Noname manuscript No.
(will be inserted by the editor)

Timetable Construction: The Algorithms and Complexity Perspective

Jeffrey H. Kingston

the date of receipt and acceptance should be inserted later

Abstract This paper advocates approaching timetable construction from the algorithms and complexity perspective, in which analysis of the specific problem under study is used to find efficient algorithms for some of its aspects, or to relate it to other problems. Examples are given of problem analyses leading to relaxations, phased approaches, very large-scale neighbourhood searches, bipartite matchings, ejection chains, and connections with standard NP-complete problems. Although a thorough treatment is not possible in a paper of this length, it is hoped that the examples will encourage timetabling researchers to explore further with a view to utilizing some of the techniques in their own work.

Keywords Timetabling · Algorithms · NP-completeness

1 Introduction

When tackling a problem, a researcher utilizes a certain perspective, or set of techniques, which he or she understands and has experience with. There is a tendency to stay within one's own perspective, which, though natural, may not be the most scientific thing to do.

One of the strengths of the PATAT conference series is that its contributors bring a variety of perspectives to the timetabling problems they study, thereby exposing its participants to healthy doses of unfamiliar techniques. Judging by the contributions to the most recent conference (Burke and Gendreau 2008), the field is strongly dominated by local search, especially simulated annealing (Dowsland 1993; Kirkpatrick et al. 1983) and tabu search (Glover and Laguna 1998); but there are also papers written from the integer programming perspective, and indeed their number is growing as integer programming packages improve. Other perspectives are also represented, although on a smaller scale: constraint programming, machine learning, and cooperating agents are three examples.

Some of the most interesting papers apply techniques from one perspective to problems that had previously been studied only from another. For example, the Travelling Tournament

Jeffrey H. Kingston
School of Information Technologies
The University of Sydney, NSW 2006, Australia
`http://www.it.usyd.edu.au/~jeff`
E-mail: jeff@it.usyd.edu.au

Problem (Easton et al. 2003) was formulated by researchers associated with the integer programming perspective, but good solutions were later obtained with local search (Ribeiro and Urrutia 2004). And, in the reverse direction, two simplified university course timetabling data sets, compiled by researchers who typically use local search, were recently solved to optimality using an integer programming formulation (Burke et al. 2008).

This paper advocates the *algorithms and complexity* perspective on timetable construction. In general terms, a researcher who utilizes the algorithms and complexity perspective will devote considerable time to analysing the specific problem under study. The outcome of this *problem analysis* might be the discovery that some aspect of the problem is amenable to efficient solution, leading to the design of an algorithm which exploits that fact. In other cases, the outcome might be the discovery of a close connection with another NP-complete problem, which can be helpful in pointing to a body of relevant prior work.

Although problem analysis is practised by all researchers, it is less emphasised in some other perspectives. Researchers who use local search, for example, typically devote most of their time to empirical work and parameter tuning. Indeed, the idea of basing a solution approach on detailed properties of the problem under study may even be deprecated, as tying the algorithm too closely to specific conditions that could change.

It is emphatically not the aim of this paper to show that algorithms and complexity techniques will always be superior, or even that they will always yield anything useful. They are too dependent on specific properties of particular problems for that. Nor does it aim to depreciate the advantages of other perspectives. A great strength of metaheuristics is the ease with which they can be applied to a wide range of problems. The integer programming perspective has the major advantage of emphasising lower as well as upper bounds: either an optimal solution is found, or a lower bound is produced which gives some indication of how close to optimality the solution lies.

What this paper does do is offer examples where algorithms and complexity techniques prove to be useful, sufficient, I hope, to show that they are worth adding to the mental toolkit of the timetabling researcher. The topics covered are relaxation, the phased approach, very large-scale neighbourhood search, bipartite matching, ejection chains, and NP-completeness analysis. Although this paper includes a few original constructions and experiments, it is offered more as a tutorial than as a research contribution. A thorough treatment is not possible in a paper of this length, but it is hoped that the examples will encourage researchers to explore further with a view to utilizing some of the techniques in their own work.

2 Relaxation

One way to find problems of low complexity within an NP-complete problem is to *relax* the NP-complete problem: loosen some of its constraints, or discard them altogether. Although the solution of the resulting *relaxed problem* is not usually a solution of the original problem, it may contain useful information. In particular, the cost of an optimal solution of the relaxed problem is a lower bound on the cost of any solution of the original problem.

Finding relaxations requires problem analysis. On the one hand, the relaxed problem must be sufficiently close to the original for its solution to be relevant; on the other, it must be efficiently solvable, since otherwise nothing is gained by using it.

Relaxation is an important part of the integer programming perspective. The archetypal relaxation replaces the *integrality* constraints of an integer program (which specify that each variable x_i must be assigned an integer in some range $a_i...b_i$) with *linear* constraints (which specify that each variable x_i must be assigned some value in the range $a_i \leq x_i \leq b_i$, with

fractional values allowed), replacing an integer program, which in general describes an NP-complete problem, by a linear program which can be solved in polynomial time (Martin 2007). Another well-known technique is *Lagrangean relaxation* (Beasley 1993), in which different versions of the relaxed problem are solved repeatedly.

Relaxation is useful for determining whether a problem is likely to have a feasible solution. For example, the nurse rostering problem has many complex constraints on the layout of each nurse's shifts. Discarding them leaves a much simpler problem, solvable by bipartite matching (Sect. 5), which checks whether there are enough nurses of the right kinds to cover the required work. There is little point in starting a long solution process if not.

3 The phased approach

The *phased approach* divides the problem into parts, called *phases*, and solves them one by one. Each phase has only limited information about the other phases, so there is little hope of the overall solution being optimal.

Problem analysis is needed to find a decomposition into phases which are efficiently solvable separately, and independent enough to satisfy concerns about the loss of optimality.

Room assignment is often a good candidate for making into a separate phase. When all events have duration 1, this can be done without loss of optimality. During time assignment, one merely tests, using bipartite matching (Sect. 5), whether the events assigned to each time can be assigned rooms, without committing specific rooms to specific events. The actual assignments are made separately at the end.

Large problems, such as whole-university course timetabling and student sectioning problems, are typically solved in phases (Carter 2000; Murray et al. 2007). Their size, and the complexity of the whole process, can make phasing a practical necessity in such cases.

A key module in a student sectioning system is a branch-and-bound algorithm for finding the best possible timetable for one student, holding the rest of the timetable fixed. One run of such an algorithm constitutes one phase.

A few scattered examples exist where the results of a later phase are fed back to a subsequent run of an earlier phase. For example, in unpublished work by staff of the University of Sydney more than a decade ago, after timetabling each student as just described, a second pass over the student list was made, and each student was removed and re-timetabled. The result was a much improved distribution of students into sections. This idea qualifies as an example of very large-scale neighbourhood search, to be described next.

4 Very large-scale neighbourhood search

Very large-scale neighbourhood (VLSN) search (Ahuja et al. 2002) is a form of local search. To move from one solution to its neighbour, a large piece of the solution is deassigned, then reassigned in a different and hopefully improved way.

Although the reassignment can be carried out by any simple constructive heuristic, the method is particularly interesting when problem analysis identifies a piece to deassign whose reassignment may be carried out optimally.

Several examples of the application of VLSN search to timetabling problems are given in Meyers and Orlin (2007). The point is made there that in some cases where other local search methods proceed by swaps, a more general and potentially more effective VLSN search based on the 'cyclic exchange neighbourhood' is possible. This author has used the

	Avail	W1	W2	R1	R2	M8	F5	M3	M4	W5	W6	R8	F2	T3	T4	R5	R6	W8	F3
Art01	0	12-3A-P	*7CKO.*	12-3A-Photograph						7AS2-1			History	8CKO2-1		8CKO4-2			
Art02	4	*7CKO.*	11-3A/1			11-3A/12-3A-Cera				10-4-Art				12-1-VisualArts					
Art03	0	7CKO1-1								7AS1-1				8CKO1-1		8CKO3-2			
Unassigned																			

	M1	M2	T5	T6	W7	F4	M5	M6	T1	T2	R7	F1	W3	W4	R3	R4	M7	T7	F6	F7	F8	T8
Art01	11-4-VisualArt						12-2-Photography-2U						11-6-Photography						Sport			StaffMe
Art02	8AS3-3		7AS3-3		9-4Art-1				9-4Art-1				7CKO4-2		10-4-Art			10-4-Art		8AS2-1		
Art03													7CKO3-2							8AS1-1		
Unassigned			*12-4B-VisualDe*			*12-4B-V*																

Fig. 1 Part of a high school timetable, showing the assignments of the school's three Art teachers. Each teacher occupies one row, with a fourth row holding one Art class which failed to be assigned. Each column holds one of the 40 times of the cycle, except the column adjacent to the teachers' names, which shows the remaining available workload of the teachers.

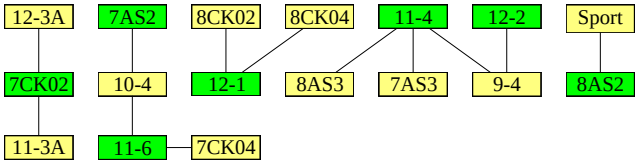

Fig. 2 The clash graph for the meetings assigned to teachers *Art01* and *Art02* in Fig. 1, slightly simplified, and showing one 2-colouring.

cyclic exchange neighbourhood in high school teacher assignment, to permute the assignments at a given set of times among the teachers. It was effective in removing some kinds of defects (Kingston 2008). For variety, this paper offers a different example, again from teacher assignment.

The literature contains a smattering of timetabling papers which improve their resource assignments by deassigning all the work assigned to two resources and then reassigning that work to those same two resources. This clearly qualifies as VLSN search, and when the resources are high school teachers, the reassignment can be done to optimality in practice.

Take the example of teachers *Art01* and *Art02* in Fig. 1. These make good candidates for deassignment and reassignment, because they are qualified to teach similar kinds of classes, and they share a split assignment (occupying times $W1$ and $W2$) whose removal is desirable. A split assignment is a case where one teacher teaches one subject to one class at some times, and another teaches that subject to that class at other times.

Deassign the meetings assigned to the two teachers and build their clash graph, in which each deassigned meeting is a node and two nodes are joined by an edge whenever their meetings share at least one time (Fig. 2). A clash-free reassignment is a 2-colouring of this graph. Each of the graph's K connected components can be coloured independently, and has two distinct 2-colourings; in various special cases there are fewer, making at most 2^K distinct colourings altogether. In practice, K is small enough to permit an exhaustive search for a colouring that does not exceed the teachers' workload limits. For safety, the author's implementation imposes a fixed upper limit on the number of colourings tried.

On real instances this method runs quickly, but its results are disappointing, averaging only about one improvement per instance. Still, it might be useful in other resource assignment problems, such as nurse rostering, where similarly qualified resources work together.

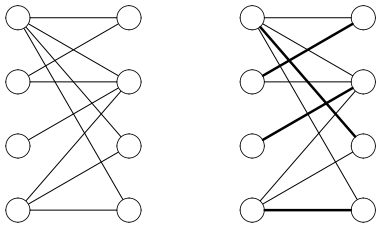

Fig. 3 A bipartite graph (left), and the same graph with a maximum matching, shown in bold (right).

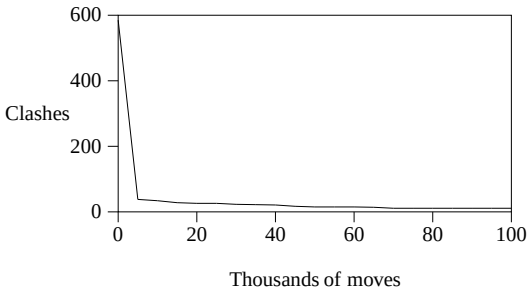

Fig. 4 Performance of tabu search on a bipartite matching problem with 3686 demand nodes and 5378 supply nodes, taken from a real instance of a high school timetabling problem, whose optimal solution is known (by applying the standard polynomial-time algorithm) to have 5 unmatched demand nodes, or equivalently (if every demand node is assigned) 5 clashes. After 100000 moves, starting from an initial greedy solution with 585 clashes, the best solution found had 11 clashes. The neighbourhood was all single moves of a demand node's assignment from one supply node in its domain to another. A move was tabu if it involved a demand node that had been moved recently; the tabu list length was 1500 demand nodes. About ten values for tabu list length were tried; results improved as the length was increased to 1500, and then worsened.

5 Bipartite matching

A *bipartite graph* is an undirected graph whose nodes may be divided into two sets, such that every edge connects a node of one set to a node of the other. A *matching* in an undirected graph (bipartite or otherwise) is a subset of the edges such that no two edges touch the same node. A *maximum matching* is a matching containing as many edges as possible (Fig. 3). The *bipartite matching problem* is the problem of finding a maximum matching in a bipartite graph. There is a standard polynomial-time algorithm for this problem, used in timetabling for more than forty years (Csima and Gotlieb 1964; Gotlieb 1962; de Werra 1971).

In timetabling, it is usual for one of the two sets of nodes to represent variables (events, meetings, etc.) demanding something to be assigned to them, while the other set represents entities (times, resources, etc.) which are available to supply these demands. Accordingly, these two sets will be referred to as the *demand nodes* and the *supply nodes*. A maximum matching assigns one distinct supply node to as many demand nodes as possible.

One application of bipartite matching to timetabling is to assign rooms to meetings after the meetings' times are fixed. At each time, build a bipartite graph with one demand node for each demand for a room at that time, and one supply node for each room available at that time, and connect each demand node to those supply nodes representing rooms which are qualified to satisfy the demand (rooms which are large enough and contain the appropriate facilities). Then a maximum matching gives an optimal assignment of rooms at that time.

Fig. 4 documents a case where a standard local search method (tabu search) could not find an optimal solution to a large, real instance of the bipartite matching problem, even

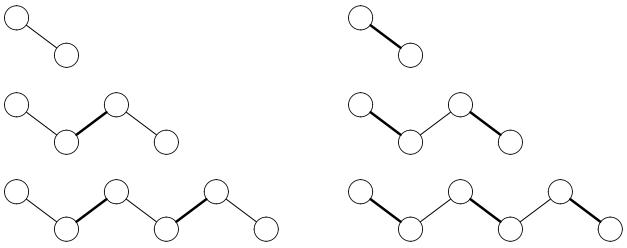

Fig. 5 Augmenting paths (at left) and the effect of applying them (at right). The first augmenting path carries out a simple assignment; the second carries out two assignments and one deassignment; and so on. In each case the size of the matching increases by one. No matching edge may initially touch the first or last node.

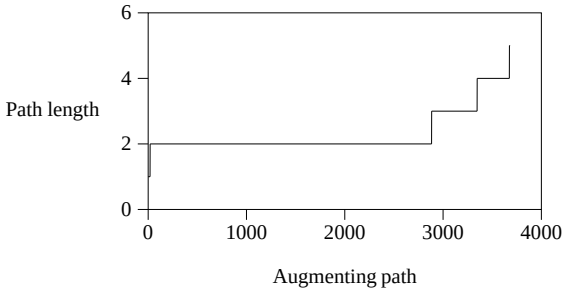

Fig. 6 This graph shows how the length of augmenting paths increases as the bipartite matching algorithm proceeds. The bipartite graph from Fig. 4 was used. For each value of k from $k = 1$ to $3686 - 5$, the length of the longest of the first k augmenting paths found is shown, defining length to be the number of demand nodes on the path. At first, the paths are short (at most 2 demand nodes), but by the end of the algorithm they have length 4 or 5. Breadth-first search was used to find these paths, so this shows that some steps near the end require *at least* 4 or 5 coordinated reassignments of demand nodes in order to reduce solution cost.

when several settings of its parameters were tried and ample time was allowed. Thus, when instances of bipartite matching problems lie within timetabling problems, it may be advantageous to solve them directly using the polynomial-time algorithm, as was done, for example, by the winning entry in the First International Timetabling Competition (Kostuch 2005).

The performance of tabu search on this problem raises a question: if the problem is difficult, then how can the standard algorithm solve it to optimality so quickly? Can that algorithm be applied to other problems, perhaps to NP-complete problems? To answer these questions it is necessary to examine the standard algorithm in detail.

The algorithm is the *augmenting path method*. Starting at each unmatched demand node in turn, it searches the graph for a path from that node to a supply node, from there back to the demand node currently assigned that supply node, from there to a different supply node, and so on, ending at a currently unmatched supply node. Then making each non-matching edge on the path into a matching edge, and each matching edge on the path into a non-matching edge, increases the size of the matching by one (Fig. 5). A theorem guarantees that each node has to be searched through only once, so the cost of finding an augmenting path is bounded above by the total size of the graph. Another theorem guarantees that after each unmatched demand node has been taken as the starting point, the matching is maximum. These results are proved in many text books, for example Papadimitriou and Steiglitz (1982).

The secret of the success of this algorithm is revealed in Fig. 6. At first, it finds very short augmenting paths, such as any local search algorithm could easily find. But towards the end, the paths become longer, until, on large examples such as the one used in the figure,

at least 4 or 5 coordinated reassignments of demand nodes to supply nodes are required to improve the solution. This is difficult for local search algorithms based on simple moves and swaps: they become trapped in what seem to them to be large, featureless plateaus.

To dispel any idea that this algorithm is difficult to implement, here is the key procedure, for searching for an augmenting path out of a given demand node, and applying it if found:

```
bool Augment(DEMAND_NODE demand_node, int visit_num)
{
    int i;  SUPPLY_NODE supply_node;
    for( i = 0;  i < demand_node->domain_size;  i++ )
    {
        supply_node = demand_node->domain[i];
        if( supply_node->visit_num < visit_num )
        {
            supply_node->visit_num = visit_num;
            if( supply_node->supply_asst == NULL ||
                Augment(supply_node->supply_asst, visit_num) )
            {
                supply_node->supply_asst = demand_node;
                demand_node->demand_asst = supply_node;
                return true;
            }
        }
    }
    return false;
}
```

Additional code is needed for initialization and trying each demand node in turn.

6 Ejection chains

It is not hard to see how to apply the augmenting path method of the previous section to assignment-type problems other than bipartite matching. Start at any point where an assignment is required but is currently missing. Mark all elements of the instance unvisited. Try to assign a valid value at the starting point. If that can be done directly, do it; otherwise, find all ways in which a valid value can be assigned, at the cost of one deassignment at some other point. For each of these ways, make the indicated deassignment and assignment, mark the elements involved as visited to ensure that they will not be touched again during the current search, and continue trying to reassign the deassigned element, using the same method recursively. If the search ever reaches an element that can be assigned directly, it does so and terminates, having completed a chain of assignments and deassignments which amount to an augmenting path. Repeat until no further progress occurs.

This idea was given the name *ejection chains* by Glover (the inventor of tabu search), who applied it successfully to the travelling salesman problem (Glover 1996). Similar ideas had probably been used earlier. For example, the widely used Kempe chain method from graph colouring, dating from the work of A. B. Kempe in 1879, could be described as an ejection chain method, although it differs in detail from the method presented here. An accessible account appears in Dowsland (1993).

In general, theorems which guarantee effectiveness (as in the bipartite matching case) will not be available; nevertheless, ejection chains preserve the other virtue of the augmenting path method, namely its ability to explore large plateaus.

A key restriction of the ejection chain method as formulated here is that only one deassignment is permitted for each assignment, ensuring that the structures searched are limited to paths. Although more complex augmenting structures, such as trees, could be permitted, they complicate the implementation and seem less likely to pay off than paths. See Müller et al. (2005) for a simple method which can explore such structures.

This author has used ejection chains to improve the assignment of teachers to meetings in high school timetabling problems, after the meetings' times are assigned. Each meeting may contain several time blocks spread through the week, and may request a teacher of a particular kind. Vertex colouring may be embedded in this problem, making it NP-complete.

First, an initial assignment is made by taking each teacher in turn and applying a branch-and-bound tree search (with a fixed upper limit on the number of nodes searched, for safety) to pack as much workload as possible into the teacher, assigning only meetings for which the teacher is qualified, and avoiding clashes and hard workload limit overloads. Next, from each unassigned meeting the algorithm searches for ejection chains as described above. Finally, split assignments are introduced, in which the classes of unassigned meetings are split between two or more qualified teachers. Split assignments are undesirable, so it is important to minimize the number of unassigned meetings at the point they are resorted to. A full description has been given elsewhere (Kingston 2008).

It is interesting to compare the performance of this algorithm with a standard local search (tabu search). It is not clear how to do so fairly, however, since the author's algorithm never introduces a clash or a hard workload limit overload, preferring to leave a meeting unassigned, whereas tabu search assigns a teacher to every meeting, at the cost of some clashes and hard workload limit overloads. There is no simple and fair means of interconversion as there was for the bipartite matching problem studied earlier.

When run on a typical instance, the author's algorithm produced a resource assignment in which there were 22 unassigned meetings after the initial assignment, and 15 after the ejection chain phase. This was a significant improvement, since it meant that 7 fewer meetings required split assignments. For comparison only, this solution was extended greedily to one in which every meeting had an assignment, and that solution had 34 clashes and hard workload limit overloads.

The best run of tabu search (Fig. 7) on the same instance produced 23 clashes and hard workload limit overloads. Again for comparison only, this solution was reduced greedily to one in which there were no clashes or hard workload limit overloads, and that solution had 22 unassigned meetings. Even if we call these results a draw, the ejection chain method still has considerable advantages: it runs much faster, and there are no parameters to tune.

7 NP-completeness analysis

Sometimes the outcome of problem analysis is not an idea for an efficient algorithm, but instead the discovery of a connection with another NP-complete problem. Even this apparently negative result may be useful, however, in suggesting that prior work on the other problem may be relevant, either directly or with some adaptations.

Two examples of this occur in examination timetabling, one well-known, the other less so. The well-known example is the connection with vertex colouring. Algorithms from the

Fig. 7 Performance of tabu search on a typical teacher assignment problem (*bghs93* from Kingston 2008) with 305 teacher slots totalling 1275 times. The cost measure was the total number of clashes and hard workload limit overloads. After 4000 moves, starting from an initial greedy solution with cost 79, the best solution found had cost 23. The neighbourhood was all moves of one assignment from one qualified teacher to another. A move was tabu if its slot had been moved recently; the tabu list length was 50, which gave the best result of about ten values tried.

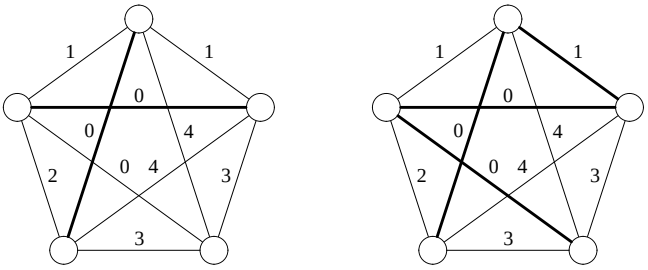

Fig. 8 A clash graph, showing a minimum matching (left), and a travelling salesman path (right).

vertex colouring literature, such as the saturation degree heuristic and Kempe chains, have been adapted to examination timetabling in many papers.

To uncover the less well-known connection, it is necessary first to dispose of the 'no-clashes' constraint that points to vertex colouring. This may be done, for example, by a phased approach whose first phase clusters examinations so that there are as many clusters as time slots. For simplicity, this discussion will assume from now on that there are as many examinations as time slots, and that the aim is to assign one examination to each time slot.

With clashes out of the way, the remaining problem is to minimize cases of students having examinations too close together in time. This requirement can be formalized in several ways, two of which will be examined here.

Construct the familiar *clash graph*, where each examination (or cluster of examinations) is represented by a node, and each pair of nodes is joined by an edge weighted by the number of students who attend the examinations of both nodes. One formulation is to have two examination time slots on each day, and aim to minimize the number of cases of students attending two examinations on the same day. This corresponds to finding a maximum matching of minimum total weight in the clash graph, which can be done in polynomial time (Papadimitriou and Steiglitz 1982). Another formulation does not consider the division of time slots into days, but merely their sequencing in time, and aims to minimize the number of cases of students having consecutive examinations. This corresponds to finding a travelling salesman path (like a travelling salesman tour, but with no requirement to end at the starting point) in the clash graph. These constructions are illustrated in Fig. 8.

It is not suggested that these ideas provide an immediate solution to the examination timetabling problem. Rather, they make connections with other work that might bear fruit when suitably adapted. For example, to the author's knowledge, no attempt has been made to adapt the work of Glover (1996) on ejection chain neighbourhoods for the travelling salesman problem to examination timetabling.

8 Conclusion

This paper has highlighted the algorithms and complexity perspective on timetable construction, and shown by example that its use can be beneficial.

It is difficult to offer guidance in the application of the techniques advocated here, since they must be adapted to specific details of the problems under study. Familiarity with the list of standard algorithms and NP-complete problems is a prerequisite. In searching for algorithmic ideas, a focus on sets of related variables is often rewarding: the schedules of all sports teams in a given local area, the room requirements for all meetings at a given time, and so on. Indeed, where algorithms and complexity techniques have advantages, these seem to be due to their ability to handle sets of related variables together, rather than one by one as local search and integer programming solvers do. This point is illustrated repeatedly throughout this paper.

There is no practical barrier to combining algorithms and complexity techniques with other perspectives in timetabling research; the possibilities are limited only by our ingenuity. There is however one point of philosophical disagreement that should not be glossed over.

Metaheuristics are *general* approaches to optimization problems, easily applied to any problem. Integer programming, too, is a general approach. From the algorithms and complexity perspective, generalization is bad, not good, because it brings with it a corresponding weakening of the tools available to solve the problems. For example, the consequences of treating bipartite matching as a general optimization problem were demonstrated in Sect. 5. Thus, the algorithms and complexity perspective tends to lead the researcher towards specific details; other perspectives often lead in the opposite direction.

Problem analysis is a very hit-and-miss process, but it does have the advantage of being open-ended. One can always hope to find a new algorithm, or a new connection with another problem. And when something does turn up, the payoff can be large. For these reasons, the algorithms and complexity perspective will continue to have a place in timetabling research.

References

Ahuja R, Ergun Ö, Orlin J, Punnen A (2002) A survey of very large-scale neighbourhood search techniques. Discrete Applied Mathematics, 123:75–102

Beasley JE (1993) Lagrangean relaxation. In Reeves CR (ed.), Modern Heuristic Techniques for Combinatorial Problems, Blackwell

Burke EK, Gendreau M (2008) Proceedings, PATAT2008 (Seventh international conference on the Practice and Theory of Automated Timetabling, Montreal)

Burke EK, Marecek J, Parkes AJ, Rudová H (2008) A branch-and-cut procedure for Udine course timetabling. In: Proceedings, PATAT2008 (Seventh international conference on the Practice and Theory of Automated Timetabling, Montreal)

Carter MW (2000) A comprehensive course timetabling and student scheduling system at the University of Waterloo. In Practice and Theory of Automated Timetabling III (Third

International Conference, PATAT2000, Konstanz, Germany, Selected Papers), Springer Lecture Notes in Computer Science 2079:64–81
Csima J, Gotlieb CC (1964) Tests on a computer method for constructing school timetables. Communications of the ACM 7:160–163
Dowsland KA (1993) Simulated annealing. In Reeves CR (ed.), Modern Heuristic Techniques for Combinatorial Problems, Blackwell
Easton K, Nemhauser G, Trick M (2003) Solving the travelling tournament problem: a combined integer programming and constraint programming approach. In: Practice and Theory of Automated Timetabling IV (Fourth International Conference, PATAT2002, Gent, Belgium, August 2002, Selected Papers), Springer Lecture Notes in Computer Science 2740:100-109
Glover F (1996) Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65:223–253
Glover F, Laguna M (1998) Tabu Search, Kluwer
Gotlieb CC (1962) The construction of class-teacher timetables. Proc. IFIP Congress, 73–77
Kingston JH (2008) Resource assignment in high school timetabling. In: PATAT2008 (Seventh international conference on the Practice and Theory of Automated Timetabling, Montreal)
Kirkpatrick S, Gellat CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
Kostuch P (2005) The university course timetabling problem with a three-phase approach. In: Practice and Theory of Automated Timetabling V (5th International Conference, PATAT 2004, Pittsburgh, PA), Springer Lecture Notes in Computer Science 3616:109–125
Martin RK (1999) Large scale linear and integer optimization: a unified approach. Kluwer
Meyers C, Orlin JB (2007) Very large-scale neighbourhood search techniques in timetabling problems. In: Practice and Theory of Automated Timetabling VI (Sixth International Conference, PATAT2006, Brno, Czech Republic), Springer Lecture Notes in Computer Science 3867:24–39
Müller T, Rudová H, Barták R (2005) Minimal perturbation problem in course timetabling. In: Practice and Theory of Automated Timetabling V (5th International Conference, PATAT 2004, Pittsburgh, PA), Springer Lecture Notes in Computer Science 3616:126–146
Murray K, Müller T, Rudová H (2007) Modeling and solution of a complex university course timetabling problem. In: Practice and Theory of Automated Timetabling VI (Sixth International Conference, PATAT2006, Brno, Czech Republic), Springer Lecture Notes in Computer Science 3867:189–209
Papadimitriou CH, Steiglitz K (1982) Combinatorial optimization: algorithms and complexity. Prentice-Hall
Ribeiro CC, Urrutia S (2004) Heuristics for the mirrored travelling tournament problem. In: Proceedings, PATAT 2004 (5th International Conference on the Practice and Theory of Automated Timetabling, Pittsburgh, PA), 323–341
De Werra D (1971) Construction of school timetables by flow methods. INFOR – Canadian Journal of Operations Research and Information Processing 9:12–22