Variants of simulated annealing for the examination timetabling problem

Jonathan M. Thompson and Kathryn A. Dowsland

Statistics and O.R. Group, European Business Management School, Swansea University, Singleton Park, Swansea SA2 8PP, UK

This paper is concerned with the use of simulated annealing in the solution of the multi-objective examination timetabling problem. The solution method proposed optimizes groups of objectives in different phases. Some decisions from earlier phases may be altered later as long as the solution quality with respect to earlier phases does not deteriorate. However, such limitations may disconnect the solution space, thereby causing optimal or near-optimal solutions to be missed. Three variants of our basic simulated annealing implementation which are designed to overcome this problem are proposed and compared using real university data as well as artificial data sets. The underlying principles and conclusions stemming from the use of this method are generally applicable to many other multi-objective type problems.

Keywords: Timetabling, simulated annealing, graph theory, multi-objective optimisation.

1. Introduction

Many scheduling and timetabling problems have a primary objective of finding a feasible solution in which a set of required events are scheduled into a given time period without conflict. However, in practice there are usually a number of secondary objectives or constraints which relate to the quality of the timetable. Many of these may conflict with each other or with the primary objectives and there is usually some hierarchy of importance, with some factors being regarded as more vital than others. Where there are more than two or three different objectives, it is often impossible to produce good solutions using single-phase methods, and a multi-phase approach is frequently adopted, with the more important objectives being considered in earlier phases and the less important ones being delayed until a later phase. This has the disadvantage that the decisions made in the initial phases cannot be undone, yet they may have a detrimental effect on the overall quality of the final solution.

This paper is concerned with the problem of multi-objective examination scheduling. We suggest an approach based on simulated annealing. Our initial implementation goes some way towards overcoming the problems outlined above. It solves the problem in phases, but allows decisions made in earlier phases to be undone later, as long as the quality of the solution with respect to earlier objectives does not deteriorate. Although this approach is an improvement over strictly phased methods, it is still possible that not enough of the initial decisions will be reversed to produce good solutions with respect to the minor objectives. We suggest variants to the standard simulated annealing algorithm which are able to overcome this problem. In the next section, the examination timetabling problem is described in detail, followed by a full explanation of our solution method. We then go on to describe the potential problem with this approach and outline three variants intended to surmount it. The results of tests on both live and artificially generated examination timetabling data are presented.

2. The examination timetabling problem

Although the major requirement when producing an examination timetable is that no student be scheduled to sit two exams simultaneously, many establishments define other secondary objectives. The most common are listed below.

(1) Minimise the total examination period or, more commonly, fit all exams within a given time period.

(2) Minimise the number of students scheduled to sit simultaneous exams. This is known as minimising first-order conflict.

(3) Increase student comfort by spacing the exams fairly and evenly across the whole group of students.

(4) Place certain exams within pre-specified time windows. This includes exams which are pre-assigned to a set date.

(5) Schedule certain pairs or groups of exams in the same time period.

(6) Schedule subgroups of exams in a specific order.

(7) Timetable large exams towards the beginning of the exam period to give lecturers maximum time to complete their marking.

(8) Allocate each exam to a suitable room.

(9) Allocate suitable invigilators to each exam. For certain exams, invigilators with a certain amount of relevant knowledge may be required.

(10) Maximise the number of exams which can be moved to another time period without disrupting the rest of the schedule. In the event of a last minute change having to be made, this maximises the probability that an exam can be moved to a new period without introducing first-order conflict.

The relative importance attached to these different aspects will vary from institution to institution and may range from that of a binding constraint to less vital secondary objectives. It is clear that many of these are in conflict. For example, students on popular courses will tend to be involved in large exams. Hence, scheduling these towards the start of the exam period will increase the number of students having two consecutive exams. Even if only a subset of the above requirements are involved, the problem is a true multi-objective optimisation problem.

3. Solution methods

Until recently, the many solution methods which had been proposed for the examination timetabling problem tended to fall into two categories. Some of the earliest are greedy construction heuristics, which attempt to solve the problem in a single pass. They tend to be oversimplified because they have no look-ahead facility, leading to large costs being incurred towards the end of the process. An example of this is the algorithm suggested by Broder [1] in which each exam is allocated in turn to the "best" available period until all have been scheduled. Such methods are able to solve simple problems extremely quickly, but may perform poorly on highly constrained problems. Our own experiments with a greedy heuristic for scheduling exams at Swansea University support this point of view.

To overcome this problem, multi-phase methods which split the problem into several parts and solve each separately were proposed. For example, Lotfi and Cerveny [2] split the problem into several distinct phases. Phase one groups all exams into a given number of exam blocks while ensuring no student has two exams in the same block. These blocks are then allocated to days while minimising the number of students with same-day exams. Phase three arranges the exam days and the final phase allocates exams to rooms. Each phase can then be solved using more sophisticated methods than are possible if the entire problem is to be solved in a single pass, but this may be to the detriment of later phases.

The problem of allocating the exams to blocks such that no two exams in the same block are in conflict is most commonly modelled and solved as a graph colouring problem. Each vertex of the graph represents a different exam and edges join any two vertices which represent any pair of exams requiring a common resource or which are both taken by the same student. A feasible colouring of the graph (i.e. an allocation of colours to the vertices such that all pairs of adjacent vertices have different colours) corresponds to a feasible timetable, with the exams mapped onto each set of vertices in the same colour being held at the same time.

Once a suitable partition has been established, phase two is usually concerned with spacing out the exams, but the precise objectives may differ from institution to institution. A number of different solution approaches have been suggested. For example, Arani et al. [3] and Balakrishnan et al. [4] formulate variants of the problem as integer programs and solve them using Lagrangian relaxation. Arani et al. minimise the number of students with same-day examinations using a set covering model with side constraints. Balakrishnan et al. use a network model to solve the problem of minimising the number of students with exams in consecutive time slots, commonly known as second-order conflict. The nodes in their network correspond to block/time slot pairs and edge weights correspond to the number of students taking exams in the blocks represented by both endpoints. The optimal solution corresponds to the shortest path traversing the entire network and visiting exactly one node associated with each block. The position of this node corresponds to the timing of this block. Computational results are presented for the problem of minimising the number of students with consecutive exams on a single day. However, with appropriate edge weights, the model is equally valid for the more general problem of minimising second-order conflict. Others have suggested heuristic approaches. For example, Johnson [5] uses simulated annealing to minimise the number of students with same-day exams. As stated earlier, the drawback of multi-phase methods is that the solution obtained from an early phase is then fixed in later phases and may lead to poor overall performance. Where there is a requirement for time windows and pre-assigned exams, such methods are often inappropriate since there is very little freedom to arrange the exam blocks once these have been determined.

Recent approaches have attempted to overcome these difficulties. Eiselt and Laporte [6] solve the problem in phases but allow exam blocks determined in the first phase to be broken up in later phases. They produce a hierarchy of requirements, placing the objectives into three levels of importance. They then define the objectives in the first two levels of importance as constraints and create an initial conflict-free solution using what is essentially a greedy heuristic with some backtracking. The second phase involves using a random descent algorithm to improve the solution by moving single exams into different time periods. Although this allows greater flexibility than standard phased methods, the final solution is still largely dependent on the solution obtained in phase one. This problem was noted by Hertz [7], who solves the problem using tabu search. All types of conflict are combined into a single weighted cost function, and an arbitrary initial solution is improved by moving one exam at a time using the principles of tabu search to explore the solution space. He reports success with this method for both school and examination timetabling problems, but does not indicate how the weights for the cost function were determined. Carter [8] gives a comprehensive review of examination timetabling methods.

Our proposed solution method is based on an extension to the graph colouring model. Although the basic model is concerned only with the problem of eliminating clashes, many of the additional constraints outlined previously can be included. Exams which have to be scheduled in the same time period are merged into one vertex. As suggested by Balakrishnan [9], time windows can be incorporated by adding a clique (i.e. a set of vertices which are all pairwise adjacent) of dummy vertices, one for each time slot. Edges are added between time-windowed exam vertices and any unsuitable time slot vertices. A pre-assigned exam will be adjacent to all time slot vertices except for the one relating to its specified date. Room capacities can be incorporated by including vertex weights equal to the number of students sitting each exam, and information concerning the number of students with clashes or consecutive exams can be represented by edge weights. This model is the basis of our simulated annealing approach. Thus, we consider the problem as a graph colouring problem with additional objectives and side constraints.

4. Simulated annealing

Sequential meta-heuristics such as simulated annealing and tabu search have been used successfully in the solution of graph colouring problems (Chams et al. [10], Dubois and de Werra [11]). Here, we use simulated annealing, although many of our general comments and conclusions are equally applicable to other neighbourhood search methods. Simulated annealing is briefly summarised in figure 1, but for a full discussion, see Dowsland [12]. It is an iterative procedure which searches the set of possible solutions through a pre-defined neighbourhood structure. Starting from a randomly generated solution, a neighbouring solution is sampled and compared with the current one according to an appropriate cost function. If the cost function is improved, the sampled solution is accepted. Only accepting improving moves would leave the search trapped in the first local minimum encountered. Simulated annealing overcomes this by also accepting some inferior solutions but controlling such acceptances using a probability function. This function is dependent on a parameter t and moves are more likely to be accepted for higher values of t. Because of the origin of simulated annealing in the field of statistical thermodynamics, t is referred to as the temperature and is decreased or cooled during each run. If t is allowed to become sufficiently small, the search eventually becomes trapped in a local minimum. Solution quality is dependent on the rate of cooling and the optimal cooling schedule will vary from problem to problem.

A common approach to solving multi-objective problems with simulated annealing is to combine all objectives and constraints into a single linear cost function using a series of weights and penalties. This is the approach used by Hertz and Abramson [13] and Abramson and Dang [14] also report a successful implementation of simulated annealing with a weighted cost function for the solution of the school timetabling problem. In the latter case, selection of weights did not appear to be a problem, with all but one of the objectives being given equal weight. However, it is worth noting that their objective was to obtain a zero cost solution with respect to all elements of the cost function. When this is not the case, the selection of such weights can be very difficult. For example, Dige et al. [15] addressed the school timetabling problem and found that weights which reflected the importance of each objective did not produce satisfactory solutions. Wright [16]

```
Simulated Annealing Procedure

Given a function $f$ to be minimised over a given solution space

Step 1. Produce an initial solution $S$.
Step 2. Define an initial temperature $t > 0$.
Step 3. Do the following $L$ times:
        3.1. Choose $S^1$, a random neighbour of $S$.
        3.2. $\delta = f(S^1) - f(S)$.
        3.3. If $\delta \leq 0$, then $S = S^1$, otherwise set $S = S^1$ with probability $\exp(-\delta/t)$.
Step 4. Set $t = r * t$, where $r$ is some cooling parameter $< 1$.
Step 5. If $t$ is still significant, goto step 3.
Step 6. Return $S$ or best solution achieved during run.
```

Figure 1. Simulated annealing procedure.

points out that when selecting weights, one should consider the importance of the objective, the difficulty of achieving it and the way in which the objectives interact with the chosen neighbourhood structure. Such methods may produce acceptable results for problems with two or three objectives, but cannot be expected to solve more difficult ones. We therefore chose to use a variant of the phased approach in which the more important objectives are considered in the earlier phases, similar to that suggested by Eiselt and Laporte [6]. This method has more flexibility than traditional phased approaches in that some decisions made during earlier phases may be undone during subsequent ones. This is achieved by gradually reducing the solution space so that solutions which are inferior with respect to earlier phases are removed. By using simulated annealing instead of a straightforward descent approach, the final solution should be less dependent on the solution to phase one. While this overcomes many of the drawbacks associated with phased methods, it is possible that the removal of large numbers of solutions may disconnect the solution space with respect to the neighbourhood structure. This means that one of the underlying assumptions for the success of simulated annealing, that of the reachability of all solutions from all others, will be violated.

5. Implementation

As aforementioned, our basic model is that of graph colouring and our underlying solution space and neighbourhood structure are those proposed by Chams et al. [10]. Thus, a feasible solution is any partition of the vertices into k subgroups or colour classes and the neighbourhood consists of all those solutions obtained by changing the colour of a single vertex. At the start of each phase, constraints are imposed by removing any solutions which violate them from the solution space and the cost function is chosen to reflect the objectives of that phase. Ideally, each phase would involve a single objective, thereby eliminating the problem of choosing appropriate weights. However, where large numbers of objectives are involved, it is more practical to combine them into groups of similar importance. This approach was implemented as the basis of an exam scheduling package at Swansea University. In order to fully describe this implementation, we firstly need to outline the objectives and constraints in operation there.

At Swansea, the examination period consists of twenty-four time slots with Sunday breaks. We are concerned only with degree exams, which are all scheduled in the morning. Thus, each time slot corresponds to an actual date. The exams must be scheduled within this period without any student clashes and with a limited number of desks available at any one time. Certain exams are subject to time windows and/or pre-specified sequences. There is also a preference for large exams to be scheduled within the first two weeks and the number of students with exams on consecutive days to be kept as small as possible. Thus, all except the final two requirements are in practice binding constraints. Our objective in phase one is therefore to find a feasible timetable subject to all binding constraints and to optimise the secondary objectives in phase two. We decided to do this by reflecting the pre-orderings, time windows, and the fixed number of time slots in our definition of solution space in phase one and minimising the unweighted sum of first-order conflict and room overflow. In our case, this policy of equal weighting worked well, but this may be due to the fact that the number of desks available is far in excess of those required. Thus, this part of the objective is easy to satisfy. The break in consecutive days between Saturdays and Mondays was incorporated by allowing Sundays to form feasible time slots but setting the room capacity for them at zero. This implementation ensures that the reachability condition is satisfied because the constraints imposed on the solution space do not disconnect it. In practice, phase one solved very easily and we therefore decided to initialise the search with a good solution obtained using a greedy heuristic. This often produced an optimal solution to this phase, i.e. a solution with no student clashes and no room overflow. Where this was not the case, annealing quickly solved the problem.

The solution space in phase two is therefore restricted by excluding all solutions which include student clashes or room overflow. We decided to include both the objective of placing all large exams early and that of minimising second-order conflict into the cost function. The initial solution is obviously the output from phase one, but a new cooling sequence with different parameters is initiated. Due to the ease with which all large exams could be scheduled in the first two weeks of the exam period, we decided to place a heavy penalty on such exams falling outside of this period. In practice, this two-phase approach gave satisfactory results for the 1993 timetable. However, the solution space for phase two is considerably reduced and therefore it may be disconnected. Thus, the optimal solution may not be reachable from the initial solution by a series of valid neighbourhood moves.

6. Improved solution methods

Although, as in our case, acceptable results may be achieved using this basic solution method, better results would be possible if we could circumnavigate this problem. In this section, we describe three variants of our annealing algorithm which allow us to do this. They are:

(1) perform several independent runs using different starting solutions,

(2) change the neighbourhood structure to include more connections,

(3) penalise infeasible solutions rather than removing them from the solution space.

All three methods are applicable to our problem and the way in which we implemented is fully explained below.

6.1. PERFORM SEVERAL INDEPENDENT RUNS

If the solution space has been split into several disjoint parts, then performing an independent search in each section will ensure that the global optimal solution is reachable from some starting solution. However, purely random starts cannot guarantee that different sections of the solution space will be explored. Therefore, in generating a sequence of different starting solutions, an intelligent form of diversification is required in order to increase the likelihood of each one being situated in a different part of the solution space. This is similar to the idea of diversification or a long-term memory function in tabu search (Glover [17]) in which an attempt is made to learn about that part of the solution space already explored, and to use this information to force the algorithm to move on to new areas.

We can identify two types of information which can be easily recorded. The simplest and most obvious is to record, for each vertex, the set of colours which have been allocated to it. At the diversification stage, as many vertices as possible should be allocated new colours in the new starting solution. However, this may result in the same groups of vertices being placed in the same colour class as each other. This can be avoided if we record those pairs of vertices which have been in the same colour class at any time during the search. Diversification may then attempt to split pairs of vertices which have remained the same colour throughout or, alternatively, to force non-adjacent pairs which have never been the same colour into the same colour class.

In our experiments, we deal only with the placing of vertices into new colour classes. Information concerning the colours allocated to each vertex is only recorded from phase two because this is the phase where the solution space may be disconnected. The results are then used to re-initialise phase one. It is possible that phase one may push vertices back into old colours. Rather than waste time during phase two, the output of phase one is tested to ensure that at least one vertex takes on a completely new colour. However, it should be noted that the dummy time slot vertices are not recoloured since each represents a set day. Thus, this approach could also be regarded as a restricted version of the second approach in which pairs of exam vertices and time slot vertices which have never taken on the same colour are forced into the same colour class. This process can be repeated for several runs and has a natural endpoint when all vertices have taken on all feasible colours. As the number of runs increases, more and more time is required to obtain a feasible solution to phase one as increasingly more vertices have to be recoloured. It should be noted that this approach does not guarantee that each starting solution is in a separate component of the solution space. However, research into the performance of the long-term memory function in tabu search has shown that such diversification is still valuable even when the solution space is not known to be disconnected. Thus, this solution method has the double advantage of overcoming the problems of a disconnected solution space and ensuring that the search within each component is sufficiently diverse. Taking this strategy to its extreme when each vertex has been coloured in every possible colour may be very time consuming, but the process can be stopped before this stage is reached depending on the amount of time available.

6.2. CHANGE THE NEIGHBOURING STRUCTURE

An alternative to initialising several searches to cover a disconnected solution space is to redefine the neighbourhood structure so that the space is no longer disjoint. This can be achieved by widening the neighbourhoods so that each solution has more immediate neighbours. However, there is a trade-off between solution time and solution quality because more iterations may be necessary to adequately search larger neighbourhoods. A general strategy for increasing the neighbourhood is to increase the number of elements moved or exchanged. Our neighbourhood currently consists of the set of solutions which differ from the current solution by the colour of a single vertex. Therefore, in our case this would involve extending the number of vertices to be recoloured, for example, including moves where the colours of two vertices are exchanged. Figure 2 illustrates an example where the solution space is disconnected under our original neighbourhood structure, but is connected when we allow two vertices to swap colours. The cost function represents the sum of the edge weights between nodes in consecutive colours, and is clearly lower for the solution shown in figure 2(b). The solution shown in figure 2(a) has no neighbours under our original neighbourhood structure, but if we allow swaps, the solution in figure 2(b) can be reached in two moves by swapping the colours of vertices 5 and 6, followed by 4 and 5.

In general, a large proportion of swaps would be infeasible and, hence, much time would be wasted examining such moves. It is also possible that this would not be sufficient to completely reconnect the solution space. Morgernstern and Shapiro [18] suggested Kempe chains as an alternative neighbourhood structure for the graph

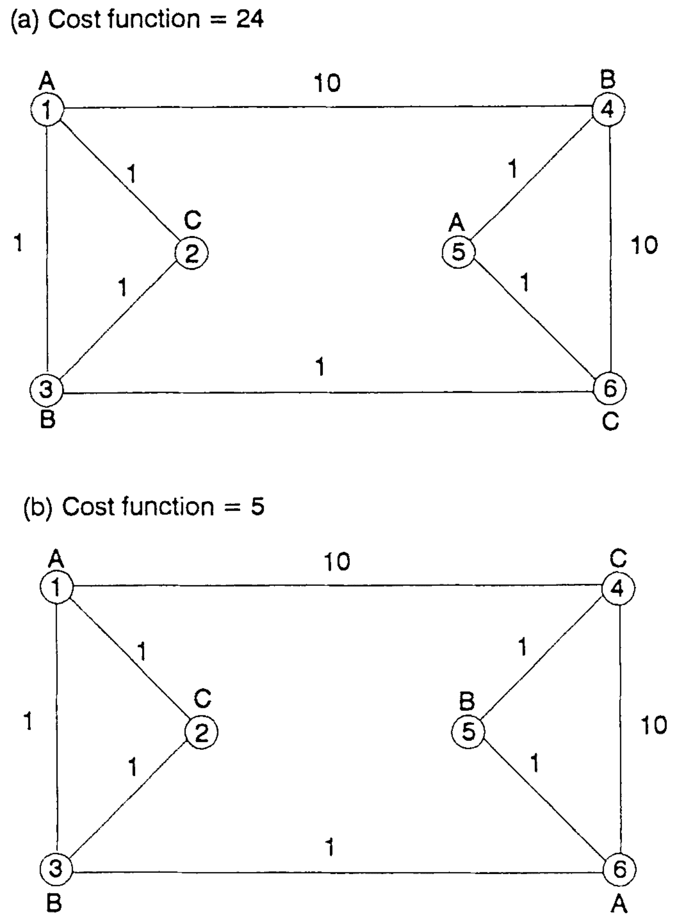

Figure 2. An "unreachable" colouring under a standard neighbourhood. Colour A corresponds to time period 1, B to 2, and C to 3.

colouring problem. Although Kempe chains are only moderately successful when the objective is to minimise the number of colours, there is no reason to suppose that they will not be successful for our cost function. An i–j Kempe chain is a connected component of the subgraph induced by the set of vertices coloured i or j. An example of a graph and its corresponding Kempe chains is shown in figure 3. Because any j coloured vertices adjacent to any i coloured vertices in a given chain are themselves in that chain, the colours of the vertices within a chain can be swapped without affecting the feasibility of the colouring. A Kempe chain neighbourhood of a given colouring is defined as the set of colourings obtained by swapping the colours on a single Kempe chain. A single chain can be generated by firstly choosing a vertex, v, coloured i. Any adjacent vertices coloured j are then added to the chain, as are any vertices coloured i or j which are adjacent to any vertex currently in the chain.

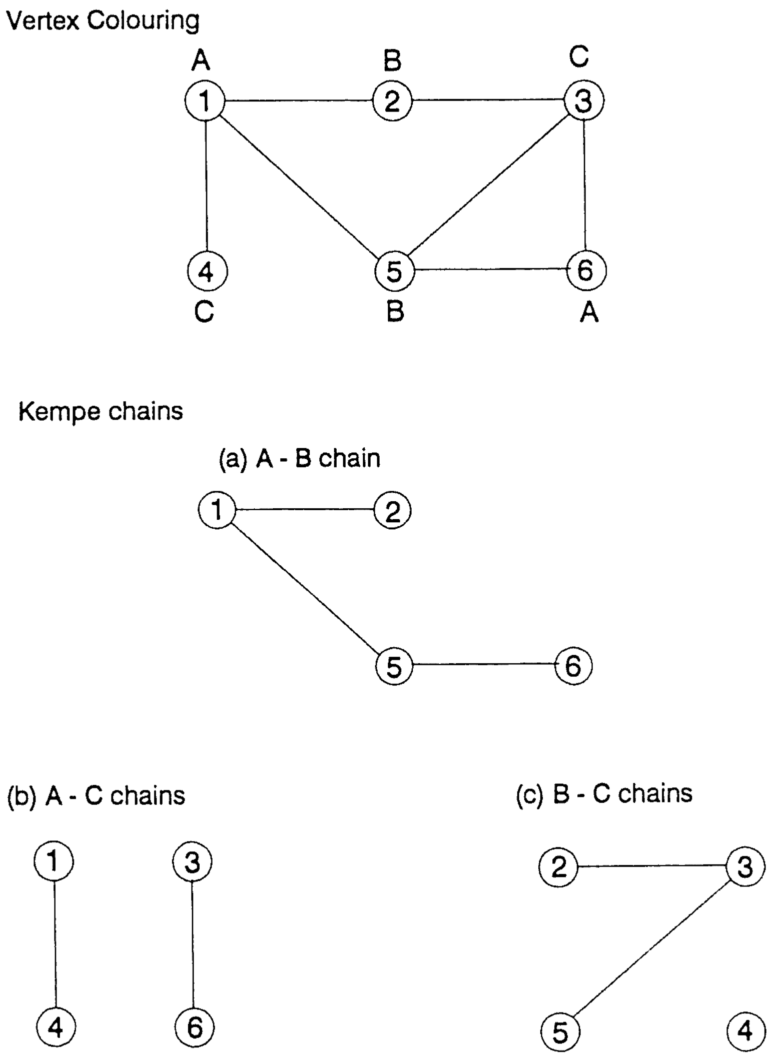

Figure 3. A vertex colouring and its Kempe chains.

Note that all feasible moves from the standard neighbourhood will be valid Kempe chain moves since any vertex which can be recoloured from i to j has no neighbours coloured j and is therefore an isolated vertex forming its own i–j Kempe chain. Likewise, all feasible 2-swap moves are included.

In our implementation, Kempe chains are generated randomly using the method described above and selecting randomly a colour i, a vertex in that colour, and then a second colour j. The colours on the corresponding i–j Kempe chain are then swapped. The only exception to this is if the chain includes a time slot vertex. Swapping such a chain will reallocate the colours to days. In order to avoid this, we leave the vertices in the chain unchanged but swap the colours of all other vertices coloured i or j. This produces the same vertex partitions as swapping the original chain, but avoids complex updating of our cost function.

However, it should be noted that a neighbourhood based on Kempe chains is not guaranteed to reconnect the solution space. Figure 4 shows two solutions which cannot be reached from one another using a series of Kempe chain moves. The reason for this is that each Kempe chain is full, i.e. contains all the vertices in either of its

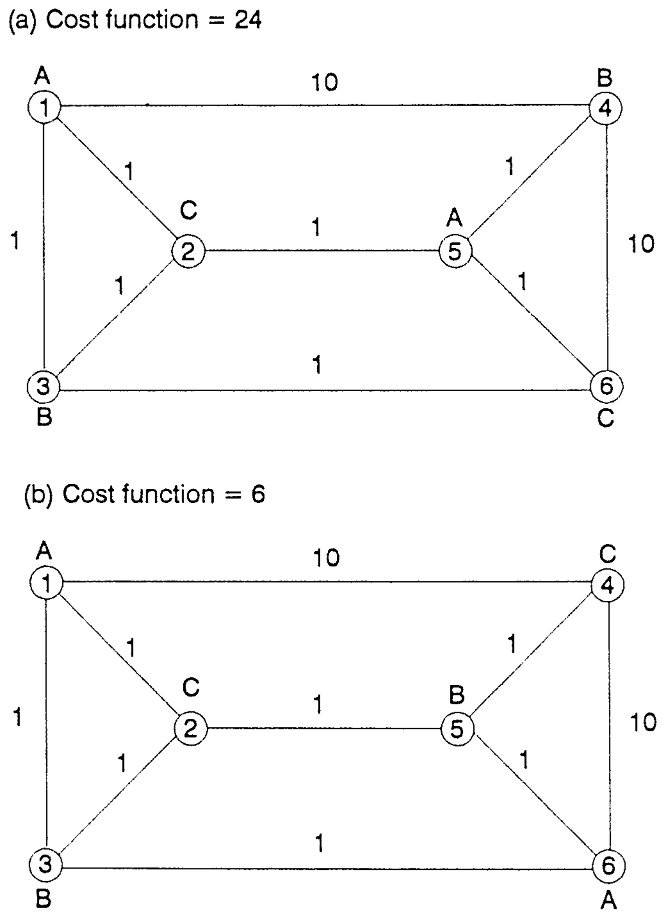

Figure 4. An "unreachable" colouring under a Kempe chain neighbourhood. Colour A corresponds to time period 1, B to 2, and C to 3.

two colours. Therefore, the vertices grouped together in the colour classes will not change although the colours of any two classes may be swapped. Any subset of vertices which cover all colours and which satisfy these conditions cannot be re-coloured by a series of Kempe chain moves. In situations where the room constraints are more restrictive, this may also make a number of Kempe chain moves infeasible, thereby increasing the possibility of the space remaining disconnected. Sampling Kempe chains is more computationally expensive than sampling single vertices, but the access it provides to a greater part of the solution space may make it worthwhile.

6.3. PENALISE INFEASIBLE SOLUTIONS

We have already noted the possibility of using weights in a single-phase method to guide the search towards solutions which satisfy the more important constraints while optimising the secondary objectives. It is also possible to introduce weights into a multi-phase method, using them to penalise undesirable solutions in later phases rather than eliminating them completely from the solution space. If such an approach is to work correctly, it is obviously important to get the correct balance between the weights, the cooling schedule and the rest of the cost function. As highlighted by Dige et al. [15], this can be particularly difficult if the secondary objectives cannot all be satisfied simultaneously. Clearly, placing too high a weight on any objective has the effect of eliminating highly penalised solutions from the solution space, while very low weights will mean that the objective concerned will be practically ignored until all others have been optimised.

If the starting temperature for phase two is sufficiently high for effective annealing, it is likely that a relatively large number of clashes will be reintroduced at the start of phase two, suggesting that the time spent eliminating all clashes in phase one may be wasted. We deal with this problem in two ways. Our simplest approach dispenses with phase one and combines first-order conflict, room overflow, second-order conflict and the number of large exams out of place into a single weighted sum. In order to get the balance between first- and second-order conflict correct, we decided to measure first-order conflict as the number of students with clashing exams, rather than the number of exams themselves, since this allows us to measure both types of conflict on a similar scale. First-order conflict must obviously be weighted higher and we experimented with several weights. Room overflow was given a weighting of 10 per student with no seat, and large exams out of place were weighted at 200.

Our second approach maintains the two separate phases. In order to impose greater control on the degree to which the quality of solutions to earlier phases is allowed to deteriorate, we impose a threshold so that only solutions whose quality exceeds this threshold are included in the solution space. In practice, these threshold limits were used such that moves which created first-order conflict greater than their value would be accepted, but as soon as this occurred, the algorithm switched to the next phase. The problem exists that unless the weights can be chosen to ensure that all local minima are optimal with respect to earlier objectives, it is possible that some degree of deterioration in these objectives will be evident in the final solution. If these objectives do not represent binding constraints, then some deterioration may be acceptable. However, if, as in our case, the phase one objectives are binding, this will not be satisfactory. Our solution is to change the cost function once the algorithm has converged with respect to the weighted cost function, so that only the penalties on earlier objectives remain. Rather than continue to anneal at low temperatures, we perform random descent based only on first-order conflict at this stage. However, we found that this was not always successful in removing all clashes.

This multi-staged approach can be summarised as follows.

1. Anneal using the reduced neighbourhood to convergence. (This ensures at least one good solution in phase two.)
2. Allow first-order conflict to enter the solution space and let t rise slowly according to $t \rightarrow t/1 - \alpha t$, until the clash penalty reaches a predefined threshold.
3. Apply standard annealing using a weighted cost function, to convergence.
4. If no feasible timetable has been found in stage 3, use random descent based solely on first-order conflict.

Although extending the solution space in this way will reconnect it, the problem in allocating weights makes it difficult to ensure that the search is guided in the right direction. Ideally, only those solutions necessary to bridge the gaps should be reintroduced, but we do not know how these may be identified. It is also worth noting that the solution space may be expanded in other ways, by relaxing the constraints implicit in the definition of feasibility in phase one. An example of this type of expansion is used by Abramson and Dang [14] in the school timetabling problem, where they allow additional dummy time periods. Use of these periods is penalised in the cost function. This facilitates neighbourhood moves which are equivalent to swaps, without incurring the high penalty values which would result if the swap had been achieved using two separate moves. This approach would theoretically allow all our Kempe chain moves, and many others, to be generated as a series of moves under the standard neighbourhood. However, since we are unlikely to obtain a zero cost solution with respect to second-order conflict, setting the weights to ensure that the dummy periods are not used in the final solution could prove problematical. Given the relative lack of success of the two penalty function methods described above, we did not carry out any experiments on this type of expansion.

7. Computational experience

7.1. THE DATA

The purpose of the computational experiments was twofold; firstly, to compare the performance of simulated annealing with that of a simple greedy heuristic and random descent, and secondly, to compare the various improvements outlined above.

The algorithms were coded in FORTRAN and experiments were conducted using a DEC Alpha. We used a number of different data sets based on genuine 1993 Swansea University data, in which 570 exams were to be timetabled into a fixed exam period of four weeks and data was available for over 3000 students. One hundred exams were time-windowed, including 49 which were preassigned. There were 23 large exams to be scheduled within the first two weeks, while an additional 17 exams were specified as especially difficult to mark and therefore an early date was requested. Various exams had to be in a set order and 1200 desks were available at any one sitting. This data set and four variants form the basis of our experiments. These are outlined below.

(1) *Full data set.*

(2) *Tightened data set.*

In the original data, the room capacity constraints were fairly loose and we thought them unlikely to affect the connectedness of the solution space. Therefore, we reduced the number of desks available at any one sitting from 1200 to 800. Previously, room utilisation was just 46%, i.e. an average of 46% of desks would be used at any one time. Decreasing the number of desks available puts a greater strain on our algorithms because the room utilisation rises to 69%.

(3) *Kempe data set.*

Since we do not know whether or not our solution space in phase two is disconnected, it was decided to construct a set of data which we knew to be disconnected using our standard neighbourhood, but with at least two components which would be reconnected using the Kempe chain neighbourhood. This was achieved by adding additional vertices to data set (1) and ensuring that these extra vertices had an optimal colouring not reachable from the initial colouring using the standard neighbourhood, but reachable when using a Kempe neighbourhood. Edge weights were chosen so that the initial colouring produced a heavier cost function value than the alternative colouring which can only be reached via Kempe chains. This effectively adds a penalty of 900 units of second-order conflict to any solution in the "wrong" component of the solution space. These additional vertices were connected to the standard graph using edges which maintained the feasibility of all solutions encountered during a single run when using the original data set.

(4) *Disconnected data set.*

The logic behind this data set is similar to that behind data set (3). Our aim here was to produce a disconnected solution space which could not be overcome using Kempe chains. This was achieved by small alterations to data set (3).

(5) *Cost free data set.*

The above data sets allow us to compare the different methods, but do not allow us to measure overall solution quality. Therefore, we constructed a data set with a known optimal cost of zero. This was done by taking a good solution and removing all edges which contributed to the cost function. The total number of edges was restored to its original value by adding new, random edges which did not add to the cost function. Thus, we have a solution of zero cost. However, it should be noted that this solution will have several zero cost neighbours, and it is possible that other zero cost solutions may have been generated in other areas of the solution space.

7.2. INITIAL EXPERIMENTS

Preliminary experiments used a greedy heuristic to obtain some foundation results. The exams were ordered such that the time-windowed and large exams preceded the rest and the exams within each group were sorted according to the number of conflicting exams. Each was then allocated to the period which minimised the secondary objectives while not breaking any binding constraints (if no free time slot was available, the algorithm terminated unsuccessfully). As anticipated, results confirmed the inadequacies of such methods and substantiated the need for a more sophisticated solution method.

As stated in section 5, it is relatively easy to obtain a feasible timetable for data set (1) by the method described for phase one, and there is no reason why the methods used to generate data sets (2) to (5) should make this phase more difficult. Therefore, the remaining experiments concentrate on phase two. In order to get the best out of simulated annealing, it is necessary to optimise the parameters for the cooling schedule. There are a number of theoretical approaches to this, e.g. White [19]. However, we found that setting the starting temperature according to this method required ridiculously high values. We therefore chose the parameters as follows. The starting temperature needs to be high enough to accept a large proportion of moves. The maximum increase in cost will be twice the number of students in the exam being moved and will occur if an exam with no second-order conflict is moved to a position where all students have exams in the preceding and following slots. A starting temperature of 20 allows a reasonable probability of accepting this type of increase for exams of average size, and this was therefore adopted. The smallest possible cost increase is 1 unit. A final temperature of 0.1 was selected since this gives a probability of less than 1 in 10^4 of accepting any uphill move. The approximate size of our neighbourhoods in phase 2 is 4000 and we therefore chose to conduct 10,000 iterations per temperature. We experimented with five different cooling ratios between these limits. These were carried out on data sets (1) and (2) using five separate runs, with different random number streams for each data set and each set of parameters. The results are summarised in table 1. The numbers in this and all

Table 1

Results for the standard annealing implementation.

Cooling ratio	Date set (1)			Date set (2)		
	Average	Best	Time	Average	Best	Time
0.8	439.4	424	54.01	457.2	432	52.59
0.9	395.8	365	133.66	389.2	369	121.82
0.95	371.0	345	253.88	377.4	352	249.77
0.97	354.4	331	447.19	374.2	359	426.24
0.99	330.4	313	1357.65	337.0	315	1314.06

Times are in CPU seconds.

subsequent tables show the number of cases of students having second-order conflict (this excludes second-order conflict due to consecutive pre-assigned exams). In all cases except where otherwise indicated, all other objectives were completely satisfied. All solution times given are in CPU seconds. As expected, solution quality increases with slower cooling, but at the expense of longer solution time. Since the results for the slowest rate of 0.99 appear to be significantly better, 0.99 was used in the remaining experiments. However, the run time may be prohibitive for practical situations and faster cooling may be necessary.

Table 2

Comparison of greedy heuristic, random descent and simulated annealing.

	Data set (1)	Data set (2)	Data set (3)	Data set (4)	Data set (5)
Greedy	441	n/s	n/s	n/s	212
RD	619*	860**	1547**	1547**	491.8
SA	330.4	337.0	1225.6	1225.6	26.0

n/s: some exams could not be scheduled while obeying all binding constraints.
* In four of the five cases, the large exams could not be fitted into the first two weeks of the exam period.
** In three of the five cases, the large exams could not be fitted into the first two weeks of the exam period.

Results using this ratio were compared with the greedy heuristic and with random descent in which only improving moves were accepted. These results are summarised in table 2 and show that simulated annealing outperforms the other two methods. Comparing the results for data sets (1) and (2) with those in table 1 indicates that this remains true even for the faster cooling ratios. It is particularly interesting to note the poor performance of both the greedy heuristic and random descent on data set (5), which has a known optimal solution of zero.

7.3. MAIN EXPERIMENTS

The second set of experiments were intended to measure the success of the various methods proposed for overcoming the reachability problem. We deal first with diversification. Since each diversification will have the same characteristics as an ordinary run, diversification experiments were carried out using the same parameters as standard annealing. Complete runs, when termination occurs when phase one produces a solution with no vertices in new colours or all vertices have been every colour, were deemed impractical. This was because early results showed that many diversifications were needed before these stopping conditions were met, yet solution quality just fluctuated without homing in on particularly good solutions. In order to give a more meaningful comparison with other methods, it was decided to limit the number of diversifications to two (i.e. three complete runs). The results are shown in table 3. Table 3(a) shows the outcome of five diversification runs on

Table 3

Results for diversification method: diversification limited to two.

(a) Complete results using data set (1)

	Seed # 1	Seed # 2	Seed # 3	Seed # 4	Seed # 5
Result 1	322	336	313	355	326
Result 2	360	346	350	322	317
Result 3	325	290	318	326	313

(b) Summary of all results

	Data set (1)	Data set (2)	Data set (3)	Data set (4)	Data set (5)
Average	312.0	319.2	326.0	313.4	19.2
Best	290	311	310	298	14
Time	3966.75	3847.88	4248.38	3952.54	3844.53

Times are in CPU seconds.

data set (1), using different random number streams. The large number of differing results indicates a large number of local optima in our problem, showing the difficulty of finding the optimal solution. It also shows that this diversification method did achieve its stated objective, that of taking the search to different parts of the solution space. However, it is apparent that there were also significant differences among the different random starts. This is probably due to the size of the solution space. It should be noted that the random starts cannot guarantee to move to a new area of the space, whereas our diversification method can. Table 3(b) shows a summary of the diversification results for all the data sets. For each diversification run, the best cost function value achieved was taken and the results given are the average of these over five different random runs. The most noticeable observation is that the diversification method dealt effectively with data set (4), which we know to be disconnected. The other data sets may all be connected and little benefit can apparently be gained by diversifying.

Before comparing Kempe chain annealing with the other strategies, we need to determine suitable parameters. Table 4 shows the results for Kempe chain annealing on the first two data sets with different cooling ratios. The parameters used are the same as for standard annealing. Although there is scope for larger changes in the

Table 4

Results for Kempe chain annealing.

Cooling ratio	Date set (1)			Date set (2)		
	Average	Best	Time	Average	Best	Time
0.8	312.8	304	98.41	331.0	320	90.80
0.9	299.8	284	212.86	316.4	290	183.51
0.95	280.0	272	419.43	290.6	275	382.60
0.97	284.0	277	703.89	283.8	277	643.52
0.99	263.4	257	2166.07	276.2	271	1867.20

Times are in CPU seconds.

cost function in Kempe chain annealing, this did not seem to be the case in practice and, therefore, the same starting and stopping temperatures were used. Neighbourhoods are larger under Kempe chain annealing, about 6000 in size, but it was felt that more iterations per temperature were not necessary. The results were significantly better than those obtained by standard annealing, with the cooling ratio of 0.99 again working well.

Results were then obtained from the three remaining data sets. These are shown in table 5. All results were obtained using a cooling ratio of 0.99. The results show that the Kempe chain neighbourhood does produce superior results and this is especially noticeable for data set (3). Here, standard annealing produced very poor results, while Kempe chains consistently performed well. This was not the case for data set (4), as expected, where neither method achieved worthwhile results. These results too can be compared with diversification (table 3) and show the significant advantage of using Kempe chains over the diversification strategy in terms of both solution quality and solution time. Results from data set (5) stress this point and show that Kempe chains came extremely close to obtaining the optimal solution of zero, and in fact achieved near-optimal results for all random number streams.

Table 5

Results for standard annealing and Kempe chain annealing on the remaining data sets.

	Standard annealing			Kempe chain annealing		
	Average	Best	Time	Average	Best	Time
Data set (3)	1225.6	1204	1460.64	264.6	254	2035.64
Data set (4)	1225.6	1204	1391.67	1161.4	1152	2072.32
Data set (5)	26.0	17	1342.22	2.6	1	2091.14

Times are in CPU seconds.

Further experiments were conducted in order to determine whether improved solutions could be obtained by using a standard neighbourhood followed by a Kempe chain neighbourhood. The rationale behind this was to obtain a decent solution fairly quickly and then to improve on this more slowly using Kempe chains. This was achieved by running through the entire cooling schedule on the original neighbourhoods, and then reheating the temperature so that half the number of iterations conducted during the standard run would be conducted during the Kempe chain run. The results are shown in table 6 and show that we did not get any benefit from this method, with results being poorer than those obtained by pure Kempe chain annealing. This indicates that annealing at higher temperatures is an important part of the search and in a pure Kempe chain run, the search traverses the solution space towards good solutions at these temperatures.

Table 6

Results for standard annealing followed by Kempe chain annealing.

	Average	Best	Time
Data set (1)	269.2	259	2552.00
Data set (2)	274.6	265	2371.28
Data set (3)	274.4	260	2635.05
Data set (4)	1164.2	1149	2590.92
Data set (5)	4.4	1	2437.41

Times are in CPU seconds.

Experiments using the penalty method were conducted, initially using data set (1). The first set of experiments simply combined first-order conflict, room overflow, second-order conflict and the number of large exams out of place as a weighted sum and annealed in a single phase. Different weightings of the first-order conflict as compared to second order were used. These results are shown in the first four

Table 7

Results for the penalty method on data set (1).

	Average	Best	Time
Pen = 2	n/s	–	–
Pen = 3	n/s	–	–
Pen = 10	360.0*	354	1556.22
Pen = 20	371.4	350	1586.31
Pen = 3, t = 50	599.8	451	2316.72
Pen = 10, t = 50	407.0	396	1723.81
Pen = 3, t = 100	507.6	392	1899.27
Pen = 10, t = 100	463.4	406	2007.28

Pen = weight of first-order conflict, t = threshold value.
n/s: no feasible solution was found.
*Only two out of the five runs produced a feasible solution.
Times are in CPU seconds.

rows of table 7. With low penalties, it appeared that the algorithm struggled to return to solutions of zero cost in terms of the primary objectives. The higher penalty of 20 did ensure that a feasible solution is found, but first-order conflict is unlikely to re-enter the solution. Therefore, using such a penalty is equivalent to performing phases one and two in a single phase but with fewer total iterations than were used for standard annealing. The second group of experiments used the staged method. First-order conflict was penalised by factors of 3 and 10 when compared with secondary conflict, and threshold limits were set at 50 and 100. These results are shown in the bottom four rows of table 7. Annealing was conducted with a cooling rate of 0.99 and iterations per temperature were again 10,000. Combining a penalty of 10 with a relatively low threshold of 50 resulted in feasible solutions from stage 3 in all cases. In general, lower penalties and higher thresholds increased the likelihood of requiring stage 4. Results are poor in comparison with other methods. Further research into this method may lead to improved results, but for the purposes of this study, no further experimentation was carried out.

Finally, table 8 summarises all solution methods on all five data sets. The results clearly show the superiority of the Kempe chain neighbourhood over all other methods, except in the case of data set (4). So far, we have not been able to reap any benefit from the penalty function approach, although it is an improvement on both the greedy heuristic and random descent. The Kempe chain approach is the best method of those proposed and works well on all our connected data sets.

When comparing the results, it is important to consider the total run times. Such long run times resulted from our determination to ensure each method was given the maximum opportunity to produce its best solution, in order that our methods

Table 8

Average results from all solution methods.
GH: greedy heuristic; RD: random descent; SA: standard annealing;
KCA: Kempe chain neighbourhood; MIR: more independent runs; PF: penalty method.

	Data set (1)	Data set (2)	Data set (3)	Data set (4)	Data set (5)
GH	441	n/s	n/s	n/s	212
RD	619[*]	860[**]	1547[**]	1547[**]	491.8
SA	330.4	337.0	1225.6	1225.6	26.0
KCA	263.4	276.2	264.6	1161.4	2.6
MIR	312.0	319.2	326.0	313.4	19.2
PF	371.4	–	–	–	–

n/s denotes no feasible solution was obtained.
[*] From five different random number streams, in only two cases were results obtained with all objectives satisfied apart from second-order conflict.
[**] From five different random number streams, in only one cases was a results obtained with all objectives satisfied apart from second-order conflict.
Times are in CPU seconds.

themselves could be fairly compared. In practice, many universities prefer to use a standard personal computer such as a 486pc, where the average time to complete 1000 iterations under a standard neighbourhood was 12.6 seconds compared with 15.3 seconds under a Kempe chain neighbourhood. The improved solution quality when using Kempe chains would seem to make the extra solution time worthwhile; however, complete run times on a pc with the best parameter settings would be in the region of eighteen and a half hours. Therefore, either faster cooling or fewer iterations at each temperature may be necessary, although a lower starting temperature and higher finishing temperature would also save time without having a large effect on solution quality. With faster cooling, results equivalent to those obtained using a cooling ratio of 0.8 can be found within half an hour's run time.

8. Conclusions

We have laid down an effective framework for solving multi-objective problems using simulated annealing. The proposed solution method groups the objectives into sets of similar importance and optimises each set in turn. Previously optimised objectives are considered as binding constraints in later phases and this is achieved by removing unsuitable solutions from the solution space. This phased approach appears to be especially useful for problems where weighting difficulties occur. The nature of simulated annealing means decisions made in earlier phases can be undone in later ones. However, this may disconnect the solution space. We introduced three ways of dealing with this, of which two proved satisfactory. The penalty approach proved too difficult in terms of setting suitable weights, but both diversification and, in particular, the neighbourhood widening, were successful. In our case, this was achieved using Kempe chains. It is worth commenting that this is a result of modelling the examination scheduling problem as a graph colouring problem. It was only because this model was used that Kempe chains were considered as a possibly improved neighbourhood for our problem. Therefore, although a model may not be necessary in order to use a technique such as simulated annealing, it may be useful to consider appropriate models in order to generate new ideas for improving solution quality. The Kempe chain neighbourhood vastly outperformed our standard annealing implementation, but does not guarantee to satisfy the reachability condition. The other two methods designed to satisfy the reachability condition are either excessively time consuming or of poor quality and, hence, are not generally recommended although the diversification option may be better in very disconnected solution spaces. It would be extremely useful to be able to ascertain for any data set whether or not the solution space is disconnected. This is a matter for future research and may also provide clues as to which solutions should be reintroduced for a penalty approach.

The 1993 Swansea University timetable, produced using the described basic simulated annealing method proved to be entirely acceptable. Some unilateral changes had to be made, following publication of a draft timetable, due to errors in the initial data. Therefore, the final timetable produced had 490 cases of students sitting two exams in two days, a vast improvement over previous years. The solution method is relatively simple, easy to apply, and can produce a feasible timetable in about one minute. The user can specify how long they are willing to wait for their results, thereby controlling the time used to optimise the secondary objectives. In practice, the cooling schedules used in these experiments may be too slow to form the basis of a practical solution method. For this reason, the 1993 Swansea timetable was produced using the undulating cooling schedule discussed in Thompson and Dowsland [20]. However, we plan to use the results of this research to produce an even better timetable for 1994, using a Kempe chain neighbourhood.

References

[1] S. Broder, Final examination scheduling, Commun. ACM 7(1964)494–498.

[2] V. Lotfi and R. Cerveny, A final-exam-scheduling package, J. Oper. Res. Soc. 42(1991)205–216.

[3] T. Arani, M. Karwan and V. Lotfi, A Lagrangian relaxation approach to solve the second phase of the exam scheduling problem, Euro. J. Oper. Res. 34(1988)372–383.

[4] N. Balakrishnan, A. Lucena and R.T. Wong, Scheduling examinations to reduce second-order conflicts, Comp. Oper. Res. 19(1992)353–361.

[5] D. Johnson, Timetabling university examinations, J. Oper. Res. Soc. 41(1990)39–47.

[6] H.A. Eiselt and G. Laporte, Combinatorial optimization problems with soft and hard requirements, J. Oper. Res. Soc. 38(1987)785–795.

[7] A. Hertz, Tabu search for large scale timetabling problems, Euro. J. Oper. Res. 54(1991)39–47.

[8] M.W. Carter, A survey of practical applications of examination timetabling by computer, Comp. Oper. Res. 34(1986)193–202.

[9] N. Balakrishnan, Examination scheduling: A computerized application, OMEGA 19(1991)37–41.
[10] M. Chams, A. Hertz and D. de Werra, Some experiments with simulated annealing for coloring graphs, Euro. J. Oper. Res. 32(1987)260–266.
[11] N. Dubois and D. de Werra, EPCOT – an efficient procedure for coloring optimally with tabu search, Comp. Math. Appl. 25(1993)35–45.
[12] K.A. Dowsland, Simulated annealing, in: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C. Reeves (Blackwell Scientific, 1993).
[13] D. Abramson, Constructing school timetables using simulated annealing: Sequential and parallel algorithms, Manag. Sci. 37(1991)98–113.
[14] D. Abramson and H. Dang, School timetables: A case study using simulated annealing, in: *Applied Simulated Annealing*, Lecture Notes in Economics and Mathematical Systems, vol. 396 (Springer 1993) pp. 104–124.
[15] P. Dige, C. Lund and H.F. Ravn, Timetabling by simulated annealing, in: *Applied Simulated Annealing*, Lecture Notes in Economics and Mathematical Systems, vol. 396 (Springer, 1993) pp. 151–174.
[16] M.B. Wright, Scheduling English cricket umpires, J. Oper. Res. Soc. 42(1991)447–452
[17] F. Glover, Tabu search – Part I, ORSA J. Comp. 1(1989)190–206.
[18] C. Morgemstern and H. Shapiro, Chromatic number approximation using simulated annealing, Technical Report CS86-1, Department of Computer Science, University of New Mexico (1989).
[19] S.R. White, Concepts of scale in simulated annealing, *Proc. IEEE Int. Conf. on Computer Design* (1984) pp. 646–651.
[20] J.M. Thompson and K.A. Dowsland, Multi-objective university examination scheduling, Working Paper No. EBMS/1993/12, EBMS, Swansea University (1993).