Brief Papers

A Multistage Evolutionary Algorithm for the Timetable Problem

E. K. Burke and J. P. Newall

***Abstract*—It is well known that timetabling problems can be very difficult to solve, especially when dealing with particularly large instances. Finding near-optimal results can prove to be extremely difficult, even when using advanced search methods such as evolutionary algorithms (EA's). This paper presents a method of decomposing larger problems into smaller components, each of which is of a size that the EA can effectively handle. Various experimental results using this method show that not only can the execution time be considerably reduced but also that the presented method can actually improve the quality of the solutions.**

***Index Terms*—Evolutionary algorithms, timetable problems.**

I. Introduction

A. The Timetable Problem

THE timetabling problem consists of allocating a number of events to a finite number of time slots (or periods) such that the necessary constraints are satisfied. In general, the nature of the constraints varies between different instances of the timetable problem, but most problems share the following conditions.

- No individual entity (e.g., person) should be required to attend two events simultaneously.
- For each period there should be sufficient resources (e.g., rooms) to service the events scheduled within that period.

These constraints are fundamental to any timetabling problem and generally form the basis for a feasible timetable but, as mentioned above, a number of other constraints may be introduced depending on the particular flavor of the timetable problem being considered. Often, these varying constraints can be at odds with each other. For instance, when scheduling university examinations it is often considered undesirable for students to have to sit for exams in adjacent periods. Alternatively, when scheduling university lectures it is often actually preferred for students to have two or three lectures in a row. Putting aside these differences for our experiments, we will be concentrating on an instance of the examination problem. While there are many possible side constraints for this problem, the most difficult task to carry out (aside from producing a feasible timetable at all) is often to minimize the number of conflicts between adjacent periods to allow students time to recover between exams. If a large number of periods are allocated, it would most likely be the case that this would not cause any problems. Unfortunately limits on time and rooms often mean that finding any feasible solution is a considerable task in itself and that no zero penalty solution will exist.

The timetable problem is known in general to be NP-complete [15], [21]. Of the many approaches that have been applied to the problem, the earliest methods are those that utilize heuristics. A good example of this is heuristic sequencing [13], which involves using a heuristic to estimate how difficult each event will be to schedule. If the events that are regarded as the most difficult are then scheduled before the other events we should produce a substantially better timetable than by merely using a random ordering. A useful addition to this process is backtracking, such as that used in [8]. This treats situations where there is no valid period available to schedule an event because of placements already made. It does this by unscheduling the conflicting events from a chosen period to allow the current event to be scheduled in that period. The unscheduled events are then placed back in the ordering for later scheduling.

In addition to these heuristic techniques, a number of general search methods have also been applied to the timetable problem. Such approaches include simulated annealing [19], Tabu search [14], and variants on genetic algorithms [2], [4], [10], [17]. While most of these have shown worth, it is notable that many of the more successful methods employ some kind of "trick" in the algorithm to improve results for timetable problems. It follows that specialized timetabling algorithms may function substantially better than those based on more general optimization techniques.

For the interested reader, more information on the examination timetabling problem and the methods that have been applied can be found in two excellent surveys [6], [7] which go into considerable depth. For further information about the problem itself, see [1] which presents and discusses the results of a questionnaire on examination timetabling that was sent to universities throughout the United Kingdom.

B. Memetic Timetabling

In [4], an approach was proposed for timetabling problems based on memetic algorithms. In essence, memetic algorithms are evolutionary algorithms that utilize local search to some extent. The concept of memetics originates from Dawkins [12] who describes the meme as an idea or concept that is passed around society. Individuals can then adapt ideas to suit

Manuscript received December 30, 1997; revised June 4, 1998 and November 10, 1998.

The authors are with the Automated Scheduling and Planning Group, Department of Computer Science, University of Nottingham, Nottingham, NG7 2RD U.K.

Publisher Item Identifier S 1089-778X(99)02081-0.

their own environment, in contrast with biological evolution where genes are passed down and cannot be altered by the recipient. In the same way that memes were related to genes by Dawkins, memetic algorithms were proposed as an alternative to genetic algorithms [11] by Moscato and Norman [16]. The main motivation of this approach is that by applying a hill-climbing operator after each of the genetic operators, we can then treat the process as a search through local optima, rather than the entire search space. Applying a local search operator like this will clearly cause each generation to take considerably longer but this can be justified if sufficiently more is achieved per generation than if local search were not used.

The memetic algorithm proposed in [4] uses a combination of mutation and local search to effectively explore the solution space. Mutation actually consists of two separate operators, light and heavy mutation. The light mutation operator merely shuffles a few individual events to other valid periods, the purpose of this being to "nudge" solutions away from local optima to find a new solution. The heavy mutation operator, on the other hand, functions by targeting periods where large amounts of penalty are arising. The algorithm then randomly reschedules the events in these periods to produce a new solution that retains some of the (hopefully better) characteristics of the original solution. Neither of these operators on their own achieves any substantial improvements in solution quality but when followed by an application of a simple hill-climber the process becomes much more effective [4].

II. Problem Definition

The technique presented here was tested on an instance of the examination timetabling problem. Instances of this problem usually involve hard and soft constraints. Hard constraints are those that must be satisfied (at all costs). Soft constraints, on the other hand, are regarded as desirable but it is not absolutely essential to satisfy all of them.

In this problem there are E exams that must be scheduled in P periods with S examination seats available for each period (where E, P, and S are nonnegative integers). There are three periods in a day. Any two of the exams may conflict with each other. This means that they have a number of students enrolled for both exams. When scheduling the exams we can encounter two types of conflicts.

- *First-Order Conflicts:* This term is used to describe situations where conflicting exams are scheduled in the same period. This is highly undesirable as it involves quarantining some students after one exam so they may sit for the other exam after the main sitting. In all but the most difficult of situations, this is regarded as a basic hard constraint.
- *Second-Order Conflicts:* These conflicts, on the other hand, are less important and represent situations where two conflicting exams are not scheduled in the period, but are scheduled in periods too near to each other. For instance we might not want students to have to sit for two exams in consecutive periods or to sit for two exams in the same day. To totally satisfy all of these constraints is often not practical. They are usually treated as soft constraints.

In addition to avoiding these conflicts, it is also very important to adhere to the limitations on available seating for each period. The problem can be formally specified by first defining the following:

- t_{ip} one if exam i is scheduled in period p, zero otherwise;
- c_{ij} the number of students taking both exams i and j;
- d_{pq} three if period p is on the same day as period q, one if they are on adjacent days, and zero otherwise. This provides a higher weight for same day conflicts because it is more important to satisfy them;
- s_i the number of students taking exam i.

To take into account that the construction of a feasible timetable might not actually be possible, we need an extra period, the $(P+1)$th period, into which any exams can be placed that are not yet scheduled in a valid period. There are no constraints to prevent scheduling in this period, but it will need to be heavily penalized. Now we need to minimize (1) subject to (2)–(4) (all presented below). Equation (1) sums up occurrences where students must attend two exams in consecutive periods and weights the number of unscheduled exams by 5000 to strongly discourage incomplete timetables. If two consecutive periods are on the same day then any adjacent conflicts are weighted by three. If there is a single night between them the conflicts are weighted by one, otherwise any conflicts are ignored (for example when periods are split by a weekend). Equation (2) states that every event should scheduled once, and only once in the timetable. Equation (3) specifies that no conflicting events should be scheduled within the same period and (4) enforces the condition that the total number of seats required for any period is not greater than the number of seats available. The problem can be presented formally as follows.

Minimize

$$\sum_{i=1}^{E-1}\sum_{j=i+1}^{E}\left[\sum_{p=1}^{P} t_{ip}t_{j(p+1)}c_{ij}d_{p(p+1)} + t_{ip}t_{j(p-1)}c_{ij}d_{p(p-1)}\right] + 5000t_{i(P+1)} \quad (1)$$

subject to

$$\sum_{p=1}^{P+1} t_{ip} = 1, \quad \forall i \in \{i, \ldots, E\} \quad (2)$$

and

$$\sum_{i=1}^{E-1}\sum_{j=i+1}^{E}\sum_{p=1}^{P} t_{ip}t_{jp}c_{ij} = 0 \quad (3)$$

and

$$\sum_{i=1}^{E} t_{ip}s_i \leq S, \quad \forall p \in \{1, \ldots, P\}. \quad (4)$$

III. A Multistage Memetic Algorithm

A. Basic Framework

While memetic algorithms show promise for timetabling problems [4], [17] the time involved in optimizing large

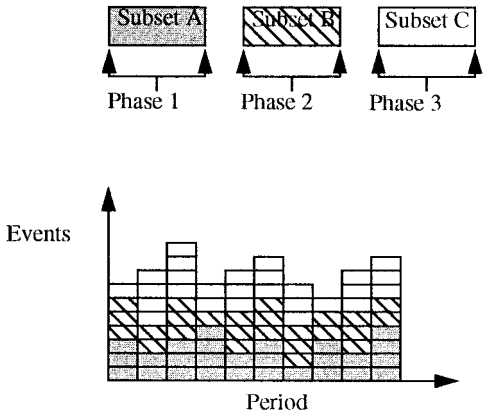

Fig. 1. The problem is divided into three subsets which are scheduled one at a time.

problems (of say greater than 500 events) is much longer than would be ideal. The memetic algorithm would be a more desirable approach if it could produce a schedule in a matter of minutes rather than hours. It might achieve this goal by first considering a subset of events. After fixing those events, the algorithm could consider another subset and add those to the previously created schedule. The process of decomposition has been studied by Carter [5], who proposed a heuristic method of recursively splitting a large problem into smaller problems until each subproblem is small enough to be solved by conventional methods such as linear integer programming.

The idea has also been studied by Weare [20] who applied such a technique to random data on flexible length timetables. The results indicated that as the number of events considered at one time is decreased, the time taken to produce solutions also decreases, but unfortunately so did the quality of solutions. A contributing factor to the lower quality could have been the use of the flexible-length timetable model and a preference for shorter timetables. This may have caused shorter timetables to be produced in the earlier phases (at the expense of second-order conflicts) when the final timetable may be much longer regardless. Therefore for the purposes of our experiments we use a fixed-length timetable model to avoid these problems. We also experiment with modifications and enhancements for this method. Fig. 1 shows how a set of events could be split into three subsets and then scheduled in three different phases, where the darker portions of the graph represent how early on in the process that portion was fixed in the timetable.

This should substantially reduce the complexity of the problem but there will of course be the obvious pitfall that by fixing events in periods in this fashion it may be impossible to schedule events later on in the process. Fortunately, there are a number of methods that can be employed to reduce the chances of this happening. First, it is possible to borrow an idea from heuristic sequencing methods and allow the algorithm to choose subsets according to some heuristic ordering. This should help improve the quality of produced timetables by optimizing first those events that are likely to cause most problems. Second, the algorithm could look ahead in the process by optimizing two subsets at a time but only fixing the first of these subsets at the end of every stage. This will

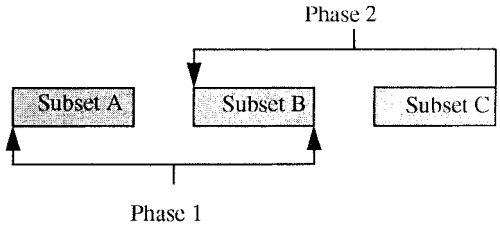

Fig. 2. Using a look-ahead approach, two subsets are now considered at one time but only the first is fixed at the end of a phase.

inevitably lead to the process taking longer than if a single subset of the same size was considered, as the algorithm is considering most of the events twice, but this could be justified if it leads to substantial improvements in quality. Fig. 2 illustrates how the earlier three-phase example would be handled using a look-ahead approach.

Having this look-ahead approach will help, but it is probably more important to choose a good heuristic to order the events. If the algorithm concentrates on scheduling those more difficult events earlier on in the process and reducing the penalty caused by these events with respect to each other, we should find that fewer problems are encountered later on in the process, regardless of whether or not a look-ahead approach is used. For these experiments we will use three heuristics that are generally accepted to be suitable for the exam timetabling problem [9].

- *Largest Degree:* The first events to be chosen for scheduling are those with the greatest number of conflicts with other events. These would generally be considered to be more difficult.
- *Color Degree:* This is similar to largest degree except that the events that are scheduled first are those that have the greatest number of conflicts with other events that have already been scheduled. It is expected that events with conflicting events already scheduled will be more difficult to place than events with a large amount of conflicts, but little or none of these events already scheduled.
- *Saturation Degree:* The first events to be scheduled here are those events with fewer valid periods remaining in the timetable. The expectation here is that if these events were not to be scheduled earlier on in the process then there may no valid periods at all remaining to schedule these events later on.

We can use the formal definition of the problem to define the above terms unambiguously. The *degree* of an exam is defined in (5), where # gives the cardinality of a set. As the degree of an exam is constant this need only be calculated once in any run of an algorithm

$$\text{degree}_i = \#\{j \in \{1, \ldots, E\}: c_{ij} > 0\}. \tag{5}$$

Similarly the *colorDegree* of an exam is defined in (6). Unlike the degree, the color degree of an exam changes every time one of its conflicting exams is moved from the unscheduled list to a valid period

$$\text{colorDegree}_i = \#\{j \in \{1, \ldots, E\}: (\exists p \in \{1, \ldots, P\}) \cdot (t_{jp} = 1 \ \wedge \ c_{ij} > 0)\}. \tag{6}$$

Finally the *saturation degree* of an exam is defined in (7). This will need to be recalculated every time an event is moved to a period, either from the unscheduled list or another valid period

$$\text{satDegree}_i = \#\left\{ p \in \{1,\ldots,P\}: \sum_{j=1}^{E} t_{jp}c_{ij} = 0 \ \wedge \ \sum_{j=1}^{E} t_{jp}s_j \leq S - s_i \right\}. \quad (7)$$

The multistage process when scheduling N exams at a time can be described by the following pseudocode.

- $E' = 0$
- **REPEAT**
  - $E' := E' + N$
  - **IF** $E' > E$ **THEN**
    - $n = N(E' - E)$
    - $E' := E$
  - **ELSE**
    - $n = N$
  - **FOR** each unscheduled exam
    - Calculate the desired degree of that exam
  - Pick the n exams with the largest/least degrees
  - Apply the memetic algorithm to schedule those n exams, keeping the $(E' - n)$ previously scheduled exams fixed in their current position.
- **UNTIL** $E' = E$

The point at which to exit the memetic algorithm presents a choice. Normally the memetic algorithm would exit when the population has fully converged but this is inappropriate with our mutation-driven memetic algorithm. Two other options would be to either run the algorithm for a fixed number of generations, or alternatively to stop the algorithm after a set number of generations have passed without finding a new "best so far" solution. In these experiments the algorithm was halted after five generations without a new "best so far" solution. While five generations may seem a little low, note that a generation usually achieves considerably more (while also consuming considerably more CPU cycles) than a typical evolutionary algorithm due to the use of the hill-climbing operator.

While the decomposition method is in itself independent of the particular technique used to solve each subproblem, the memetic algorithm presented in [4] has been shown to be effective at solving all but the very largest timetabling problems.

B. The Memetic Timetabling Algorithm

The memetic approach employs a simple evolutionary model with a population of solutions (of size 50 for all experiments described here). The algorithm applies mutation operators (one of the light or heavy forms, decided by a 50/50 probability), followed by a hill-climbing operator, to produce an oversize population (which is twice the normal population size). Solutions are then selected from this oversize set to form the new population for the next generation. Previous experiments [3] with recombination operators for this algorithm have proved unfruitful and as such were not used here.

1) Quality and Evaluation: A simple linear weighted penalty function is used to assess the quality (and hence the fitness, or rather unfitness) of solutions. This function is identical to that shown in (1).

2) Initial Population Generation: The initial population is generated by taking events in a random order and scheduling in the first valid period. The hill-climbing operator is then applied to each member of this population.

3) Light Mutation: As mentioned in Section I-B this operator selects a number of events (30 in this case) at random and tries to place each in an alternative period, also picked at random. Moves that violate hard constraints are not allowed, and in this case another period is tried.

4) Heavy Mutation: This operator is one of the more complicated operators in the algorithm, and we describe it in some detail. It targets periods that appear to be causing "large" amounts of penalty for "disruptions." We use the term "disruption" to mean that all events contained within a period are temporarily unscheduled. All events from disrupted periods can then be randomly rescheduled in the timetable. The penalty for a period p is based on the evaluation function and is defined in

$$\text{penalty}_p = \sum_{i=1}^{E-1} \sum_{j=i+1}^{E} t_{ip}t_{j(p+1)}c_{ij}d_{p(p+1)}. \quad (8)$$

What actually constitutes a "large" amount of penalty is clearly relative so the algorithm also needs something with which to compare. For this it can calculate the average penalty per period of the best solution in our population as in

$$\text{average} = \frac{\text{Fitness best}}{\text{Number of periods}}. \quad (9)$$

Having calculated these, the algorithm can now calculate a probability of being disrupted for each period. Equation (10) shows how the probability of being disrupted is calculated, with bias being a definable value (0.1 here) to vary the probability of periods being disrupted. Periods causing greater than average penalty are automatically disrupted

$$\begin{aligned} &\text{Probability (Disrupt period } p) \\ &\quad = 1, \text{ if } \ \text{penalty}_p \geq \text{average} \\ &\quad = \frac{\text{penalty}_p + (2 \times \text{bias})}{2 \times \text{average}}, \text{ if } \text{penalty}_p < \text{average}. \end{aligned} \quad (10)$$

The penalty arising from a particular period is dependant on the events scheduled in the next period so the algorithm must have special cases for disrupting two periods in a row. For instance, say the operator preserved period i because it had a low number of conflicts with period $i+1$; it does not make sense to then disrupt period $i+1$. The operator, however, may well want to disrupt the next period. If the operator were to pick out period $i+1$ for disruption, it would instead disrupt period $i+2$ automatically. In this case period $i+3$ would then be the next period to be considered.

This can be summarized formally as follows.

- **FOR** each period p (where $p>1$) in the timetable in turn
  - decide if p is to be disrupted based on the probability given by (10)
  - **IF** decision is to disrupt but previous period was not disrupted **THEN**
    - do not actually disrupt this period, instead disrupt the next period
  - **ELSE IF** disrupt **THEN**
    - disrupt this period
  - **ELSE**
    - do not disrupt this period

It can be seen from the pseudocode that heavy mutation will not disrupt the first period. This period, however, is not immune to changes made by either the light mutation operator or the hill-climbing operator and will normally change quite considerably over the course of the evolution.

5) The Hill-Climbing Operator: A simple efficient hill-climbing operator is applied after each mutation operator to restore solutions to local optima. This utilizes delta evaluation [18] to avoid the time-consuming process of performing a full evaluation at every step. Where (1) provides our fitness function, our delta evaluation function when moving event i from a period p to a period q is defined in

$$\Delta\text{fitness}(i,p,q) \sum_{j=1}^{E}\left(\left(t_{ip}t_{j(p+1)}c_{ij}d_{p(p+1)}+t_{ip}t_{j(p-1)}c_{ij}d_{p(p-1)}\right)-\left(t_{i(q,q+1)}t_{j(q+1)}c_{ij}d_{q(q+1)}+t_{iq}t_{j(q-1)}c_{ij}d_{q(q-1)}\right)\right). \tag{11}$$

The actual process used by the hill-climber is as follows.

- REPEAT
  - FOR each period p (in some random order)
    - FOR each event i scheduled in period p
      - Schedule event i in the valid period causing least penalty (this includes the original period)
    - Try and schedule any unscheduled events
- UNTIL no improvement can be made

While the hill-climbing routing does occasionally check if it can schedule events that are currently unscheduled, its main aim is to improve the quality of already feasible solutions, rather than produce low quality feasible solutions from infeasible solutions. A more complicated local search procedure might achieve this, but would inevitably increase the total time spent by the algorithm.

6) Selection: Selection is achieved by using a simple rank-based selection method, with each candidate being given a probability of being selected in order of quality. The process takes candidate solutions in order of fitness. The fittest solution (that with the lowest penalty) has a probability of being selected. If it does not get selected then the second fittest solution is offered the same chance. This continues until either a solution does get selected or the final solution is reached. The probability in these experiments was set to be 0.08, which appeared to give balanced results given the population size of 50.

7) Using the Memetic Algorithm Within the Multistage Framework: The operation of this algorithm varies very little when used within the proposed multistage framework. There are basically three differences at any particular phase of execution.

1) Any events fixed at the end of an earlier phase cannot be moved by *any* operator in the memetic algorithm, though they are still used for calculating conflicts, seat usage, etc.
2) The algorithm ignores all events that will be added to the problem in a later phase.
3) When generating the initial population for a particular phase the set of events for this phase is added in the same random order over the fixed events from previous phases.

Notice that the memetic algorithm does not perform any degree measures (see Section III-A). These are performed at the higher (multistage) level. The three points presented here apply to the memetic algorithm described in Section III-B. It is important not to confuse the memetic approach with the decomposition of the problem presented in the previous section. As mentioned earlier, the decomposition (Section III-A) is essentially independent of the memetic approach (Section III-B).

C. Implementation Details

1) The Memetic Algorithm:

Conflict checking: Due to the number of times conflicts between exams need to be checked, it is impractical to obtain the number of conflicts each time by comparing the lists of enrolled students for those exams. Fortunately as we are more interested in the *number of students involved in the conflict*, rather than the *actual students involved* in the conflict the comparisons need only be performed once. Having calculated the conflicts we can then create a conflict matrix C**.** This matrix C will have dimensions E by E, the definition c_{ij} from Section II-B being the (i,j)th element of the matrix. Storing this matrix within the algorithm enables rapid conflict checking and also eliminates the number of students as a factor in the complexity of the problem. For example, assume that four students must attend both exams a and b**.** In this case the (a,b)th element of the matrix will have the value four.

Data structures: Due to the hill-climbing operator being responsible for a large part of the operation of the memetic algorithm, the encoding of solution timetables is designed to reduce the time taken to evaluate and make small steps. Fig. 3 illustrates the link list style encoding used. Each period, plus the unscheduled list, has an associated object in the solution that knows the first event scheduled in that period. Similarly each event also has an associated object that knows the next event scheduled in that period or has a null pointer as it is the last event. By representing a timetable in this way we can easily move through the events in a period to check conflicts, etc. It also facilitates rapid movement of single events from one period to another, making it highly suitable for the hill-climbing operator. Aside from this, however, we may also need to quickly establish which period an event is scheduled

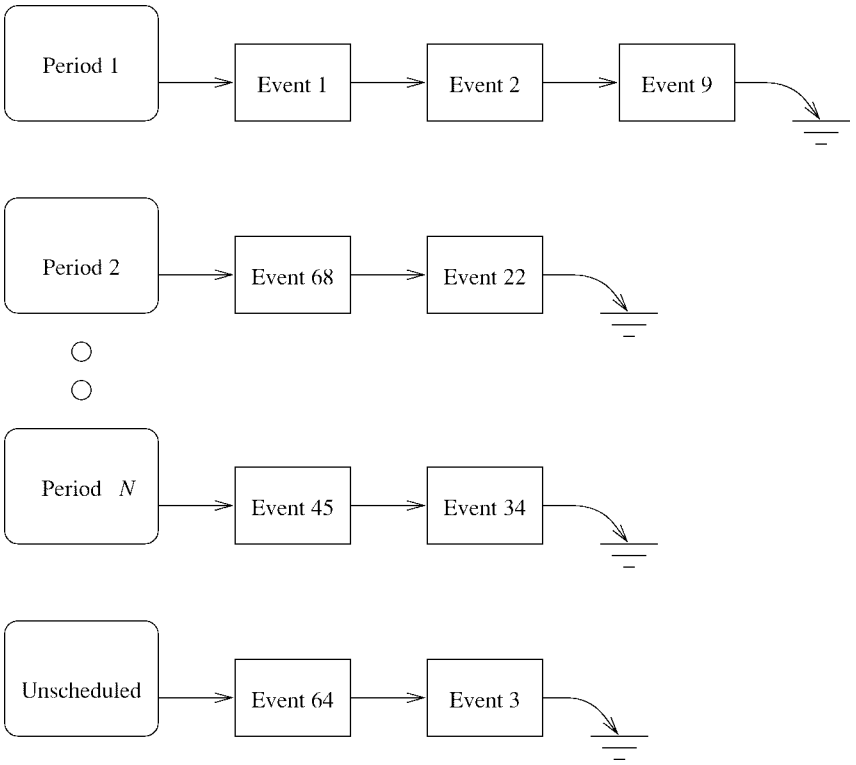

Fig. 3. The encoding used for a solution timetable. A linked list style approach is used that allows easy movement of events from one period to the other.

in. To address this, the event objects keep knowledge of which period its associated event has been scheduled in. By then storing all the event objects in an ordered array, it is possible to quickly access the information about any one event. Altogether, this structure provides for efficient manipulation and stable memory usage.

2) The Multistage Decomposition Method: There are some practical considerations when implementing this method; otherwise we could find that using it produces results little or no faster than the traditional single phase approach. The authors found that the main reason that this happens was the time taken to perform evaluations (and delta evaluations) at each stage of the algorithm. For instance, consider a problem with 400 events and suppose we have decided to schedule in subsets of 100 events with no look ahead. Now suppose that the algorithm is in the third phase. Two hundred events have already been fixed in the previous phase, and it is trying to schedule the next 100 events. Even though it is only placing those 100 events it must also consider the 200 events already scheduled when checking for first– and second-order conflicts, or other constraints. Similarly, in the final phase, it must still consider the 300 events that will have been fixed by then. In an algorithm such as ours where the majority of the execution time is spent either evaluating solutions, portions of solutions, or possible moves, it becomes clear that the time taken for each phase will increase substantially later on in the process when, ideally, each phase that considers an equal number of events should require an equal amount of time.

In the implementation used for the experiments here, the events that had already been fixed were merged into a number of virtual "super" events, one of these for each period in the timetable. These super events could be considered to be what would happen if all the students from all the events scheduled in a period so far were taken and were all registered on a new single event instead. It is then possible to forget about those initial events and remove them from the timetable, letting the new super events be used for evaluation purposes. This has obvious complexity advantages as the number of events involved in evaluations is now the subset size we have chosen plus the number of periods in the timetable, regardless of the phase that the algorithm is currently in (with the obvious exception of the first phase, where there is no need for any of these "super events"). Due to the way records of conflicts are handled (outlined in Section III-C1), there is no more overhead for checking conflicts with a "super" event than there is for a single event.

Fig. 4 illustrates how the timetable might build up phase by phase using this approach. The actual process of building a "super event" is basically a case of summing up the conflict matrices of the component events and the size of each event. For instance, if a new super event e_p is to be created from a number of events $e_1, \ldots, e_n$ scheduled in period p, then the event e_p would have the properties shown in (12). Note that as there must be a super event for each period the number of real events would now be given by $E - P$

$$c_{e_p i} = \sum_{j=1}^{n} c_{e_j i}, \quad \forall i \in \{1, \ldots, E\}$$
$$s_{e_p} = \sum_{j=1}^{n} s_{e_j}. \tag{12}$$

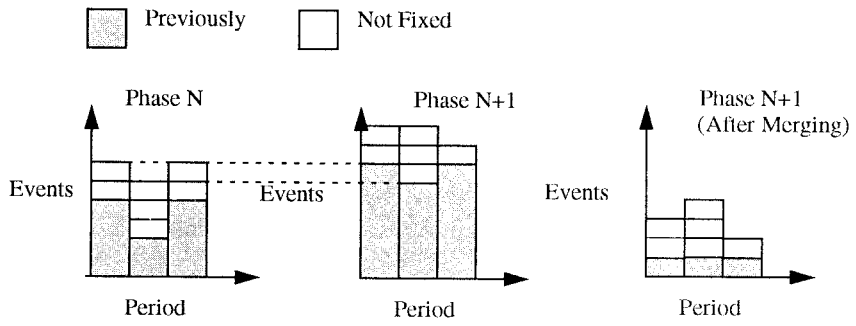

Fig. 4. Merging fixed events to improve performance. All merged events in a period can then be treated as a single "super event."

TABLE I
THE REAL DATA SETS USED FOR TESTING

Code	Institution	Number of Exams	Enrolment	Seats per Period	Number of Periods	Density of Conflict Matrix
carf92	Carleton University (1992), Ottawa	543	55,552	2,000	36	0.14
kfu	King Fahd University, Dharan	461	25,118	1,955	21	0.06
nott	Nottingham University, UK	800	34,265	1,550	23	0.03
pur	Purdue University, Indiana	2,419	120,690	5,000	30	0.03

The combined super event is treated as having its size equal to the sum of its component events. The process, however, is a little more complicated when room allocation is a part of the problem. Unfortunately only one of the data sets used for testing (the University of Nottingham data) includes information on rooms. In this case the super event preserves the room allocation of its component events, which cannot then be altered.

IV. RESULTS

A. Experimental Data

To evaluate the effectiveness of this approach, several real enrollment data sets were used for testing. Table I lists the data used together with the characteristics of each data set such as the total number of exams and the total number of student exam enrollments. The density of the conflict matrix is calculated as the average number of other exams that each exam conflicts with divided by the total number of exams. For example, a conflict density of 0.5 indicates that each exam conflicts with half of the other exams on average. As most of this data does not include data on rooms available, a simple upper limit on the number of seats available each period is used instead. The problems range from the smaller but more densely conflicting kfu and carf92 problems to the fairly large but relatively sparsely conflicting nott data set and the huge pur data set.[1]

For all of the problems, a typical timetable layout was used that was comprised of three periods per day from Monday to Friday and one period on Saturday morning. There were no periods on Sunday. This layout was repeated until the given number of periods for the problem was reached.

The primary task of the algorithm was scheduling all events within the given number of periods, and the secondary task was minimizing the number of back-to-back conflicts with adjacent periods, with adjacent conflicts on the same day being penalized more than those occurring overnight. For the purposes of solution evaluation and quality the function as shown in (1) was used.

B. Results

The method was tested on all the above data sets with subset sizes of 50, 100, 250, and 500 events both with and without a look-ahead approach. Each of these configurations was tested with all of the given heuristics five times each, and the average result given. This small sample size is sufficient here to offer some reasonable observations regarding the algorithm's performance.

Tables II–IV show the results achieved when using the relevant heuristics. The value given in the time column is the execution time in CPU seconds on a DEC Alpha computer for that particular run. Where the number of exams implies that the last subset of a run is incomplete the final phase of the process is run with a reduced subset size. The breakdown of results for the best of the five runs is given as the number of unscheduled exams and the violations of the second order conflicts. The results are also shown graphically in Figs. 5–12.

Looking at the results when using largest degree, shown in Table II, we see quite a mixed picture. On the carf92 problem the best result is obtained with a subset size of 50 and a look-ahead approach, taking less than one tenth of the

[1]All of these data sets can be obtained over the Internet: ftp://ftp.cs.nott.ac.uk/ttp/Data and ftp://ie.utoronto.ca/mwc/testprob

TABLE II
RESULTS OF APPLYING THE MULTISTAGE METHOD WHEN USING LARGEST DEGREE TO SEPARATE THE PROBLEMS

Data	Subset Size	Look Ahead	Penalty	Time	Results from Best		
					Unscheduled exams	2nd Order (Same Day)	2nd Order (Overnight)
carf92	50	no	2577	42	0	524	849
		yes	1822	173	0	313	766
	100	no	2104	119	0	469	614
		yes	2123	547	0	375	837
	250	no	3493	510	0	460	938
		yes	9476	2149	1	777	898
	500	no	12078	1937	1	777	900
		yes	12167	1942	1	812	837
kfu	50	no	2060	30	0	401	707
		yes	1608	105	0	228	704
	100	no	1811	73	0	222	838
		yes	2525	265	0	368	1008
	250	no	2708	345	0	683	839
		yes	2823	762	0	541	836
nott	50	no	31097	31	5	173	450
		yes	4762	219	0	128	350
	100	no	14984	83	0	155	416
		yes	15227	323	0	71	285
	250	no	5833	340	0	99	297
		yes	736	1434	0	94	316
	500	no	766	1265	0	121	397
		yes	1168	2551	0	216	358
pur	50	no	197635	1463	32	3385	3068
		yes	117730	1802	19	3401	3318
	100	no	139741	1097	21	3477	3473
		yes	86558	3097	13	2479	3100
	250	no	93110	2786	15	2765	2907
		yes	155096	15630	28	1939	2522
	500	no	155557	11331	28	2191	2541
		yes	219371	41422	39	1581	2370

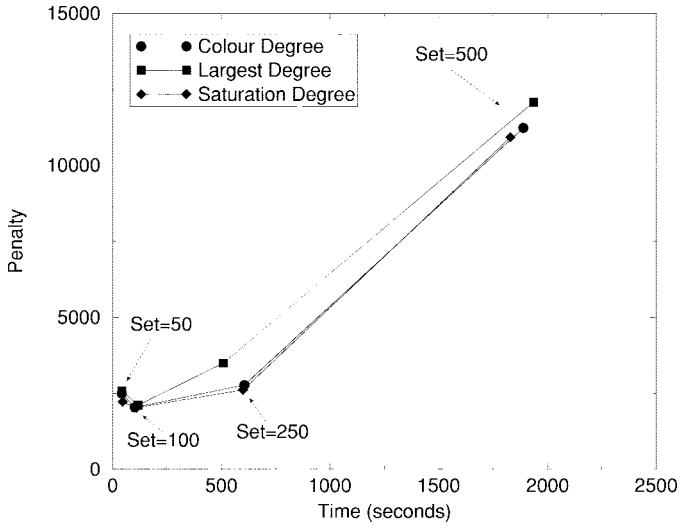

Fig. 5. Results when run on the carf92 problem without a look ahead.

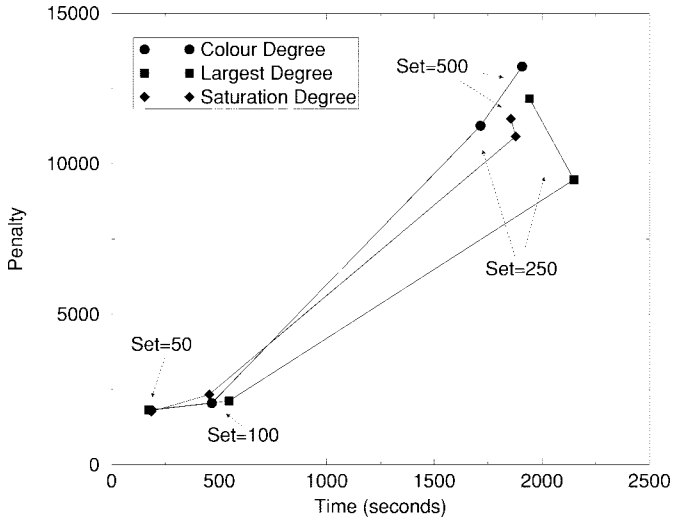

Fig. 6. Results when run on the carf92 problem with a look ahead.

TABLE III
RESULTS OF APPLYING THE MULTISTAGE METHOD WHEN USING COLOR DEGREE TO SEPARATE THE PROBLEMS

Data	Subset Size	Look Ahead	Penalty	Time	Results from Best		
					Unscheduled exams	2nd Order (Same Day)	2nd Order (Overnight)
carf92	50	no	2495	42	0	515	884
		yes	1812	184	0	302	804
	100	no	2034	102	0	359	773
		yes	2043	467	0	376	700
	250	no	2766	607	0	501	847
		yes	11260	1715	0	1250	1056
	500	no	11232	1889	0	1244	1050
		yes	13236	1908	2	673	803
kfu	50	no	2942	26	0	359	728
		yes	1627	99	0	250	802
	100	no	1836	61	0	260	999
		yes	2355	254	0	464	724
	250	no	2755	333	0	431	910
		yes	3077	878	0	468	1258
nott	50	no	50897	31	1	259	520
		yes	15923	138	0	77	396
	100	no	15621	102	1	106	330
		yes	544	467	0	67	289
	250	no	5602	607	0	100	255
		yes	759	1715	0	128	293
	500	no	757	1889	0	131	322
		yes	1061	1908	0	208	398
pur	50	no	198097	1426	28	3579	2917
		yes	122499	1830	20	4667	3209
	100	no	166370	1203	22	3958	3212
		yes	84846	2661	11	2577	3068
	250	no	105955	2518	17	2566	3079
		yes	163145	13710	22	1987	3167
	500	no	163405	9616	27	1957	2591
		yes	212961	38394	41	1976	2033

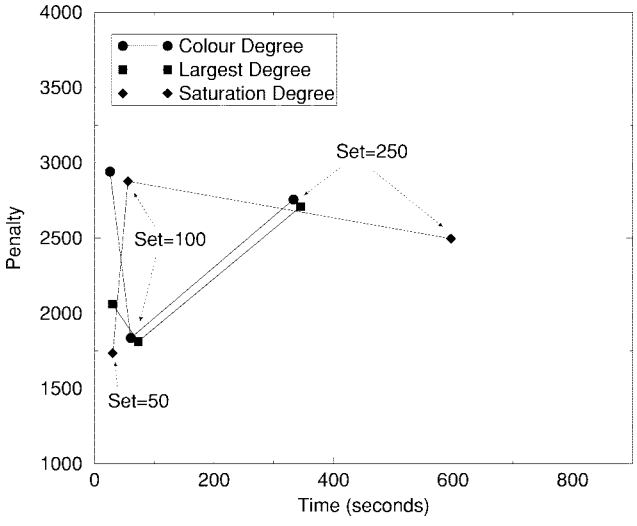

Fig. 7. Results when run on the kfu problem without a look ahead.

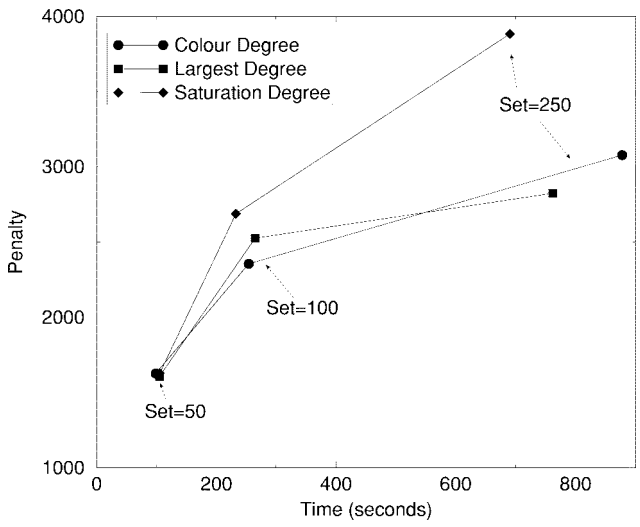

Fig. 8. Results when run on the kfu problem with a look ahead.

TABLE IV
RESULTS OF APPLYING THE MULTISTAGE METHOD WHEN USING SATURATION DEGREE TO SEPARATE THE PROBLEMS

Data	Subset Size	Look Ahead	Penalty	Time	Results from Best		
					Unscheduled exams	2nd Order (Same Day)	2nd Order (Overnight)
carf92	50	no	2211	47	0	433	792
		yes	1765	186	0	363	576
	100	no	2023	109	0	393	768
		yes	2328	455	0	433	783
	250	no	2613	600	0	463	931
		yes	10910	1878	1	694	917
	500	no	10917	1830	1	697	911
		yes	11495	1857	1	693	841
kfu	50	no	1735	31	0	228	786
		yes	1626	105	0	307	589
	100	no	2878	56	0	372	636
		yes	2688	233	0	471	890
	250	no	2495	596	0	468	794
		yes	3883	690	0	455	1078
nott	50	no	907	41	0	133	420
		yes	552	156	0	65	324
	100	no	889	78	0	127	320
		yes	524	340	0	76	282
	250	no	639	336	0	80	316
		yes	771	1120	0	130	328
	500	no	754	954	0	128	326
		yes	1076	2700	0	147	404
pur	50	no	130228	995	17	3116	3562
		yes	80323	1522	10	3514	3282
	100	no	108248	904	14	3835	3089
		yes	65461	2850	9	2890	3576
	250	no	107556	2655	16	2679	2643
		yes	160292	16411	28	1972	2294
	500	no	161776	10240	28	2057	2480
		yes	218192	26193	38	1727	2118

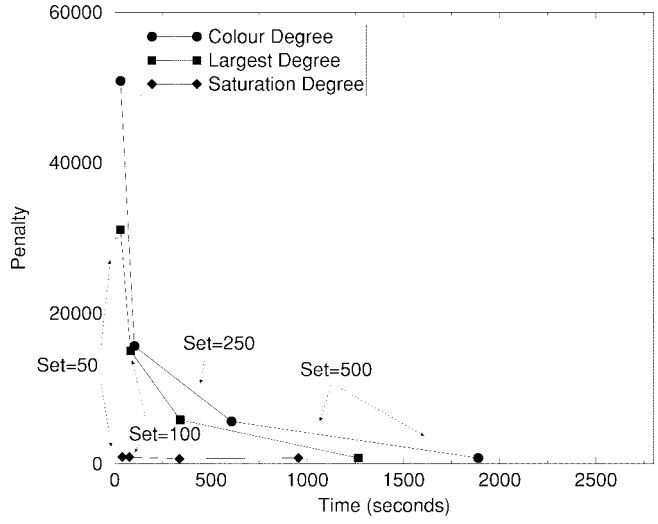

Fig. 9. Results when run on the nott problem without a look ahead.

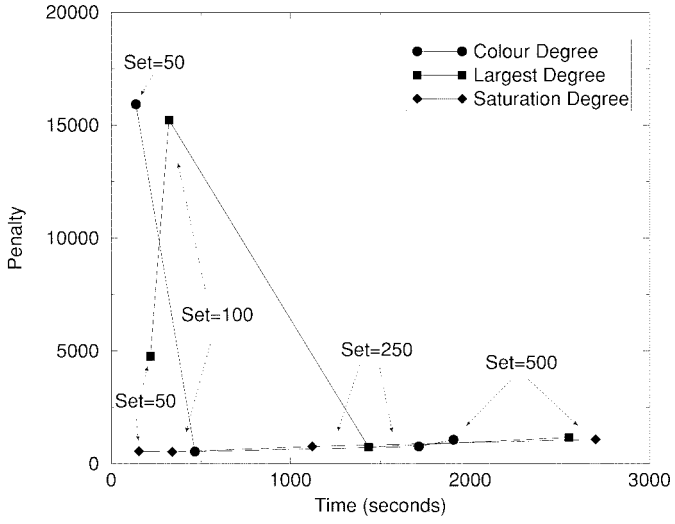

Fig. 10. Results when run on the nott problem with a look ahead.

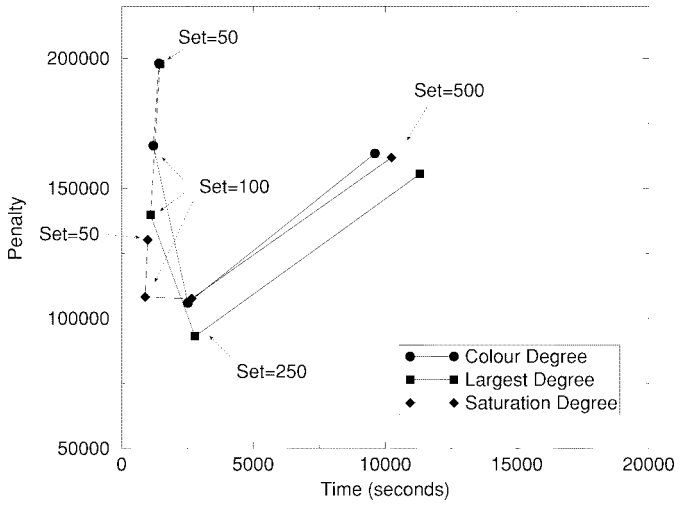

Fig. 11. Results when run on the pur problem without a look ahead.

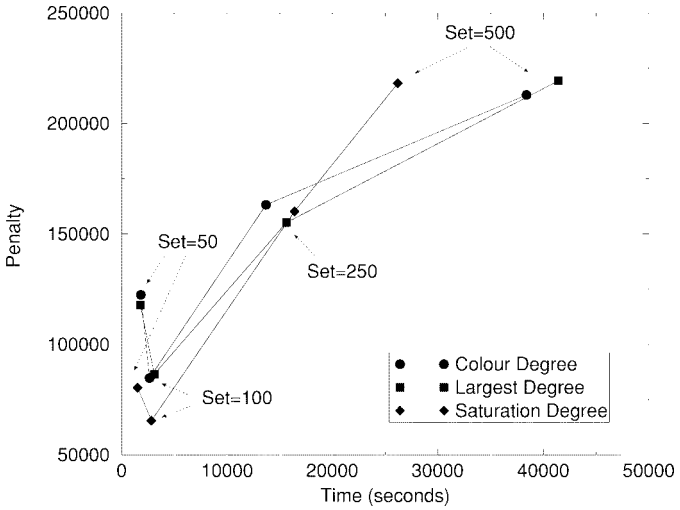

Fig. 12. Results when run on the pur problem with a look ahead.

amount of time to execute, with the larger subset sizes failing to find feasible solutions. Similarly for the kfu problem the best result is also found with the subset size of 50 and a look-ahead approach, but the difference with the higher subset sizes is not quite as great. The situation with the nott data is quite different. In this case, only the larger subset sizes are able to consistently find good quality timetables. This is most probably due to high room utilization (nearly 100%) and the low density of the problem, making it more of a bin-packing problem than a graph coloring one and therefore lessening the effects of the heuristic. When considering the results for the pur problem we must take into account the fact that "two in one [first order] conflicts are unavoidable" [9]. This means that some conflicting exams will have to be scheduled in the same period. This situation (in practice) involves quarantining students until they can sit for an exam after the main sitting. So the aim is, therefore, to reduce the amount of infeasibility in the timetable rather than to find fully feasible timetables. Bearing this in mind it is clear that a larger subset size of 100 with a look-ahead approach finds the best solution. We might have expected this subset size to be larger given the huge number of events but this does of course depend on the structure of the problem.

Considering the results when using color degree we see little difference in the individual results when compared to largest degree apart from a few minor variations in solution quality. If we instead compare with saturation degree, however, we can discern a notable improvement. While there is no substantial variation for the kfu and carf92 problem, it is clear that the approach works completely differently on the nott data set where the employment of saturation degree results in the algorithm consistently finding feasible solutions even with very small subset sizes. A probable reason for this is that the nott data set has variable length exams and periods resulting in fewer available time-slots for longer exams. This is something that neither color degree or largest degree consider but saturation degree does. For this problem, subset sizes of 50 and 100, both with look-ahead approaches, produce the best quality solutions. While the general trend for the pur problem remains roughly the same, there is a noticeable increase in quality for smaller subset sizes, indicating that saturation degree is also a much better heuristic for this problem.

Taking all the results into account it would seem that least saturation degree first is the most reliable heuristic to use for this method. Though other choices of heuristic may be more suitable for other problems it is notable that least saturation degree first outperforms the other heuristics on the nott problem, which is more constrained in terms of seats and less dense in terms of its conflict matrix, while still achieving equivalent performance on the more densely conflicting problems. In terms of which subset size to use, a size of 50 seems to be appropriate for smaller problems and a size of 100 for larger problems. In both cases a look-ahead approach should be used. Using these sizes and given a suitable heuristic, it is possible to produce substantially better results in a fraction of the time than if the memetic algorithm were applied.

For the purposes of comparison with a more established method, Table V shows the results when using a heuristic backtracking method similar to that described by Carter [9] which is well established and has been implemented at several universities. These results represent the lowest penalty found for these data sets by a published algorithm. The heuristic used here was least saturation degree first, which was found to be the most effective heuristic in [9]. This also provides the best comparison as saturation degree certainly appears to be the most effective heuristic for our method. This method requires very little run time. Even the very large pur problem requires no longer than a few minutes. Comparing the results in Table V with our results when using saturation degree we see a quite uniform reduction in penalty of roughly 40%.

V. Conclusions

The application of an algorithm to a problem in phases can drastically reduce the amount of time taken to find that solution (relative to the time taken to apply an algorithm to the entire problem) and also considerably improve the quality of that solution. In essence the method offered is a hybrid of heuristic sequencing and evolutionary methods, which (as we

TABLE V
RESULTS OF APPLYING HEURISTIC BACKTRACKING BASED ON SATURATION DEGREE TO THE SAME DATA

Data	Penalty
carf92	2915
kfu	2700
nott	918
pur	97521

have shown) can outperform either method on its own. This hybrid improves on its components by utilizing knowledge of the problem to order events according to expected difficulty but instead of taking the single most difficult event and perhaps placing it in the best available period, it takes a number of the most difficult events and applies the evolutionary algorithm EA to find the best placements with respect to each other, as well as with those events already scheduled.

In attempting to determine the best subset size to use with this method, there appears to be an optimal size that is perhaps related to the number of events in the problem, though there may be other considerations. For instance when using saturation degree the smaller kfu and carf92 problems seem to work best with the smaller subset size of 50 with a look-ahead approach, the larger nott data set produces roughly equivalent results at sizes of 50 and 100, both with look-ahead approaches, while a subset size of 100 with a look-ahead approach produces the best results on the pur data set. As mentioned earlier we might have expected the optimal subset size for the pur data set to be higher due to the huge nature of the problem. This could be due to the evolutionary algorithm perhaps having an upper limit on the amount of data it can efficiently handle (which could perhaps be overcome with larger population sizes, though this would inevitably lead to even higher run times) or maybe because of the low density of the conflict matrix. Throughout the trials, however, the use of a look-ahead approach shows obvious benefits. Although there is substantial extra time involved in doing this, it is not quite so important when we use subset sizes that are relatively small such as those which seem to produce the best results.

Even with a good choice of heuristic, and an optimal subset size with a look ahead, it is possible that on some more difficult problems placements made earlier in the process could well prevent some events being scheduled later in a valid period. To prevent this from happening we could borrow another idea from heuristic sequencing methods and allow a backtracking operator to be applied at the end of a phase if there are any events still unscheduled. Such an operator would have to be allowed to move events that have previously been fixed to be effective. This would inevitably degrade the quality of the timetable to some degree but should mean that the process will have as much chance of finding a feasible timetable as any other method.

This approach could possibly be adapted to other scheduling problems, especially those where some sort of heuristic sequencing approach exists. Early work is being carried out in the Automated Scheduling and Planning group at the University of Nottingham into employing a similar approach for power maintenance scheduling problems.

REFERENCES

[1] E. K. Burke, D. G. Elliman, P. H. Ford, and R. F. Weare, "Examination timetabling in british universities a survey," in *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference* (Lecture Notes in Computer Science 1153), E. Burke and P. Ross, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 76–90.

[2] E. K. Burke, D. G. Elliman, and R. F. Weare, "A hybrid genetic algorithm for highly constrained timetabling problems," in *Genetic Algorithms: Proceedings of the 6th International Conference*, L. J. Eshelman, Ed. San Francisco, CA: Morgan Kaufmann, 1995, pp. 605–610.

[3] E. K. Burke and J. P. Newall, "Investigating the benefits of utilising problem specific heuristics within a memetic timetabling algorithm," Dept. Comput. Sci., Univ. Nottingham, UK, Working Paper NOTTCS-TR-97-6, 1997.

[4] E. K. Burke, J. P. Newall, and R. F. Weare, "A memetic algorithm for university exam timetabling," in *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference* (Lecture Notes in Computer Science 1153), E. Burke and P. Ross, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 241–250.

[5] M. W. Carter, "A decomposition algorithm for practical timetabling problems," Dept. Industrial Eng., Univ. Toronto, Working Paper 83-06,Apr. 1983.

[6] ———, "A survey of practical applications of examination timetabling," *Oper. Res.*, vol. 34, pp. 193–202, 1986.

[7] M. W. Carter and G. Laporte, "Recent developments in practical examination timetabling," in *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference* (Lecture Notes in Computer Science 1153), E. Burke and P. Ross, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 3–21.

[8] M. W. Carter, G. Laporte, and J. W. Chinneck, "A general examination scheduling system," *Interfaces*, vol. 11, pp. 109–120, 1994.

[9] M. W. Carter, G. Laporte, and S. Y. Lee, "Examination timetabling: Algorithmic strategies and applications," Dept. Industrial Eng., Univ. Toronto, Working Paper 94-03, Jan. 1995..

[10] D. Corne, P. Ross, and H. Fang, "Fast practical evolutionary timetabling," in *Lecture Notes in Computer Science 865 (AISB Workshop on Evolutionary Computing)*, T. C. Fogarty, Ed.. Berlin: Springer-Verlag, 1994, pp. 250–263.

[11] L. Davis, Ed., *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[12] R. Dawkins, *The Selfish Gene*. London, U.K.: Oxford Univ. Press, 1976.

[13] E. Foxley and K. Lockyer, "The construction of examination timetables by computer," *Comput. J.*, vol. 11, pp. 264–268, 1968.

[14] A. Hertz, "Tabu search for large scale timetabling problems," *Eur. J. Oper. Res.*, vol. 54, pp. 39–47, 1991.

[15] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York: Plenum, 1972, pp. 85–103.

[16] P. Moscato and M. G. Norman, "A 'memetic' approach for the travelling salesman problem implementation of computational ecology for combinatorial optimization on message-passing systems," in *Proc. Int. Conf. Parallel Computing and Transputer Applications*. Amsterdam: IOS Press, 1991.

[17] B. Paechter, A. Cumming, and H. Luchian, "The use of local search suggestion lists for improving the solution of timetable problems with evolutionary algorithms," in *Lecture Notes in Computer Science 993 (AISB Workshop on Evolutionary Computing)*, T. C. Fogarty, Ed. Berlin: Springer-Verlag, 1995, pp. 86–93.

[18] P. Ross, D. Corne, and H.-L. Fang, "Improving evolutionary timetabling with delta evaluation and directed mutation," in *Parallel Problem Solving in Nature*, vol. III, Y. Davidor, H-P. Schwefel, and R. Manner, Eds. Berlin: Springer-Verlag, 1994.

[19] J. M. Thompson and K. A. Dowsland, "General cooling schedules for a simulated annealing based timetabling system," in *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference* (Lecture Notes in Computer Science 1153), P. Ross and E. Burke, Eds. Berlin: Springer-Verlag, 1996, pp. 345–364.

[20] R. F. Weare, "Automated examination timetabling," Ph.D. dissertation, Univ. Nottingham, Dept. Comput. Sci., 1995.

[21] D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetable problems," *Comput. J.*, vol. 10, pp. 85–86, 1967.