then corresponds to a colour, and the task (from a graph colouring point-of-view) it to try and find a solution that uses no more colours than there are available timeslots. (See Fig. 2 for an example.)

One useful parallel that we can draw between these problems involves the identification of features known as "cliques". (A "clique" is simply a collection of vertices that are mutually adjacent, such as vertices 1, 3, 4, 6, and 7 in Fig. 2, which is a clique of size 5.) This is because graph colouring problems that reflect real-world timetabling instances will often contain large cliques, because it is typical to encounter large collections of events that must not be scheduled together (e.g. a first-year degree in Welsh might feature a large number of compulsory events that all first-year Welsh students

[2] See the work of Garey and Johnson (1979) (p. 133) for further details of this problem's NP-hardness. Also note that the NP-*complete* version of this problem defines a similar task but, as usual, it is expressed as a decision problem. Thus: given $G = (V, E)$ and a positive integer $k < n$, is it possible to assign a colour to each vertex in V so that no pair of adjacent vertices has the same colour, and only k colours are used? A proof of this decision problem's NP-completeness can be found in Garey et al. (1976).

are required to attend). In the equivalent graph colouring problem, the vertices that represent these events will form a clique, and it is easy to appreciate that all vertices within this clique must be assigned to different colours (or equivalently, all of the corresponding events will need to be assigned to different timeslots).

Some authors have also chosen to take this method of conversion one step further by incorporating other timetabling constraints into the graph colouring problem. For example, in the work of both Balakrishnan (1991), and Thompson and Dowsland (1998), the authors have converted their timetabling problems into a graph colouring problem in the usual manner, but have then also introduced k additional vertices representing each of the k available timeslots (in this case k needs to be fixed in advance). Edges are then added between each pair of these "timeslot-vertices" to form a clique of size k, and further edges are then also added between any event-vertices and timeslot-vertices where the corresponding event is unavailable. (For example, if we have the constraint "event a cannot be taught in timeslot b" then an edge is added between event-vertex a and timeslot-vertex b in the corresponding graph).

Beyond these examples, however, it is worth noting that conversions to graph colouring are only usually conducted when relatively simple constraints are being considered. Indeed, when other real-world constraints are considered in a problem (particularly those relating to the *ordering* of events within a timetable), then often the simple graph colouring model will not be sufficient on its own. Regardless of this factor though, it is still the case that nearly all timetabling problems do feature the underlying graph colouring problem in some form or another in their definitions, and it is certainly the case (as we will see) that many current timetabling algorithms do use various bits of heuristic information extracted from this underlying problem as a driving force in their searches for a timetabling solution.

3 Heuristic algorithms for university timetabling

Given the close relationship between graph colouring and university timetabling problems, it is unsurprising that many of the early algorithmic techniques for timetabling were derived directly from graph colouring-based heuristics. The algorithms of White and Chan (1979), and Carter et al. (1996), are prime examples of this.[3] Other timetabling approaches, meanwhile, have made use of constraint programming-based techniques, such as the work of Deris et al. (1997), Lajos (1996), Boizumault et al. (1996), Guéret et al. (1996), and Goltz and Matzke (1999). Further studies have also explored the application of integer programming, such as the algorithms of Carter (1986a) and Tripathy (1984) in the 1980s, and Daskalaki et al. in 2004. A particularly noteworthy example of integer programming can also be found in the recent study of Schimmelpfeng and Helber (2007), who have taken an interesting real-world timetabling problem from the School of Economics and Management at Hanover Univeristy, Germany, and have solved the resultant integer model using both an open source

[3] The latter example was also the study that introduced a set of problem instances that have since become somewhat of a benchmark in the exam timetabling community. Hereon, we will refer to these particular instances as the Carter Instances, which can be downloaded at http://www.or.ms.unimelb.edu.au/timetabling/atdata/carterea.tar.

mixed-integer solver and the commercial CPLEX solver. The results reported in this paper indicate that their methods, which require only very short run times, "can be used to create substantially better timetables than the school ever had" (Schimmelpfeng and Helber 2007).

As we have mentioned, in the past decade-or-so there has also been a surge in interest of applying *metaheuristic* algorithms to timetabling problems. According to the website of the Metaheuristics Network (http://www.metaheuristics.org), a pan-European research project that was run between 2000 and 2004, a metaheuristic "…can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications [being needed] to make them adapted to a specific problem". Given these characteristics, and remembering the idiosyncratic nature of timetabling problems that we have noted, it seems that metaheuristics are therefore quite suitable in this problem domain. On their website, the Metaheuristics Network also state that there are five main metaheuristic paradigms—namely Evolutionary Algorithms, Ant Colony Optimisation, Tabu Search, Simulated Annealing, and (Iterated) Local Search—and it will be these five methods that we will be concentrating on in this review. However, we appreciate that other techniques, such as Neural Networks and Hyper-heuristics, might also be considered metaheuristics by some, and it is worth mentioning that there have also been successful applications of these latter techniques to various timetabling problems in the past (Burke et al. 2003b, 2007; Carrasco and Pato 2004; Gislen et al. 1989; Kovacic 1993; Smith et al. 2003).

Considering timetabling problems in general, it is apparent that one aspect that separates these problems from many other types of combinatorial optimisation problem is the presence of both hard *and* soft constraints. In practical terms, for a metaheuristic approach to be worthwhile, it is therefore necessary for such an algorithm to be supplied with some sort of system for distinguishing the two. A survey of the timetabling literature with regards to this matter indicates that most metaheuristic algorithms for timetabling fall into one of three categories:

(1) **One-stage optimisation algorithms:** where a satisfaction of both the hard and soft constraints is attempted simultaneously (Abramson 1991; Abramson et al. 1996; Carrasco and Pato 2001; Colorni et al. 1991, 1992, 1997; Corne et al. 1995; Costa 1994; Di Gaspero and Schaerf 2002; Elmohamed et al. 1998; Hertz 1991; Melicio and Caldeira 1998; Schaerf 1996, 1999a; Smith et al. 2003).

(2) **Two-stage optimisation algorithms:** where a satisfaction of the soft constraints is attempted only once a feasible timetable has been found (Arntzen and Løkketangen 2005; Burke et al. 2003a; Casey and Thompson 2002; Chiarandini et al. 2003; Kostuch 2005; Lewis 2006; Lewis and Paechter 2004; Socha and Samples 2003; Socha et al. 2002; Thompson and Dowsland 1998; Yu and Sung 2002).

(3) **Algorithms that allow relaxations:** where violations of the hard constraints are disallowed from the outset by relaxing some other feature of the problem, and attempts are then made to try and satisfy the soft constraints, whilst also giving consideration to the task of eliminating these relaxations (Batenburg and Palenstijn 2003; Burke et al. 2001, 1995b,c, 1996b; Burke and Newall 1999, Cote et al. 2005; Erben 2001; Merlot et al. 2003; Paechter et al. 1998; Petrovic and Bykov 2003; White and Chan 1979).

We will now conduct a survey of the field of metaheuristics and timetabling using these three categories in order to classify the various algorithms. For each case we will first provide a simple and general overview of the method, and will then provide a description of various works that have used this basic approach. Note that although this method of classification might be instructive for our reviewing purposes, it should not be interpreted as a definitive taxonomy, and it is arguable that some of the algorithms that will be mentioned could belong to more than one of the categories.

3.1 One stage optimisation algorithms

In general, one-stage optimisation algorithms allow the violation of both hard and soft constraints within a timetable, and the aim is then to search for a solution that features an adequate satisfaction of both. Ordinarily, this is achieved through the use of some sort of weighted sum function intended for giving violations of the hard constraints much higher penalties than violations of the soft constraints. The following function is typical. Given a problem with k types of constraint, where the penalty weighting associated with each constraint i is w_i, and where $v_i(T)$ represents the number of constraint violations of type i in a timetable T, the quality of T can be calculated using the formula:

$$f(T) = \sum_{i=1}^{k} w_i v_i(T). \quad (1)$$

There are two main advantages of this sort of approach. First, because the aim is to simply search for a candidate solution that minimises a single objective function, it can, of course, be used with any reasonable optimisation technique. Second, this approach is, in general, very flexible and easy to implement, because any sensible constraint can be incorporated into the problem provided that an appropriate penalty weighting (which indicates its relative importance compared to others) is stipulated. In particular, this second factor is highly convenient for timetabling problems where we can often encounter an abundance of different constraint combinations in practice.

It is perhaps for these reasons that there have been a large number of timetabling algorithms of this type proposed in the literature. Possibly, one of the earliest examples was provided by Colorni et al. (1991, 1992, 1997) using an evolutionary algorithm (EA). In this particular approach, a number of hard and soft constraints are considered, and a weighted sum function is used to distinguish between the two types. Additionally, the priorities that are implied by the weightings are also complemented by a genetic repair procedure (that the authors call the filtering procedure), which is used to "fix" a particular timetable if the number of hard constraint violations is seen to rise above a predefined level. The authors also report that improvements are gained when the EA's operators are used in conjunction with a local search procedure that moves an offspring to its local optimum when it is first constructed. In Colorni et al. (1997)

the authors compare this algorithm against a simulated annealing and tabu search approach and report that their EA produces better results (with respect to the minimal objective value found within a pre-defined cut-off point) than simulated annealing but slightly worse results than tabu search. However, they also note the obvious advantage that the EA possesses in that it is able to produce a large number of different, good quality timetables in a single run.

Another early approach using EAs was reported in Corne et al. (1995). In their studies, weights are again used in the fitness function, and three different representations are proposed. The most notable of these is the so-called clash-rich encoding whereby each event is assigned an integer that represents the particular timeslot that it has been assigned to. Thus a chromosome such as 9273453, for example, is interpreted as "event 1 is to be scheduled in timeslot 9", "event 2 is to be scheduled into timeslot 2", "event 3 into timeslot 7", and so on. As the authors note, it is likely that the vast majority of chromosomes in the search space defined by this encoding will probably feature some hard constraint violations—especially due to event-clashes (hence the name "clash-rich"). However, one noted benefit is that because of its simplicity, the authors are able to define various specialised mutation operators (what the authors call "violation-directed" and "event-freeing" mutation operators) in which problematic areas of the chromosome are identified and then mutated in such a way so that constraint violations are hopefully rectified during evolution. Such operators are shown to to improve the performance of the algorithm in general, and using this approach, Corne et al. have reported good results for a number of randomly generated test instances, as well as some real-world problem instances taken from Edinburgh University.

Another one-stage EA for course timetabling has been proposed by Carrasco and Pato (2001). In this work the authors consider a particular problem originating from the Department of Hotel Management and Tourism at the University of the Algarve, Portugal. They also decide that two distinct measures should be used to evaluate a particular solution: the number of soft constraint violations from the students perspective (the class-oriented objective), and the number of soft constraint violations from the point-of-view of the teaching staff (the teacher-oriented objective). As the authors note, in their problem these two measures of quality conflict, and so they decide to apply a multi-objective evolutionary algorithm to the problem, so that upon completion of a run, a user might be provided with a range of trade-off solutions with regards to these two competing objectives. The proposed algorithm operates as follows. First, an initial population is constructed using a constructive heuristic procedure. The resultant candidate solutions (which may or may not also contain violations of the hard constraints) are then evaluated according to both objective functions. Extra penalties are also applied to both of these measures if the candidate solutions feature any hard constraint violations. Next, using their own specialised evolutionary operators [together with a selection pressure determined by the *Non-Dominated Sorting* approach of Srinivas and Deb (1994)], the population is then evolved, and non-dominated solutions that are discovered during this process are copied into an archive population. Upon completion of a run, the final archive contains a collection of timetables that can be both feasible and infeasible. The authors report that this algorithm, which in this case is only tested on one problem instance, is able to produce feasible solutions that are better than the previous manually produced timetables with regards to both of the competing objectives.

Moving away from EAs, various other authors including Abramson (1991); Abramson et al. (1996), Melicio and Caldeira (1998), and Elmohamed et al. (1998) have reported one-stage optimisation algorithms that make use of simulated annealing (Kirkpatrick et al. 1983). In the approach of Elmohamed et al., for example, the authors consider the timetabling problem of Syracuse University in the USA and use a weighted-sum scoring function that penalises hard constraints violations more heavily than soft constraint violations. As well as using general simulated annealing practices to try and optimise these timetables, the authors also point out that good results can be gained if three modifications are made to their system: (1) the introduction of adaptive cooling, (2) the use of a rule-based pre-processor to intelligently build good starting solutions for the annealing phase, and (3) the use of specialised methods for choosing appropriate *neighbourhood moves*—that is, small perturbations of the candidate solution.

Meanwhile, other authors, such as Costa (1994), Hertz (1991), and Schaerf (1996) have chosen to implement one-stage optimisation algorithms using the tabu search metaheuristic. In Schaerf's approach, for example, the author suggests inserting periods of local search in-between phases of tabu search, with the best results of each stage then being passed into the next stage until no further progress can be made through the search space. For this approach the author makes use of two neighbourhood operators, with the first simply moving some events between timeslots. However, it is noted that this operator often causes violations of the hard constraints to occur, and so a second neighbourhood operator is also used which specifically choses moves that have a good chance of canceling out any infeasibilities occurring as a result of the first move. Weights are also used in the evaluation function in order to deal with any other hard constraint violations that occur. The results from this paper (with some experiments being performed on some real data taken from some Italian schools) show that the algorithm is able to produce feasible solutions that are better than the hand-made ones originally used in the schools in reasonable amounts of time. In later work, Di Gaspero and Schaerf (2001) have also made use of similar methods (i.e. using tabu search, together with a weighted evaluation function) for the examination timetabling problem. A comparison with other algorithms proposed for the same problem, however, reveals that this approach is not always able to perform as well in some instances; although the authors do make note of a number of possible improvements that could be made in the future.

Di Gaspero and Schaerf (2002) have also taken a look at the applicability of multi-neighbourhood search to a course timetabling problem. In this work, the authors show how various combinations of different neighbourhood operators can be combined in order to produce effective searches through the search space (in this case only two operators are considered). The combinations suggested are neighbourhood-union, neighbourhood-composition and token-ring search. A macro-perturbation operator (that the authors call a "kick" operator) is also used to help reinvigorate the search from time-to-time in order to help the algorithm avoid getting stuck in local optima. In this work the authors once again make use of weights in the objective function in order to penalise the occurrence of hard constraint violations.

At this point, we have reviewed a number of different algorithms that, in one way or another, have made use of weightings in their objective functions in order to penalise more heavily the occurrence of hard constraint violations within a candidate solution. It is worth noting here, however, that although the one-stage approach has some advantages (noted at the beginning of this subsection), it also has some inherent disadvantages. Firstly, it has been argued by various authors, such as Richardson et al. (1989), that the weighted evaluation scheme does not work well in problems that are sparse (i.e. where only a few solutions exist in the search space). Secondly, although the choice of weights for discriminating between different sort of violations will often critically influence an algorithm's navigation of the search space (and therefore its timing implications and solution quality), there does not actually seem to be any standardised method for choosing them. In practice, what this means is that, often, a rather unsatisfactory amount of ad hoc reasoning will be required from the user when encountering different problem versions (and even different instances of the *same* problem version). Thirdly, the use of weights in an evaluation function will often mean that when applying a small change to a candidate solution (which is what neighbourhood operators typically do), this could result in an overly large change to its cost. Thus, it would seem that weights are problematic as they can magnify the already naturally existing discontinuity in a problem's fitness landscape, making it even *more* difficult for an algorithm to navigate.

With regards to timetabling, however, it is worth noting that some of the algorithms that we have reviewed above have had automated methods built into them in order to try and alleviate some of these problems. For example, in both (Di Gaspero and Schaerf 2001) and (Schaerf 1999a), Schaerf and Di Gaspero have used a method by which the penalties awarded to hard constraint violations can actually be altered dynamically during the search. In Schaerf (1999a), for example, Schaerf uses a weighting value W that is initially set to 20. However, at certain points during the search, the algorithm increases W when it is felt that the search is drifting into search-space regions that are deemed too infeasible. Similarly, W can also be reduced when the search consistently finds itself in feasible regions. It is proposed that such a scheme allows a better quality search than simply keeping W fixed throughout the run. As a second example, the Tatties timetabling system of Paechter et al. (1998) (which we will also look at in Sect. 3.3) also allows weights to be altered during a run. However, this time the adjustment is performed manually by the users themselves by means of a graphical interface that allows him-or-her to specify a target and a weight for the various different problem measures that are considered, and then change these values during a run as they see fit.

3.2 Two-stage optimisation algorithms

The operational characteristics of two-stage optimisation algorithms for timetabling may be summarised as follows: in stage-one, the soft constraints are generally disregarded and only the hard constraints are considered for optimisation (i.e. only a feasible timetable is sought). Next, assuming feasibility has been found, attempts are then made to try and minimise the number of the soft constraint violations, using techniques that only allow feasible areas of the search space to be navigated.

Obviously, an immediate benefit of this technique is that it is no longer necessary to define weightings in order to distinguish between hard and soft constraints, because we no longer need to directly compare feasible and infeasible timetables. In practical situations, such a technique might also be more fitting when achieving timetable feasibility is the primary objective, and where we only wish to make allowances toward the soft constraints if this feasibility is not compromised. Indeed, there are two main reasons why the use of a *one*-stage optimisation algorithm might turn out to be inappropriate in a situation such as this. First, whilst searching for a feasible timetable, a weighted-sum evaluation function will always take the soft constraints into account to some extent, and therefore it might provoke the adverse effect of leading the search away from attractive (i.e. in this case fully feasible) regions of the search space. Second, the operational characteristics of the one-stage optimisation approach also suggest that such an algorithm will often allow a timetable to move away from feasibility in order to eliminate some soft constraint violations. However, in this particular case this strategy might be unwise, because there would, of course, be no guarantee that feasibility could then be re-established at a different point in the search space later on.

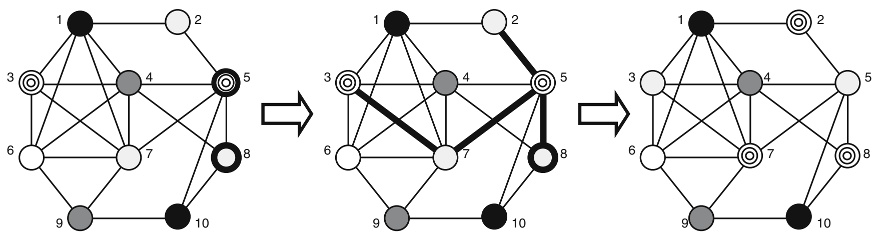
An early example of the two-stage approach was provided by Thompson and Dowsland (1998). In this study, the authors consider a simple exam timetabling problem in which a feasible timetable is deemed one that contains no event-clashes and that obeys all of the room-capacity constraints. However, soft constraints are also considered concerning the desirability of having each students exams "spread out" within the pre-specified exam period in order to decrease student stress and to aid revision. Thompson and Dowsland choose to relate their exam timetabling problem directly to the underlying graph colouring problem and suggest a two-phase approach whereby a feasible colouring/timetable is first obtained, and then various neighbourhood operators that always preserve feasibility (but, at the same time, are still able to alter the other characteristics of solution) are used. Although, as we noted earlier, the graph colouring problem is NP-hard, the authors state that the instances they use (data taken from the exam requirements at the University of Wales, Swansea, as well as some other universities) are loose enough to ensure that feasible timetables can always be found fairly easily using standard graph-colouring heuristics.

A feasible colouring timetable having been found, Thompson and Dowsland attempt to satisfy the soft constraints of their problem using simulated annealing. In their tests, three different neighbourhood operators are proposed, each which is designed to only allow movements from one feasible timetable to another:

(1) **The standard neighbourhood:** whereby the colour of a particular vertex is changed to a new colour (i.e. an event is moved to a new timeslot) in such a way so that feasibility is preserved.

(2) **Kempe chain neighbourhood:** whereby the colours of all the vertices contained in a randomly selected Kempe chain are switched (See Fig. 3).

(3) **S-chain neighbourhood:** which is, in essence, very similar to the Kempe chain neighbourhood, but which involves more colours. Thus $S > 2$ colours are identified, and using an ordered list of these colours together with a recursive procedure to produce the S-chain, we are able to identify various sub-graphs whose colours, when swapped, will result in a different but still feasible solution. (In this case



(1) Select two adjacent nodes at random. (In this case we have selected vertices 5 and 8). Assuming that the graph contains no colour conflicts then these two vertices will be different in colour.

(2) Determine the corresponding Kempe chain of these vertices by identifying all of the connected vertices that are reachable from vertices 5 and 8 and which are assigned either of the two selected colours.

(3) Swapping all of the colours within the Kempe chain will result in a different feasible colouring that uses no more colours than the original.

Fig. 3 Example of a Kempe chain neighbourhood operator

only 3-chain neighbourhoods are considered. For further reference see the work of Morgenstern 1989; Morgenstern and Shapiro 1990.)

The conclusions of Thompson and Dowslands work is that the Kempe chain neighbourhood operator produces significantly better results than the standard neighbourhood due to its increased flexibility. They also note that there is little difference in the results between Kempe chain neighbourhoods and 3-chain neighbourhoods.

A two-stage style approach for exam timetabling has also been proposed by Casey and Thompson in 2002. This particular algorithm is based upon a Greedy Randomised Adaptive Search Procedure (GRASP) and operates as follows: In stage-one, a feasible, clash-free timetable is first gained using a probabilistic-based constructive algorithm that works by taking events one at a time and attempts to place these into feasible timeslots, using backtracking when necessary [in the same fashion as Carter et al. (1996)]. Once feasibility has been achieved, the algorithm then moves onto the second phase and attempts to satisfy the soft constraints whilst always remaining in feasible areas of the search space (this is again achieved using Kempe chain interchanges and simulated annealing). After a period, the algorithm then returns to stage one, using the information gained in the previous iteration to aid the constructive algorithm. This then continues for a set number of iterations. This algorithm is tested on the Carter Instances and the results reported are generally good.

A number of good two-stage algorithms have also been proposed for the course timetabling problem-version that was used for the International Timetabling Competition, which was run in 2003 (http://www.idsia.ch/files/ttcomp2002). The basic idea in this competition was for participants to design algorithms for a simple pre-defined course timetabling problem, which were then compared against each other using a common set of benchmark instances and a fixed time limit. Upon the close of the competition, the participant whose algorithm was deemed to perform best across these instances (and a number of unseen instances only available to the organisers) was then awarded a prize.

An important aspect of this competition—and one which almost certainly influenced the choices of algorithmic methods used by many of the competition entrants—was the way in which the organisers chose to evaluate the timetables. The official rules of the competition stated that timetable quality would only be measured by looking at the number of soft constraint violations. Thus, if a timetable (a) contained any hard constraint violations, (b) used any extra timeslots beyond the pre-defined limit of 45, or (c) left any events unplaced, then it would immediately be considered worthless. Indeed, the organisers imposed a strict rule that stated that participants would only be allowed to submit an entry to the competition if their algorithms could find feasibility *on all twenty instances*. (This also meant that the problem instances were generated so that feasibility was always achievable and relatively easy to find.) Given these factors, it is perhaps unsurprising that many of the entrants to this competition therefore elected to use the two-stage timetabling approach. One successful algorithm was the two-stage tabu search-based approach presented in Arntzen and Løkketangen (2005). In the first stage, their algorithm used a constructive procedure to build an initial feasible timetable. The algorithm then used tabu search in conjunction with simple neighbourhood operators to satisfy the soft constraints. In the latter stage, feasibility was always maintained by making use of neighbourhood operators that were restricted so as never to break any of the hard constraints.

Another approach similar to this was proposed in Burke et al. (2003a). In this paper the authors make use of the Great Deluge algorithm of Dueck (1993) in order to try and reduce the number of soft constraint violations. Again, this is done whilst always remaining in feasible areas of the search space. In essence, Great Deluge is a type of neighbourhood search algorithm that is similar in style to methods such as simulated annealing and tabu search. However, a useful feature of this approach is that only one parameter needs to be defined before running the algorithm: the amount of available search time. Given this value, the algorithm then adapts the intensity of the search so that it only starts to converge on local (or global) optima when this time limit is being approached. As can be expected, the algorithm is reported to return better quality results when the amount of available search time is increased, and the algorithm is reported to return good results on the competition benchmark instances.

Socha, Knowles, and Sampels have also suggested ways of applying the ant colony optimisation metaheuristic to the competition problem-version. In Socha and Samples (2003) and Socha et al. (2002), the authors present two ant-based algorithms—an Ant Colony System and a MAX-MIN system—and provide a comparison between them. At each step in both algorithms, every ant first constructs a complete assignment of events to timeslots using heuristics and pheromone information (the latter which is accumulated in previous iterations of the algorithm). Timetables are then further improved by way of a local search procedure, originally proposed by Rossi-Doria et al. (2002). (A description of the MAX-MIN algorithm—which was actually entered unofficially to the timetabling competition—can also be found at (http://www.idsia.ch/files/ttcomp2002), where good results are reported for the competition benchmark instances.)

Another successful two-stage algorithm for this particular problem-version is offered by Chiarandini et al. (2003). Here, an initial feasible solution is first found using a combination of constructive heuristics, local search, and tabu search. Once this is achieved, the algorithm then moves onto satisfying the soft constraints by first using variable neighbourhood search, and then using simulated annealing. A simple reheating function is also used during this phase which resets the temperature to its initial starting value when it is felt that the run is starting to stagnate. The authors also make use of the *F-Race* method (Birattari et al. 2002) to automatically tune the algorithm's parameters toward the competition instances.

Kostuch (2005) has also used simulated annealing as the main construct in his timetabling algorithm. Based upon his winning entry to the timetabling competition, this method works by first gaining feasibility via simple graph colouring heuristics (plus a series of improvement steps if these heuristics prove inadequate) and then uses simulated annealing to try and satisfy the soft constraints, first by ordering the timeslots, and then by swapping events between timeslots. One of the interesting aspects of Kostuch's approach is the way in which the cooling schedule for the simulated annealing stage is determined: in this case, the algorithm's time limit is used in order to calculate a temperature decrement rule that allows a cooling that is as slow as possible, but which will still spend enough time at low temperatures for sufficient movements toward the optimal to be possible. [This is similar to the Great Deluge approach of Burke et al. (2003a) that we mentioned earlier.]

Having now examined a number of two-stage timetabling algorithms, it should be clear that for this approach to work effectively, it is essential that two criteria are met. First, timetable feasibility must be obtainable in a reasonable amount of time. Second, a practical amount of movement within the resultant feasible-only search space must be attainable. With regards to the latter requirement, if the search space of the problem is quite unconstrained and a reasonable part of this space is made up of feasible solutions, then this may be so. However, it is also possible that if the "constrainedness" of this space is quite high, then searches of this kind might potentially turn out to be extremely inefficient and/or ineffective at times, because it may simply be too difficult for the algorithm to move about and explore the feasible parts search space in any sort of useful way [see the work of Lewis (2006) for empirical evidence of this]; indeed, in these cases, perhaps, a method that allows the search to take "shortcuts" across infeasible parts of the search space might allow a better search to be conducted in some cases.

Additionally, as we mentioned earlier, whether this technique turns out to be appropriate in a practical sense will also depend on the user requirements: if a completely feasible timetable is an absolute necessity and the user is only interested in satisfying the soft constraints if this feasibility is assured (as was the case for the International Timetabling Competition), then the two-stage approach may well be fitting. On the other hand, if, for example, we were to be presented with a problem instance where feasibility was very difficult (or impossible) to achieve, then it could be the case that users might actually prefer to be given a solution timetable in which a suitable compromise between the number of hard and soft constraint violations has been achieved, suggesting that, perhaps one of the other two types of algorithm might be more appropriate instead.

3.3 Algorithms that allow relaxations

Our final category of metaheuristic timetabling algorithm contains those methods in which some aspect of the problem is "relaxed" so that the algorithm can attempt to satisfy the soft constraints, but not at the expense of violating any of the hard constraints. Typically these relaxations are achieved in one of two ways:

1. Events that cannot be feasibly assigned to any place in the current timetable are left to one side unplaced. The algorithm then attempts to satisfy the soft constraints and will hope to insert these unplaced events into the timetable at a later stage.
2. Extra timeslots are opened in order to deal with events that have no existing feasible timeslot available. If necessary, efforts are then made to try and reduce the number of timeslots down to the required amount, whilst also taking into consideration the satisfaction of the soft constraints.

An early example of the second type above was proposed by Burke et al. (1995b) and followed an evolutionary-based approach applied to exam timetabling. In this method each individual in the initial population is first created by feeding a random permutation of the events into a greedy graph colouring-style algorithm. This takes each event in turn and places it into the first timeslot where no clash occurs and where capacity constraints are not exceeded. Extra timeslots are then opened for exams that cannot be placed into existing timeslots. Having constructed the initial population, the algorithm is then concerned with meeting two requirements: (a) lowering the number of timeslots being used to a suitable level; and (b) spreading the exams of each student within the timetable in a similar manner to Thompson and Dowsland (1998) mentioned earlier. In their approach, the authors choose to combine these two objectives into a single numerical function using weights. Knowledge augmented evolutionary operators are also used for producing offspring that contain no infeasibilities. The results from this paper indicate the following trade-off: if, in the fitness function, a large emphasis is placed upon keeping the timetable short, then this will usually be to the detriment of the secondary objective, because there will be less opportunity for events to be spread out. Conversely, if a larger emphasis is placed upon the secondary objective, this will lead to the production of a longer timetable. In Burke et al. (1995c), the authors have continued this work, this time with research being directed toward the design of more specialised recombination operators for the problem. The authors define a basic scheme for recombination, and then experiment with various heuristics used in conjunction with this scheme in order to identify the most effective one. Good results to a real-world timetabling problem taken from Carter's problem instance set are claimed.

A similar strategy for exam timetabling has also been proposed by Erben (2001), who chooses to use a modified version of his Grouping Genetic Algorithm (GGA) for graph colouring (which is presented in the same paper). In this case, an initial population of feasible exam timetables is made in a similar way to Burke et al. above; however, this time the population is evolved using GGA-based operators instead [see the work of Falkenauer (1994, 1998), Lewis (2006), and Lewis and Paechter (2007) for further explanations of these operators]. As with Burke et al., Erben then concerns himself with satisfying two objectives: shortening the timetable (i.e. decreasing the number of timeslots), and spreading the exams to ease student workloads. Once again, these two objectives are combined into one fitness function, with heavier weightings being placed upon the former objective.

It is noticeable that in the exam timetabling problem-version tackled by Burke et al. (1995b,c) and Erben (2001), a compromise will usually have to be struck between (a) keeping the timetable short and (b) spreading-out each of the students events within the timetable. In this sense, this particular formulation might be considered a type of Multi-objective problem. It is for these reasons that Cote et al. (2005) have chosen to make use of multi-objective optimisation techniques for this problem-version. In essence, their approach involves the use of a hybrid multi-objective EA that ranks each member of the population using the concepts of "Pareto Strength" used in the *SPEA-II* multi-objective EA of Zitzler et al. (2001). The algorithm functions by first producing a population of timetables that use various numbers of timeslots between some fixed bounds (and which may or may not be feasible) and then evolves these timetables using two local search procedures intended for eliminating timetable infeasibilities and making improvements on the spreading of the events. The algorithm then stores any non-dominated solutions that are found in an archive population and, upon completion of a run, the user is presented with a number of alternative timetables of different length and differing levels of event spread. In practical applications the use of an archive population might be considered an advantage, because the user will be able to choose from a variety of solutions along the Pareto front, without there being any need for the manual specification of weights beforehand. (Presumably the task of choosing one of these timetables is easier than manually defining the weights in the first place.) Other notable examples of multi-objective techniques applied to timetabling have also been provided by Burke et al. (2001), and Petrovic and Bykov (2003); and in an extended abstract by Paquete and Fonseca (2001). In the first two of these works, the authors consider exam timetabling problems, and, again, the problem instances used in experiments are taken from the Carter instance set. In both cases the authors choose to impose nine different soft constraints on these problems, dealing with factors concerning room capacities, the spreading-out of exams, and the ordering of exams. Neighbourhood search techniques are then used in order to try and optimise these timetables and, in the case of Petrovic and Bykov, weights that are adjusted automatically during the run are used in order to help influence the direction of the search.

Moving away from the field of multi-objective metaheuristics, another evolutionary-based approach to exam timetabling that uses methods slightly different to those discussed thus far has been proposed by Burke et al. (1996b). In this case the number of timeslots that are to be used in the timetable is defined explicitly in advance, and events that cannot be inserted into any timeslot are then deliberately kept to one side unplaced. Consequently, an evaluation function is then used that reflects (a) the number of unplaced events (to which a large weighting is applied) and (b) how spread-out each students exams are in the timetable. During a run of the algorithm the authors then make use of three operators in order to evolve this population: their so-called light and heavy mutation operators, and a deterministic hill-climbing procedure. (No recombination is used.) In particular, the latter operator, as well as attempting to optimise the spreading of exams, also places a special preference on inserting the unplaced exams if this is possible. The authors then test this algorithm using an instance from Nottingham University and the Carter problem instances (http://www.or.ms.unimelb.edu.au/timetabling/atdata/carterea.tar), over a range of timeslot limits. In experiments the algorithm is reported to eventually schedule all of the unplaced exams, as well as making significant improvements in the spread of the exams in the timetable. Further work has also been carried out using this algorithm by Burke and Newall (1999). In this paper, however, the authors choose to investigate the idea of problem decomposition and show ways in which a timetabling problem might be broken up into smaller sub-problems, each of which is then "solved" in sequence. The results indicate that their method of decomposition allows good results to be gained in much less time than when using the same evolutionary-based algorithm on the entire problem in one go. This same decomposition method has also been used with a parallel tabu search-based algorithm by Batenburg and Palenstijn (2003).

Merlot et al. (2003) have also looked at exam timetabling. Here, the authors suggest an approach involving constraint programming, simulated annealing, and then hill climbing. In the first stage the constraint programming part of the algorithm formally combines constraints in order to reduce the search space, and uses this information to try and construct an initial feasible timetable (however, if events cannot be placed, these are placed into a so-called dummy slot at the end of the timetable). Optimisation then follows in the next two stages, where attempts are made to try and improve the timetable by (a) satisfying the soft constraints and (b) moving any events in the dummy slot back into the timetable. An evaluation function that places a large emphasis on emptying the dummy slot (through weights) is also used. In experiments, the algorithm is tested using real-world problem instances from the University of Melbourne, to which it is said to find substantially better results than the universitys existing methods. Various benchmark problem instances such as the Carter instances are also used, the results of which compare well to other methods.

Finally, another evolutionary approach—this time for university course timetabling—has been proposed by Paechter et al. (1998). Here, the authors describe a memetic approach whereby an EA is supplemented with an additional local search routine that aims to improve each timetable when it is being built, with the results then being written back into the chromosome. Like the approach of Burke et al. (1996b) this algorithm then disallows the violation of any hard constraints by leaving certain events to one side unplaced if necessary. Violations of the soft constraints—of which there are a large variety—are then also penalised through the use of weightings that can be adjusted by the user during the search (as we mentioned in Sect. 3.1). This algorithm has been successfully used for module timetabling at Napier University in Scotland: a problem that involves the scheduling of 2,000+ events into 45 timeslots and around 180 rooms. To our knowledge, this is one of the largest practical problems that has been successfully tackled using a metaheuristic.

One further aspect of Paechter et al.'s approach is the use of *sequential evaluation*. During a run, the user is given a choice as to whether he-or-she wants to give special priority to the task of inserting the unplaced events into the timetable. If this option is not taken, then unplaced events are simply treated as an additional soft constraint by the algorithm; however, if it *is* taken, then when two candidate timetables are compared, the algorithm always deems the one with the least number of unplaced events as the fitter. Ties are then broken by looking at the penalties caused by each of the timetable's soft constraint violations. This particular approach means that many of the problems encountered when judging a timetables quality through a single numerical value alone (as is the case with one-stage optimisation algorithms for timetabling—see Sect. 3.1) can be avoided. Note, however, that this method of evaluation is only useful for algorithms where it is sufficient to know the *ordering* of a set of candidate solutions, rather than an explicit numerical quality-score for each (in this case, the authors use binary tournament selection with their EA). Such a method is thus less well suited to other types of optimisation algorithms.

4 Summary and discussion

Concluding this review, it is apparent that timetabling problems can be—and indeed *have* been—addressed using a variety of different computational approaches. In this review we have focused our discussions on the application of metaheuristics to these problems, and have taken special note of the different ways in which these sorts of algorithm might be adapted for dealing with and distinguishing between the hard and soft constraints of a particular problem. Consequently, we have suggested that metaheuristic algorithms for timetabling can be loosely separated into three main classes—one-stage optimisation algorithms, two-stage optimisation algorithms, and algorithms that allow relaxations. It is worth stressing that although each scheme seems to have its advantages and disadvantages, it is not the case that any particular class of algorithm is universally superior to any other. Indeed, it is only likely that certain approaches are more suited to certain *types* of problem-situations and certain *types* of user requirements. It seems reasonable to assume that when choosing a particular approach for one's own timetabling problem, these issues should therefore be given the most consideration, especially bearing in mind that the solutions to practical problems will inevitably have to be used by real people. It goes without saying, however, that it is also desirable for an algorithm to be fast, reliable, and robust whenever possible.

One noticeable trait from our review is the fact that in some cases, the algorithms that have been proposed for a particular timetabling problem may have only been tested on a small number of the authors' own problem instances. From a practical perspective, this situation is understandable, because if a timetabling problem needs to be solved at the authors own institution, they will, of course, be more motivated in solving this problem, rather than spending their time trying to solve other peoples'. However, from a research point-of-view, this characteristic will, of course, often make it very difficult to assess how well an algorithm is capable of performing in comparison to other algorithms. Also, in many cases, other researchers will have no idea whether or not the problem instances used in the study were actually "hard" or not. It is also quite common in these cases for authors to state that their algorithm was able to produce better timetables than the manually-produced solutions previously used by their institution. However, there are actually a number of potential pitfalls in using an argument such as this to justify an algorithm. First, this sort of claim involves making the explicit assumption that the institution's criteria of measuring timetable "goodness" (whatever these might be) have all been effectively captured by the algorithm's objective function. However, it could be the case that, often, the people who construct the timetables by hand may well make certain choices subconsciously and, consequently, may not be able to describe these processes and criteria to the algorithm designer. Secondly—and perhaps more obviously—the statement that an algorithm is better than a human-designed timetable is, of course, only actually meaningful when the reader is also given some indication of the skill-level of the human in question. This information is not normally provided, however.

On the other hand, whilst the use of publicly available timetabling instances (such as Carter's exam problems and also the International Timetabling Competition instance set) can be helpful for comparing and contrasting different timetabling algorithms, in doing so there will always be a slight risk of researchers focusing too much of their efforts on these particular problem instances and "over-fitting" their algorithms. For example, it is arguable that the problem-version and instances used for the timetabling competition has tended to direct authors' design choices towards two-stage optimisation algorithms. However, as has been shown by Lewis (2006), when considering highly constrained problem instances, movements in the feasible-only search space (which is the space that will be explored during the second phase of the two-stage process) will often be too restricted to allow significant improvements to be gained. It is likely, therefore, that for different problem versions—or, indeed, simply different instances of the *same* problem version—that other types of algorithm might prove more fruitful in some cases.

More generally, there also seems to be come confusion in the field about how different timetabling algorithms should be formally tested and evaluated against one another. As we have seen in this review, different approaches and problem formulations will often seek to achieve different sets of objectives, making it difficult to perform meaningful comparisons in many cases. However, difficulties in comparisons also exist in many of the studies where existing benchmark instances have been used as, often, different performance measures will be looked at, different stopping criteria will be imposed, and/or different numbers of trials will be performed in the testing and analyses of the algorithms.

Concerning these latter points, in 2007–2008 the author and colleagues will be organising a second International Timetabling Competition. The intention of this is to extend the knowledge gained from the first competition, and also to facilitate, as far as possible, a formal comparison of submitted algorithms using pre-defined evaluation criteria and a new set of benchmark problem instances. Entry to the competition is free, and we encourage researchers with an interest in timetabling to download our problem instances and submit their best algorithms. The competition website can be found at http://www.cs.qub.ac.uk/eventmap.

References

Abramson D (1991) Constructing school timetables using simulated annealing: sequential and parallel algorithms. Manag Sci 37(1):98–113

Abramson D, Krishnamoorthy H, Dang H (1996) Simulated annealing cooling schedules for the school timetabling problem. Asia Pacific J Operational Res 16:1–22

Arntzen H, Løkketangen A (2005) A tabu search heuristic for a university timetabling problem. In: Ikabaki T, Nonobe K, Yagiura M (eds) Metaheuristics: progress as real problem solvers, vol 32. Springer, Berlin pp 65–86

Balakrishnan N (1991) Examination scheduling: a computerized application. Omega 19(1):37–41

Batenburg KJ, Palenstijn WJ (2003) A new exam timetabling algorithm. In: Proceedings of the Belgian-Dutch artificial intelligence conference (BNAIC 2003), pp 19–26

Birattari M, Sttzle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Langdon WB, Mathias K, Roy R, Davis D, Poli R, Balakrishnan K, Honavar V, Rudolph G, Wegener J, Bull L, Potter MA, Schultz AC, Miller JF, Burke E, Jonoska N (eds) The genetic and evolutionary computation conference (GECCO) 2002. Morgan Kaufmann Publishers Inc, New York, pp 11–18

Boizumault P, Delon Y, Peridy L (1996) Logic programming for examination timetabling. Logic Program 26:217–233

Burke E, Newall JP (1999) A multi-stage evolutionary algorithm for the timetable problem. IEEE Trans Evol Comput 3(1):63–74

Burke E, Petrovic M (2002) Recent research directions in automated timetabling. Eur J Oper Res 140(2):266–280

Burke E, Elliman D, Weare R (1995a) The automation of the timetabling process in higher education. J Educ Technol Syst 23:257–266

Burke E, Elliman D, Weare R (1995b) A hybrid genetic algorithm for highly constrained timetabling problems. In: Eshelman L (ed) Genetic algorithms: proceedings of the sixth international conference (ICGA95). Morgan Kaufmann, pp 605–610

Burke E, Elliman D, Weare R (1995c) Specialised recombinative operators for timetabling problems. In: The artificial intelligence and simulated behaviour workshop on evolutionary computing, vol 993. Springer, Berlin, pp 75–85

Burke E, Elliman DG, Ford PH, Weare R (1996a) Examination timetabling in british universities: a survey. In: Burke E, Ross P (eds) Practice and theory of automated timetabling (PATAT) I, vol 1153. Springer, Berlin, pp 76–92

Burke E, Newall JP, Weare RF (1996b) A memetic algorithm for university exam timetabling. In: Burke E, Ross P (eds) Practice and theory of automated timetabling (PATAT) I, vol 1153. Springer, Berlin pp 241–250

Burke E, Bykov Y, Petrovic M (2001) A multicriteria approach to examination timetabling. In: Burke E, Erben E (eds) practice and theory of automated timetabling (PATAT) III, vol 2070. Springer, Berlin, pp 118–131

Burke E, Bykov Y, Newall JP, Petrovic S (2003a) A time-defined approach to course timetabling. Yugoslav J Oper Res (YUJOR) 13(2):139–151

Burke E, Kendall G, Soubeiga E (2003b) A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 9(6):451–470

Burke E, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyper heuristic for timetabling problems. Eur J Oper Res 176

Carrasco M, Pato M (2001) A multiobjective genetic algorithm for the class/teacher timetabling problem. In: Burke E, Erben W (eds) Practice and theory of automated timetabling (PATAT) III, vol 2079. Springer, Berlin, pp 3–17

Carrasco M, Pato M (2004) Metaheuristics: computer decision-making, chapter A Potts neural network heuristic for the class/teacher timetabling problem. Kluwer, Norwell, pp 173–186

Carter M (1986a) A langarian relaxation approach to the classroom assignment problem. INFOR 27(2):230–246

Carter M (1986b) A survey of practical applications of examination timetabling algorithms. Oper Res 34(2):193–202

Carter M, Laporte G (1998) Recent developments in practical course timetabling. In: Burke E, Carter M (eds) Practice and theory of automated timetabling (PATAT) II, vol 1408. Springer, Berlin, pp 3–19

Carter M, Laporte G, Lee SY (1996) Examination timetabling: algorithmic strategies and applications. J Oper Res Soc 47:373–383

Casey S, Thompson J (2002) Grasping the examination scheduling problem. In: Burke E, De Causmaecker P (eds) Practice and theory of automated timetabling (PATAT) IV, vol 2740. Springer, Berlin, pp 233–244

Chiarandini M, Socha K, Birattari M, Rossi-Doria O (2003) An effective hybrid approach for the university course timetabling problem. Technical Report AIDA-2003-05, FG Intellektik, FB Informatik, TU Darmstadt, Germany

Colorni A, Dorigo M, Maniezzo V (1991) Genetic algorithms and highly constrained problems: the time-table case. In: Schwefel H-P, Manner R (eds) Parallel problem solving from nature (PPSN) I, vol 496. Springer, Berlin, pp 55–59

Colorni A, Dorigo M, Maniezzo V (1992) A genetic algorithm to solve the timetable problem. Technical Report 90-060 revised, Politecnico di Milano, Italy 1992

Colorni A, Dorigo M, Maniezzo V (1997) Metaheuristics for high-school timetabling. Comput Optim Appl 9(3):277–298

Cooper T, Kingston J (1996) The complexity of timetable construction problems. In: Burke E, Ross P (eds) Practice and theory of automated timetabling (PATAT) I, vol 1153. Springer, Berlin, pp 283–295

Corne D, Ross P, Fang H (1995) Evolving timetables. In: Lance C. Chambers (ed) The practical handbook of genetic algorithms, vol 1. CRC, Florida, pp 219–276

Costa D (1994) A tabu search algorithm for computing an operational timetable. Eur J Oper Res 79:98–110

Cote P, Wong T, Sabourin R (2005) Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. In: Burke E, Trick M (eds) Practice and theory of automated timetabling (PATAT) V, vol 3616, Springer, Berlin, pp 294–312

Daskalaki S, Birbas T, Housos E (2004) An integer programming formulation for a case study in university timetabling. Eur J Oper Res 153:117–135

De Werra D (1985) An introduction to timetabling. Eur J Oper Res 19(2):151–162

Deris B, Omatu S, Ohta H, Samat D (1997) University timetabling by constraint-based reasoning: a case study. J Oper Res Soc 48(12):1178–1190

Di Gaspero L, Schaerf A (2001) Tabu search techniques for examination timetabling. In: Burke E, Erben E (eds) Practice and theory of automated timetabling (PATAT) III, vol 2079. Springer, Berlin, pp 104–117

Di Gaspero L, Schaerf A (2002) Multi-neighbourhood local search with application to course timetabling. In: Burke E, De Causmaecker P (eds) Practice and theory of automated timetabling (PATAT) IV, vol 2740, Springer, Berlin, pp 263–287

Dueck G (1993) New optimization heuristics: the great deluge algorithm and the record-to-record travel. J Comput Phys 104:86–92

Elmohamed S, Fox G, Coddington P (1998) A comparison of annealing techniques for academic course scheduling. In: Burke E, Carter M (eds) Practice and theory of automated timetabling (PATAT) II, vol 1408. Springer, Berlin, pp 146–166

Erben E (2001) A grouping genetic algorithm for graph colouring and exam timetabling. In: Burke E, Erben W (eds), Practice and theory of automated timetabling (PATAT) III, vol 2079. Springer, Berlin, pp 132–158

Even S, Itai A, Shamir A (1976) On the complexity of timetable and multicommodity flow problems. SIAM J Comput 5(4):691–703

Falkenauer E (1994) A new representation and operators for genetic algorithms applied to grouping problems. Evol Comput 2(2):123–144

Falkenauer E (1998) Genetic algorithms and grouping problems. Wiley, London

Garey MR, Johnson DS (1979) Computers and intractability—a guide to NP-completeness, 1st edn. W. H. Freeman and Company, San Francisco

Garey MR, Johnson DS, Stockmeyer L (1976) Some simplified np-complete graph problems. Theor Comput Sci 1:237–267

Gislen L, Peterson C, Soderberg B (1989) Teachers and classes with neural networks. Int J Neural Syst 1:167–176

Goltz H-J, Matzke D (1999) University timetabling using constraint logic programming. In: Practical aspects of declarative languages, vol 1511 of Lecture notes in computer science, Springer, Heidelberg, pp 320–334

Guéret C, Jussien N, Boizumault P, Prins C (1996) Building university timetables using constraint logic programming. In: Practice and theory of automated timetabling I (PATAT I), vol 1153 of Lecture notes in computer science. Springer, Heidelberg, pp 130–145

Hertz A (1991) Tabu search for large scale timetabling problems. Eur J Oper Res 54(1):39–47

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

Kostuch P (2005) The university course timetabling problem with a 3-phase approach. In: Burke E, Trick M (eds) Practice and theory of automated timetabling (PATAT) V, vol 3616. Springer, Berlin, pp 109–125

Kovacic M (1993) Timetable construction using markovian neural network. Eur J Oper Res 69:92–96

Lajos G (1996) Complete university modular timetabling using constraint logic programming. In: Burke E, Ross P (eds) Practice and theory of automated timetabling (PATAT) I, vol 1153. Springer, Berlin, pp 146–161

Lewis R (2006) Metaheuristics for University course timetabling. PhD thesis, School of Computing, Napier University, Edinburgh, October

Lewis R, Paechter B (2004) New crossover operators for timetabling with evolutionary algorithms. In: Lofti A (ed) The fifth international conference on recent advances in soft computing RASC2004. Nottingham, England, pp 189–194

Lewis R, Paechter B (2007) Finding feasible timetables using group based operators. IEEE Trans Evol Comput 11(3):397–413

Melicio F, Caldeira J (1998) Timetabling implementation aspects by simulated annealing. In: Jifa Gu (ed), IEEE systems science and systems engineering. Beijing. Aceite, pp 553–557

Merlot L, Boland N, Hughes B, Stuckey P (2003) A hybrid algorithm for the examination timetabling problem. In: Burke E, De. Causmaeker P (eds) The practice and theory of automated timetabling (PATAT) IV, vol 2740. Springer, Berlin, pp 207–231

Morgenstern C (1989) Algorithms for general graph coloring. PhD thesis, University of New Mexico

Morgenstern C, Shapiro H (1990) Coloration neighborhood structures for general graph coloring. In: Proceedings of the first annual ACM-SIAM symposium on discrete algorithms. San Francisco, California, USA, Society for Industrial and Applied Mathematics, pp 226–235

Paechter B, Rankin R, Cumming A, Fogarty T (1998) Timetabling the classes of an entire university with an evolutionary algorithm. In: Baeck T, Eiben A, Schoenauer M, Schwefel H (eds) Parallel problem solving from nature (PPSN) V, vol 1498. Springer, Berlin, pp 865–874

Paquete L, Fonseca C (2001) A study of examination timetabling with multiobjective evolutionary algorithms. In: 4th metaheuristics international conference (MIC 2001). Porto, pp 149–154

Petrovic S, Bykov Y (2003) A multiobjective optimisation approach for exam timetabling based on trajectories. In: Burke E, De Causmaecker P (eds) The paractice and theory of automated timetabling (PATAT) IV, vol 2740. Springer, Berlin, pp 181–194

Richardson JT, Palmer MR, Liepins G, Hilliard M (1989) Some guidelines for genetic algorithms with penalty functions. In: Schaffer JD (ed) The third international conference on genetic algorithms. Morgan Kaufmann Publishers Inc, San Francisco, pp 191–197

Ross P, Hart E, Corne D (2003) Genetic algorithms and timetabling. In: Ghosh A, Tsutsui K (eds) Advances in evolutionary computing: theory and applications. Springer, New York, pp 755–771

Rossi-Doria O, Knowles J, Sampels M, Socha K, Paechter B (2002) A local search for the timetabling problem. In: Burke E, Causmaecker P (eds) Practice and theory of automated timetabling (PATAT) IV. Gent, Belgium, pp 124–127

Schaerf A (1996) Tabu search techniques for large high-school timetabling problems. In: Proceedings of the thirteenth national conference on artificial intelligence. AAAI Press/MIT Press, Portland (OR), pp 363–368

Schaerf A (1999) Local search techniques for large high-school timetabling problems. IEEE Trans Syst Man Cyber Part A 29(4):368–377

Schaerf A (1999) A survey of automated timetabling. Artif Intell Rev 13(2):87–127

Schimmelpfeng K, Helber S (2007) Application of a real-world university-course timetabling model solved by integer programming. OR Spect 29(4)

Smith K, Abramson D, Duke D (2003) Hopfield neural networks for timetabling: formulations, methods, and comparative results. Comput Ind Eng 44(2):283–305

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

Socha K, Knowles J, Samples M (2002) A max-min ant system for the university course timetabling problem. In: Dorigo M, Di Caro G, Samples M (eds) Proceedings of ants 2002—third international workshop on ant algorithms (Ants'2002), vol 2463. Springer, Berlin, pp 1–13

Srinivas N, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2:221–248

Thompson JM, Dowsland KA (1998) A robust simulated annealing based examination timetabling system. Comput Oper Res 25(7/8):637–648

Tripathy A (1984) School timetabling—a case in large binary linear integer programming. Manag Sci 30:1473–1489

White G, Chan W (1979) Towards the construction of optimal examination schedules. INFOR 17:219–229

Yu E, Sung K-S (2002) A genetic algorithm for a university weekly courses timetabling problem. Int Trans Oper Res 9:703–717

Zitzler E, Laumanns M, Thiele L (2001) Spea2: improving the strength pareto evolutionary algorithm for multiobjective optimization. Technical Report 103, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland