A Multiobjective Framework for Heavily Constrained Examination Timetabling Problems

Christine L. Mumford
School of Computer Science,
Cardiff University, CF24
3AA, United Kingdom
Email: C.L.Mumford@cs.cardiff.ac.uk

December 22, 2010

Abstract

University examination timetabling is a challenging set partitioning problem that comes in many variations, and real world applications usually carry multiple constraints and require the simultaneous optimization of several (often conflicting) objectives. This paper presents a multiobjective framework capable of solving heavily constrained timetabling problems. In this prototype study, we focus on the two objectives: minimizing timetable length while simultaneously optimizing the spread of examinations for individual students. Candidate solutions are presented to a multiobjective memetic algorithm as orderings of examinations, and a greedy algorithm is used to construct violation free timetables from permutation sequences of exams. The role of the mulitobjective algorithm is to iteratively improve a population of orderings, with respect to the given objectives, using various mutation and reordering heuristics.

1 Introduction

In its most basic form, the university examination timetabling problem (UETP) involves scheduling a set of exams into the minimum number of time slots, so that there are no clashes; i.e., no student is required to take more than one examination at any one time. This version of the UETP is identical to the well-known graph coloring problem, with time slots allocated to examinations instead of colors assigned to vertices. However, the graph coloring model represents a simplistic view and limitations in resources in the 'real world', invariably impose a number of additional constraints on timetable planners in universities throughout the world. Examples include:

- Time limitations - exams are constrained within a maximum time period, to avoid overlapping with teaching or vacations.
- Capacity constraints - due to room size limitations.
- Restricted sessions - perhaps certain exams have to fit in with the needs of part time students.
- Restricted resources - some exams may need special resources in limited supply, (e.g., a fully equipped laboratory).
- Interdependencies among examinations - Perhaps exam A has to occur before exam B, for example.

A survey of British universities carried out by Burke *et al* in 1996 [7] identified a total of 32 constraints encountered by those completing the questionnaires. Nevertheless, most

state-of-the-art approaches published in the literature consider only one or two of them, and publicly available benchmarks data sets lack the variety and complexity challenging real institutions. On the other hand, those researchers who choose to focus on the complex situations encountered in real schools, tend to produce bespoke solutions, and their techniques cannot generally be compared with others. Thus, as pointed out in [7], there would appear to be a very large gap between 'theory and practice', and a need for greater generality in dealing with heavily constrained problems.

In addition to *hard* constraints, which must be adhered to, various less critical considerations, or *soft* constraints can be identified as desirable. Some examples are listed below:

- Spreading examinations as widely as possible for individual students, to give them revision breaks.
- Scheduling exams with the most candidates as early as possible, to give sufficient time for marking.
- Avoid scheduling exams of different durations in the same room.

Soft constraints are most naturally modeled as objectives, and the aim is to minimize their effect, rather than to eliminate them entirely.

In this research we represent our candidate solutions as permutations of examinations, and use a greedy algorithm as a 'decoder' to allocate time slots to the exams in a sequential fashion. Using this sequential technique it is possible to check all of the constraints at each stage of the timetable construction process and (for the most part[1]) avoid assignments that would cause violations. Furthermore, it is an easy matter to tailor this approach to situations pertaining on the ground, by adding or removing constraint checks, as appropriate, thus increasing the versatility and generality of the technique. On the other hand, many approaches in the literature rely on a 'direct representation', in which timetables are stored as vectors, with each vector element representing an exam, and each element value representing a time slot. Feasible timetables can be challenging to find when multiple constraints are applied, if a direct representation is used. Nevertheless, to guarantee feasible solutions, sequential methods require an inexhaustible supply of time slots to be available for greedy assignment, and this cannot be sustained in practice. For this reason, if sequential methods are to be competitive, it is essential that they are effective in reducing overlong timetables so that they fit into a given academic calender. Fortunately, the multiobjective algorithm presented in the present paper appears to be capable in this respect. A summary of the key components of the present approach is listed below:

1. *A simple multiobjective framework* for the UETP to simultaneously reduce timetable lengths and produce a favorable 'spread' of exams for the students.
2. *A greedy algorithm* that sequentially allocates time slots to examinations while simultaneously considering two objectives, and checking multiple constraints.
3. *Grouping and reordering techniques* for iterative improvement, based on the iterated greedy algorithm of Culberson and Luo, [19], but tailored towards the UETP.
4. Two alternative *performance measures* for assessing timetable length: the first is applied in the early stages of the multiobjective evolutionary algorithm (MOEA) to encourage shorter timetables; and the second is used in the later on to give more flexibility for the second objective, when it is clear that further reductions in timetable length are unlikely.

[1] A few constraints, such as restricted sessions, are sometimes more difficult to accommodate. These will be discussed later in the paper.

5. *Two mutations* - insertion and Kempe chain exchange.

6. *A local search* (based on Kempe exchanges), applied at the very end to the best solutions produced by the MOEA.

The rest of the paper is structured as follows. Sections 2 and 3 give some background and cover related work. Section 4 then describes the various components of the current approach. This is followed by Section 5, which gives details of the data sets used in the study. Section 6 next gives details of how the hard constraints are dealt with in the multiobjective approach, and Sections 7 and 8 present some comparative studies to assess the abilities of three greedy algorithms to construct good timetables in a multiobjective setting. The main results for the uncapacitated, capacitated and restricted sessions versions of the UETP are presented in Section 9. Finally, Section 10 summarizes the main conclusions of the paper.

2 Single Objective Approaches the UETP

There is a vast literature devoted to timetabling including a bi-annual conference, PATAT (International Conference on the Practice and Theory of Automated Timetabling) which covers this topic exclusively. The interested reader is referred to these conference proceedings, and survey papers such as [11, 13, 15, 29, 30] for more background on the subject. This Section will provide a brief overview of the main single objective techniques that appear in the literature, and the following Section will focus on multiobjective techniques. The classification used here is based on [29], however many successful methodologies overlap categories or combine strategies from different paradigms.

Early research on timetabling focussed on simple sequential methods based on graph coloring techniques [34]. In essence, sequential methods work through a list of examinations, assigning the earliest feasible time slots to each exam in turn, using a simple greedy algorithm. A large number of ordering strategies have been devised, aimed at presenting favorable sequences to the greedy assignment algorithm, generally attempting to schedule the most 'difficult' or heavily constrained exams first, for example, [21, 24, 34]. However, the performance of these techniques tends to be variable and somewhat unpredictable, and the most successful implementations rely on some kind of additional iterative improvement mechanism, on top of the basic construction scheme. Carter and his colleagues, for example, make extensive use of backtracking [14], to attempt to correct poor assignments. In another approach, by Burke and Newall [10], an heuristic modifier is utilized in an iterative framework, to adjust a 'difficulty' score for 'problematic' examinations, so that they are scheduled earlier the next time the ordering heuristic is applied. Examples of other schemes that can be found in the literature for the iterative improvement of orderings include: a fuzzy system that brings together three ordering criteria [3], and a hyper-heuristic algorithm used to select appropriate low-level ordering heuristics, each to sequence just a few of the events for the timetable [9]. Sequential methods play a significant role in the present work.

Another popular approach is to use constraint based techniques. There are two main approaches: constraint logic programming (CLP) [2] and constraint satisfaction problem (CSP) techniques [32]. These techniques operate on exams modeled as a set of discrete variables with finite domains, and a set of constraints between the variables specify which combinations of values are allowed and which are not. Optimization is based on a form of branch and bound. Used on their own, constraint based techniques tend to be computationally expensive, and the most successful approaches combine constraint based techniques with other methods. Most notable is the work by Merlot *et al* [25]. They used constraint programming to obtain initial solutions, then improved them with simulated annealing and hillclimbing. In this way they reported some of the best results (at the time) for some literature benchmarks, including several instances from the well known Toronto data sets [14]. Although other researchers have reported successful implementations using constraint

based methods and hybrid techniques, their work is mostly confined to specific instances at particular institutions, making comparisons with other approaches difficult.

The last decade has seen many studies involving local search and metaheuristics. Of particular note is a local search heuristic, known as a Kempe chain interchange[2], which attempts to improve a timetable by swapping groups of exams between pairs of time slots, where this can be done without violating any hard constraints. This technique was developed for the UETP by Thompson and Dowsland [31] and has been used subsequently by many researchers including Merlot *et al* [25] in their previous mentioned hybrid approach (also see [16, 18]). Abdullah *et al* [1] developed a technique based on larger neighbourhoods, instead of using the usual pairwise exchanges. Best results were obtained on some Toronto benchmark instances using this approach (see Table 3).

3 Multiobjective Approaches

Multiobjective optimization techniques are becoming increasingly popular methods for solving a range of real world problems where consideration of more than one optimization criterion is required. Solutions to problems involving multiple objectives are characterized by optimum sets of alternative solutions, known as *Pareto sets*, rather than by a single global optimum. Pareto-optimal solutions are *non-dominated solutions* in the sense that it is not possible to improve the value of any one of the objectives, in such a solution, without simultaneously degrading the quality of one or more of the other objectives in the solution vector. (The reader is referred to [22] for a more general treatment of multiobjective techniques.)

It has been recently argued by Cheong *et al* [17], that the UETP is essentially a multiobjective optimization problem. Treating it as such certainly avoids the need to determine the length of the timetable in advance, and 'freeing up' this constraint can allow the production of a range of alternative timetables from which the user can select the one that suits his/her purposes the best. Nevertheless, as far as the present author is aware, very few studies have been undertaken to date involving multiobjective optimization applied to the UETP.

A multi-criteria approach to timetabling involving nine criteria was published by Burke, Bykov and Petrovic in 2001 [4]. In this study several hard constraints were modeled as soft constraints, and an heuristic search used to minimize the extent to which the various constraints were violated. More recently Côté, Wong and Sabourin [18, 35], Cheong, Tan and Veeravalli [17], and the present author, [28], have tackled timetabling as a dual objective optimization problem, simultaneously minimizing timetable length and reducing proximity costs (i.e., maximizing the spread of exams for individual students).

Côté, Wong and Sabourin cover both uncapacitated [18] and capacitated (with seating capacity constraints) [35] versions of the dual objective UETP in their research. Their method uses a multiobjective evolutionary algorithm (MOEA) with a main population and an archive, and two local searches (TABU search algorithms) plus mutation and selection. A direct representation is used in their research, which means that their algorithm manipulates the time slot values for each exam. A local search algorithm is then used to reduce conflicts and repair infeasibility. The results obtained in [18] for the uncapacitated problem are used for benchmarking the present study. However similar comparisons with their results for the capacitated problem have not been possible, due to shorter timetables being obtained in the present study.

In another recent study of the bi-objective optimization of the capacitated UETP, Cheong, Tan and Veeravalli [17] employ an innovative crossover they call *day-exchange* crossover. In their study, temporal relationships within the timetable are stored explicitly in the chromosome, facilitating meaningful exchanges between parents. Unfortunately, once again, the timetables produced in this earlier study tend to be longer than those produced in the present paper.

[2]First used by Alfred B. Kempe in 1879 in a 'proof' of the four-color conjecture.

An additional factor making comparative studies very difficult for the capacitated problem is that the various studies have not all used the same objective function to measure the proximity costs. Inevitably, various authors, over the last decade or so, have come up with different ways to assess the spread of examinations, depending on their views as to what constitutes a 'good spread' of exams (see [29] for a summary). Many recent studies (including the present paper and also [18] and [28]) favor a measure introduced by Laporte and Desroches in [23] and used by Carter *et al* in [14]. This measure is simple and is based on how many free periods students have between their exams (see Section 4.5.2 for details). On the other hand, measures introduced in [5] and [6] are based on very realistic assumptions and model timetables with three sessions a day and one on a Saturday morning. Unfortunately, the present author has found a general lack of clarity and consistency in many research papers that use proximity costs calculated from week day /weekend scenarios. For this reason comparative studies have been restricted to results quoting the proximity measure of Laporte and Desroches.

4 Details of the New Multiobjective Framework

We will start with an overview of the new approach. The *multiobjective framework* operates in four phases (MOO is an abbreviation for multiobjective optimization):

1. *Initialization*
2. *MOO 1*: Optimization of dual objectives
3. *MOO 2*: Focussing on proximity costs (i.e., improving the 'spread' of exams)
4. *Local search*

Following *Initialization* in Phase 1, the second Phase, *MOO 1*, operates to simultaneously reduce timetable length (objective 1) and proximity costs (objective 2). The purpose of Phase 3, *MOO 2*, is to focus almost entirely on further reducing proximity costs, following a stagnation in the attempts to reduce the timetable lengths any further. The transition between Phase 2 and Phase 3 is accompanied by a change in the performance measure for timetable length. In Phase 2 we use a measure that favors uneven distributions of exams between the time slots, in the hope that time slots with few exams will be 'squeezed out' to make the timetables shorter. Once a good range of trade-off solutions for timetable length have been obtained in this way, however, we find that there remains much scope for improvement in the proximity costs, if movement between time slots can be freed up. For this reason we use timetable length explicitly as our performance measure in Phase 3. The final component of the multiobjective approach is the *local search*, which is only applied to the 'best' (non-dominated) individuals produced in *MOO 2*. For this stage a hillclimbing algorithm is used, based on Kempe chain interchanges (explained later). The hillclimber is useful in making small improvements at the end, but it is time consuming.

Having first taken a 'top down' approach to introduce the structure of the multiobjective framework, we shall now view the techniques we use within this framework from a 'bottom up' perspective, and concentrate initially on the detail of the sequential and greedy techniques, which are the fundamental agents of change in the MOEA (stages *MOO 1* and *MOO 2*).

4.1 The Grouping and Reordering Heuristics

One category of sequential techniques that have not so far been widely used for timetabling, are the grouping and reordering heuristics of Culberson and Luo (CL) [19]. Like most other sequential techniques they were originally developed for the graph coloring problem. The

techniques are simple to apply and operate by iteratively altering a sequence of nodes (exams) presented to a simple greedy algorithm, so that fewer colors are needed (or shorter timetables obtained). Focussing their attention on sorting and reordering whole color classes (i.e., groups of nodes assigned the same color), these techniques are capable of producing excellent results. Previous studies by the present author have used the CL heuristics as a preprocessor for new order-based crossovers (i.e., crossovers that work on orderings of exams) for an evolutionary algorithm, as well as for local search operators in their own right, [27, 28]. Of particular significance to this type of work is a rare property of the CL heuristics, which we shall call the 'non-deterioration' property:

The non-deterioration property
It is impossible to get a worse result by applying any of Culberson and Luo's reordering techniques to the graph coloring problem, and it is possible that a better result (using fewer colors) may be produced (see [19] for details).

When applied to the UETP, the CL heuristics associate a numerical measure with each time slot, such as the number of exams or the number of students, and this measure is used as a criterion for re-sequencing the permutation lists with exams grouped together in their time slots. For example, the time slot with the most exams may be selected first, and thus these exams will occupy the first section of a permutation list, followed by the exams in the second most popular time slot, etc.. Following a complete rearrangement of list, the greedy algorithm is reapplied, and it is at this stage that shorter timetables can sometimes arise. Interestingly, the CL heuristics can be applied equally effectively to the bin packing problem, and thus they are suitable for capacitated versions of UETP. Culberson and Luo suggest a random mix of various reordering heuristics and call the composite algorithm *iterated greedy, IG*. Given an initial permutation of vertices, the IG algorithm can be defined by the following repeating sequence:

1. greedy assignment
2. grouping of exams within their color classes (time slots)
3. reordering of complete color classes (time slots)

Figure 1 illustrates some key operations from a single iteration of IG applied to a small graph coloring instance with 12 vertices and 14 edges. Imagine that the nodes are examinations, and the colors time slots, and Figure 1 becomes a timetabling problem. Figure 1 (a) defines the small graph, and Figure 1 (b) gives a typical random permutation of the vertices from the graph and also the resulting greedy coloring. Figure 1 (c) shows the grouping operation used to sort the list in non-descending sequence of color label, and 1 (d) gives the arrangement following the application of one of the CL reordering heuristics called *largest first*. The largest first heuristic rearranges the color classes in non-ascending sequence of their size (i.e, the number of nodes or exams). Note that the positions of color classes 1 and 2 have been reversed in Figure 1 (d). This follows the advice in [19] to interchange positions of equal sized color classes. Figure 1 (e) shows the colors relabelled, with the lowest integer used to label members of the color class occurring first on the list. Finally, Figure 1 (f) illustrates the situation following a second application of the greedy algorithm. Note, that vertices 4 and 1 have each been assigned colors with lower integer labels, making some color classes smaller. It is likely that repetition of steps such as these will soon produce the optimum coloring of 3, on a small instance such as this.

Various numerical properties of the color classes were tried by Culberson and Luo as criteria for reordering:

1. **Reverse**: Reverse the order of the color classes

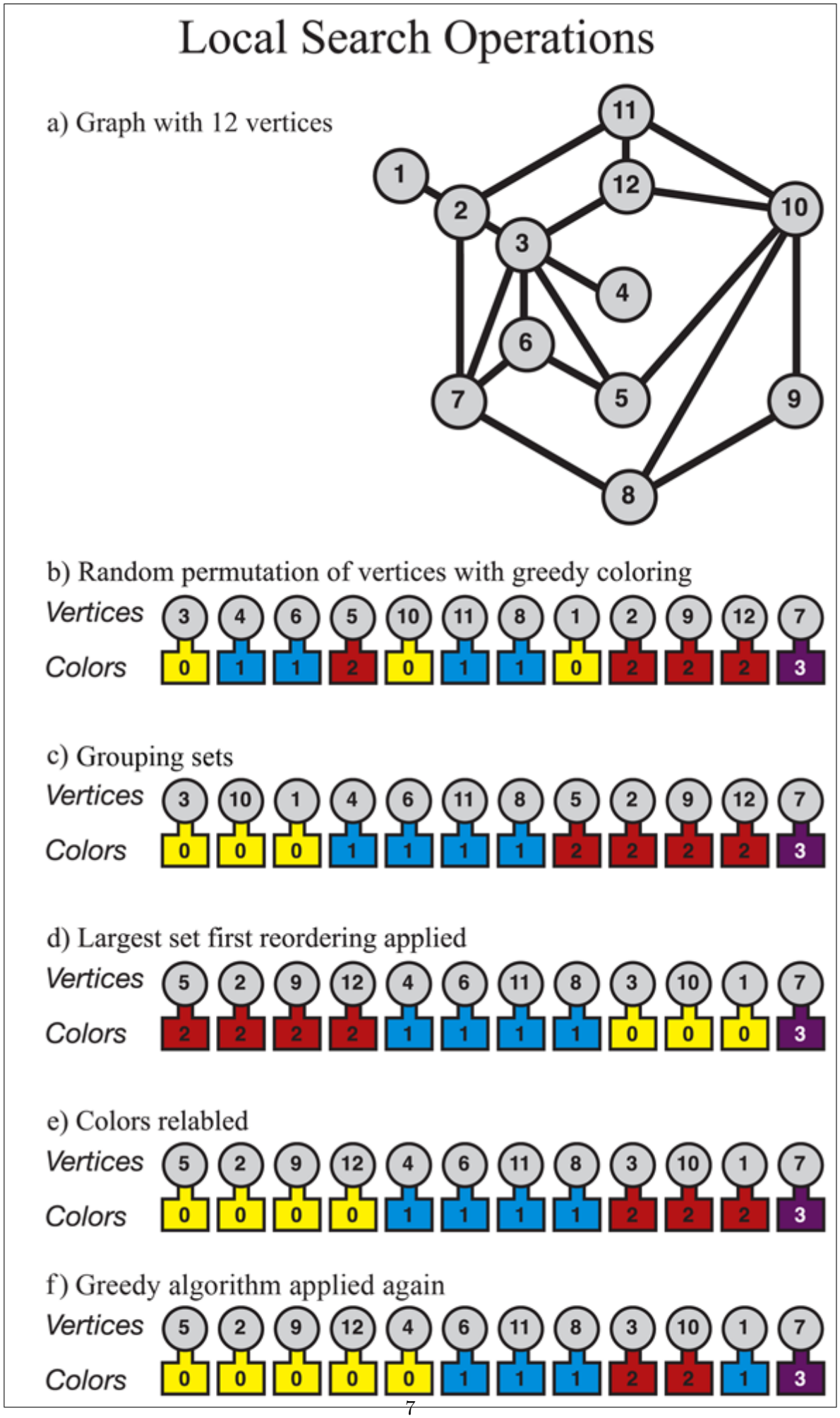


Figure 1: Various operations by Culberson and Luo, [19], used in the local search procedure

2. **Random**: Place the color classes in random order
3. **Largest first**: Place the classes in order of decreasing size (Figure 1 (d))
4. **Smallest first**: Place the smallest classes first
5. **Increasing total degree**: Place the classes in increasing order by the degree sum of the group
6. **Decreasing total degree**: Place the classes in decreasing order by the degree sum of the group

The favored combination of Culberson and Luo was: 'largest first', 'reverse' and 'random' used in the following ratio 50:50:30. We will use a slightly different regime, described later. But first, it is necessary to look in more detail at the greedy algorithms that can be used to decode the orderings to produce the actual timetables.

4.2 The Greedy Algorithms

Two new variations of greedy algorithm are presented and evaluated in this work, and the one that performed the best in tests is incorporated into the multiobjective framework.

Greedy heuristic construction approaches to timetabling problems can be classified according to whether or not the number of time slots is fixed in advance. In the absence of soft constraints, the role of a greedy algorithm is normally to attempt to convert an ordering of examinations into a timetable with no clashes, that is as short as possible. In this case each exam in turn will be allocated the earliest time slot that is available, without constraint violation. On the other hand, greedy allocation to optimize soft constraints, such as proximity costs, usually operates within a predetermined timetable length [14]. With the latter approach, the greedy algorithm will allocate to each exam the most favorable time slot in terms of its contribution to the overall proximity cost, all the while maintaining feasibility. Exams that cannot be allocated in this way within the predetermined number of time slots, however, will remain 'unallocated', unless suitable reassignments can be made by some additional procedure, such as backtracking.

At the other extreme, consider a greedy algorithm free to operate with no restrictions imposed on the timetable length. With ultimate freedom to produce timetables of any length, the greedy algorithm will be able produce conflict free timetables with no clashes and zero proximity costs. In other words, proximity cost can be treated as just another hard constraint. Although, it is likely that excessively long timetables will be produced if this scheme is used. [3]

The two new greedy algorithms presented here attempt a 'middle way': they simultaneously minimize timetable length and proximity costs, without the need to fix the number of time slots in advance. On the other hand, they do not go as far as eliminating proximity costs altogether. In essence, the new greedy algorithms work within an adaptive 'ceiling' for timetable length, and new assignments are restricted to time slots at or below the current ceiling level, provided no hard constraint is violated.

Three greedy variations of greedy algorithm are compared here. The first one is the 'standard' approach used for minimizing timetable length, whilst avoiding hard constraint violations. The second and third, however, attempt to reduce soft constraints, such as proximity costs, while at the same time avoiding hard constraint violation and minimizing the number of time slots in the timetable. Given the role of the greedy algorithms to 'decode' an ordering and produce a timetable, greedy algorithms are frequently described as 'greedy decoders' in the evolutionary computing literature. Hence, we shall use this term interchangeably with 'greedy algorithms' in what follows. The three decoders are described in detail below.

[3] This can be deduced from the poor results obtained by 'greedy 3' in a subsequent section. One can conjecture that timetable lengths producing zero proximity costs will be at least as poor as these.

Greedy 1 This is the 'standard' greedy algorithm as commonly used for graph coloring. For a given sequence of exams the algorithm starts at the beginning of the list and, working through each one in turn, it allocates the earliest available time slot so that no constraint is violated.

Greedy 2 This algorithm is similar to greedy 1, but allocation will not necessarily be limited to the earliest feasible time slot. Instead, the algorithm will try all feasible time slots that are currently 'allowed', and allocate the one that imposes the least proximity costs (or possibly the least effect of other soft constraints) on the partially constructed timetable. Time slots that are 'allowed' correspond to the those equal or below the current 'ceiling', which begins at zero, before the first exam is allocated a time slot. Allocations are allowed above the level of the current ceiling only to avoid violating hard constraints.

Under greedy 2, the first exam on the list will always be allocated time slot 1, and the 'ceiling' raised from '0' to '1'. The second exam will then allocated to time slot 1, provided no hard constraint is violated by doing so. Let us assume there is a clash between the first and the second exams, the second exam will then be allocated time slot 2, and the 'ceiling' raised from '1' to '2'. Suppose the third exam on the list does not clash with the first exam, the earliest feasible slot for exam 3 will then be time slot 1. Nevertheless, greedy 2 will go on to check whether time slot 2, if feasible, would be a more favorable assignment for exam 3. At this stage the choice for allocation is extremely limited. However, as the algorithm progresses along the list, more time slots tend to be needed to avoid hard constraint violation, giving more choice for future allocations. In this way the greedy decoder has some freedom to allocate exams to time slots favorable to proximity, yet the timetable length is nevertheless constrained.

Greedy 3 The greedy 3 algorithm begins by running greedy 1 on the ordered list of exams presented, simply to establish a lower bound on the timetable length, LB, for the particular ordering. The algorithm then starts afresh from position one of the ordering, proceeding in exactly the same way as greedy 2, but with the whole range of time slots, $\{1, 2, \ldots, LB\}$, *available from the start.* Additional time slots are introduced, $LB+1, LB+2, \ldots$, if required, to ensure a feasible timetable.

4.3 Adapting Reordering Heuristics to the UETP

Two difficulties are encountered with the iterated greedy grouping and reordering heuristics as suggested by [19] when applied to the UETP:

1. The CL grouping heuristic conserves time slot allocations for greedy 1 only, but not for greedy 2 or greedy 3.
2. Re-sequencing the time slot classes (for example using 'largest first') can radically alter the spread of exams for individual students, and this can have serious consequences for the proximity cost.

To illustrate 'problem 1', let us look at some outputs from a sample instance of the UETP consisting of just 9 exams. Each sample output lists the time slot assignments made to the exams in the sequence in which they appear in the ordering. The process begins with a random ordering of exams, then the following sequence of operations was performed:

1. Generate a random ordering of exams
2. greedy decoder
3. group into time slots

4. print assignments
5. greedy decoder
6. print assignments

Sample printed output (produced in lines 4 and 6 of the above sequence of operations) is given below for greedy 1 and greedy 3. However, disruption is difficult to observe on instances as small as this example (i.e., only 9 exams), so output for greedy 2 is omitted.

Time slot allocation using greedy 1

After grouping into time slots:

1 1 2 2 3 3 4 5 6, timetable length = 6

After second application of greedy 1

1 1 2 2 3 3 4 5 6, timetable length = 6

Note, with greedy 1 the time slot allocation of the exams does not change following the application of 'grouping'.

Time slot allocation using greedy 3

After grouping into time slots

1 1 2 3 4 5 5 6 6, timetable length = 6

After second application of greedy 3

1 1 ***6*** 3 ***5 2 4 7 7***, timetable length = 7

Time slot membership is severely disrupted by greedy 3 following 'grouping', and the timetable length has increased from 6 to 7, invalidating the **non-deterioration property**.

To guarantee **the non-deterioration property**, the reapplication of the greedy decoder following the 'grouping' process should not alter the assignments of exams to time slots in any way (only the reordering process should be capable of doing this). Given that our multiobjective evolutionary algorithm will rely heavily on the CL grouping heuristic, the prognosis for greedy 3 does not appear to be good, based on this example. Greedy 2 , on the other hand, shows rather more promise, as no disruption was observed on small instances and proved to be very slight, even on large instances.

We will now move on to the problems associated with re-sequencing the time slots. Once examinations have been allocated to time slots, re-sequencing entire time slots in the schedule will not introduce any new clashes, neither will it change the seating requirements. On the other hand, the consequences for individual students could be highly significant, with an entirely different 'spread' of exams being produced. It is worthy of note, however, that some criteria for re-sequencing are more disruptive than others. For example, a *random* reassortment of time slots could have dire consequences, yet reversing the time slot sequence will have no effect at all on the average proximity cost.

Recall that Culberson and Luo apply the following heuristics to reorder their color classes 'largest first', 'reverse' and 'random' in the ratio 50:50:30. For the present study

the 'largest first' heuristic will be replaced by an heuristic that measures how heavily constrained time slots are. The measure we use is called *decreasing total degree (DTD)*, and it will be described later. The 'random' ordering heuristic, was also abandoned in favor of an alternative which we will call 'deletion and insertion'. This heuristic simply selects a time slot (color), and deletes it from one part of the chromosome, reinserting it elsewhere at random, respecting class boundaries. Deletion and insertion is less disruptive to the proximity costs than randomly reordering all of the time slots. We will apply the heuristics *DTD*, 'reverse' and 'deletion and insertion' in the ratio 25:25:50. This mix appears to work well, although the scheme is fairly robust to changes.

4.4 The Multiobjective Framework

The general framework that we shall use for multiobjective optimization is outlined in Algorithm 1. The main ideas are adapted from the SEAMO algorithm (simple evolutionary algorithm for multiobjective optimization), [26, 33]. Recall the four phases outlined at the start of Section 4: *Initialization*, *MOO 1*, *MOO 2* and the *Local search*. Following Initialization, Algorithm 1 illustrates a simple steady-state approach for *MOO 1* and *MOO 2*, which sequentially selects every individual in the population for breeding. Once an individual is selected, the grouping, reordering and mutation operations are applied, as well as *greedy 2*, as specified in **applyOperations** (described later in Sections 8.1 and 8.2). Note: there is no crossover used in this study.

Algorithm 1 Simple Multiobjective Framework

Generate N random strings {*Phase 1: Initialization*, N is the population size}
Initialize $bestOB_1$ and $bestOB_2$ to arbitrarily large values
{$bestOB_1$ and $bestOB_2$ store the best-so-far values for OB_1 and OB_2}
for all strings in the population **do**
    Apply the greedy decoder
    Group the exams into their consecutive time slots
    Evaluate the objective vector and store it: (OB_1, OB_2)
    {*Phase 2: MOO 1*, $OB_1 = P$ for the first 1/4 of *totalGeneration*}
    {*Phase 3: MOO 2*, $OB_1 = T$ for the final 3/4 of *totalGenerations*}
    Also evaluate and store the number of time slots, T
for ($generation = 1$; $generation < totalGenerations$; $generation++$) **do**
    for all strings in the population **do**
        Select each string in turn, it becomes *parent*
        Call applyOperations
        Evaluate the objective vector (and T) for the offspring
        if the offspring is a duplicate **then**
            It dies
        else if the offspring dominates its parent **then**
            It replaces it in the population
        else if the offspring neither dominates nor is dominated by its parent **then**
            it replaces another individual that it dominates at random, if such an individual exists
        else if The offspring's objective vector improves on $bestOB_1$, or $bestOB_2$ **then**
            The offspring replaces the individual with the worse value for OB_1 in the population
            $bestOB_1$ or $bestOB_2$ is updated, as appropriate
Apply *Phase 4: local search*, to non-dominated solutions for (T, *proximity*), {optional}
Output the non-dominated vectors, (T, *proximity*)

Following the application of the chosen operators, the new individual will be evaluated

according to the performance measures, for timetable length (OB_1) and proximity cost (OB_2). As previously discussed in Section 4, we will use different performance measures for OB_1 at different stages of the MOEA. We use P for the first 25 % of generations, and T for the remaining 75 %. P encourages shorter timetables , and T is more flexible in allowing exams to be moved from one time slot to another. (These measures are defined and explained in detail in Section 4.5).

Once the new offspring has been evaluated, a decision will be made regarding its survival, according to the conditions stated in Algorithm 1. Offspring that have identical values for their objective vector, (OB_1, OB_2) with one or more member of the current population, will be deleted immediately. An offspring that survives the 'duplicates test' will next be compared to its parent, and if it dominates its parent, it will replace it in the population. If, on the other hand, the offspring is dominated by its parent, then it will die. A final possibility is that the offspring and its parent will prove mutually non-dominating. When this is the case, a random search will be undertaken, without replacement, to try to find *any* member of the current population dominated by the new offspring, and the first such individual to be found will be replaced by the new offspring. If no such candidate for replacement is found, however, the new offspring will die, unless either of the offspring's objective functions variables, OB_1 or OB_2, harbor a global *best-so-far* value. When this happens, the MOEA will ensure that this offspring is not lost, by replacing the population member with the worst value for OB_1 with the new individual. This somewhat biased replacement criterion has been chosen to try to prevent timetables from getting too long. Please note that the precise replacement criteria for the simple MOEA framework are easily changed.

In addition to the switch of performance measures for OB_1, the transition between *MOO 1* and *MOO 2* also includes a mechanism to prune the objective space (not shown in the psuedocode). This allows the user to restrict the range of timetable lengths produced in the final set of solutions, and thus avoid time-wasting computations on solutions that are considered out of range (e.g., timetables too long to be of practical interest). To achieve this, the objective values for unwanted solutions are reset to arbitrarily high values, to allow *MOO 2* to eventually replace them all with timetables within the required range.

4.5 Performance Measures for the UETP

The dual objectives we wish to tackle involve the simultaneous minimization of the timetable length, OB_1, and the total proximity costs, OB_2. For the early part of the multiobjective optimization (*MOO 1*) we use a measure, $OB_1 \leftarrow P$, that favors timetables with an uneven spread of exams among the time slots, to encourage reassignment from sparsely occupied time slots so that they can be eliminated to produce shorter timetables. Once the MOEA has achieved a good spread of timetable lengths, however, P can be rather restrictive, allowing examinations little mobility between time slots. Thus, during the latter part of execution, the MOEA will switch to direct measurement of the number of time slots, $OB_1 \leftarrow T$, giving more scope for exams to move between time slots. Proximity costs, OB_2, are calculated using weights to penalize examination schedules that require individual students to sit examinations that occur in time slots that are close together. The closer together they are, and the larger the number of students affected, the higher the total proximity cost will be. More details of the performance measures used are given below.

4.5.1 Minimizing the Number of Time Slots

As mentioned above, the idea in *MOO 1* is to devise a function, P, capable of making some kind of distinction between 'good assignments' and 'bad assignments' of examinations to time slots, for a given length of timetable.

Assume that we have T time slots and rank them according to some criterion that measures the 'fullness' or 'inertia' of that time slot, such as the number of exams scheduled

within it, or the degree sum for that time slot, for example. Suppose that the time period that is deemed the most difficult to eliminate is ranked '1', and the second most difficult is ranked '2', and so on, with the one that is considered the easiest to eliminate ranked 'T'. Equation 1 defines such a performance measure based on degree sums within each time slot:

$$P = \sum_{1}^{T} r_i D_i + T \sum_{1}^{T} D_i \tag{1}$$

In Equation 1, $D_i = \sum_{j \in S_i} d_j$ represents the *degree sum* for time slot i, with d_j denoting the exam degree of the j^{th} exam (i.e., the total number of exams that cannot be scheduled in the same time slot as exam j). r_i is the rank of time slot i, and T represents the total number of time slots, as before. P favors solutions with large numbers of highly constrained exams concentrated in the same time slot. Under this regime it is assumed that weakly constrained exams in sparsely populated time slots will gradually be reassigned, eventually driving down the total number of time slots. P was inspired by a measure introduced by Culberson and Luo, [19], for the graph coloring problem, in which the sum of the values of the integer color labels was modified by a term nc (the number of vertices times the number of colors). In this earlier measure the colors are not ranked, making the quantity rather unstable due to its heavy dependence on the arbitrary assignments of integer labels to colors. On the other hand, the measure introduced in the present paper does not suffer in this way, as it uses ranks (indicating the level of constraints) in place of arbitrary time slot labels.

If the time slots are ranked according to different criteria (other than degree sums), other performance measures can be designed along similar lines to Equation 1. Pilot studies indicate that degree sum is indeed a good criterion to use for ranking, when compared with others, such as counting the number of examinations or students in each time slot. Of course we are not at all interested in the P values for our final solution set, only timetable lengths, T and *proximity*.

4.5.2 Minimizing Proximity Costs

The proximity costs described by Laporte and Desroches in [23] and used by Carter *et al* in [14] have been chosen for the present study. In this scheme a cost, w_s, is imposed, whenever a student has to sit two examinations scheduled s periods apart. The weights applied are as follows: $w_1 = 16, w_2 = 8, w_3 = 4, w_4 = 2$ and $w_5 = 1$. Using these weights, cost values can be evaluated for each student and all of these are then added together to give a total cost accumulated for all students. We will call this accumulated cost our *proximity* cost, with $OB_2 \leftarrow proximity/Number\ of\ students$.

4.6 Mutations

We use two mutation operators to provide extra diversity in the search. One mutation operator used is the *insertion mutation* (also known as *position based mutation*) described by Davis, [20]. This operator simply involves selecting two exams at random from a permutation list, and placing the second before the first.

The second mutation is more sophisticated and aims to swap groups of exams between pairs of time slots, where this can be done without violating any hard constraints. The previously mentioned Kempe chain interchange forms the basis of this operation. However, in order to reflect time slot swaps in an order-based representation, it is necessary to resequence the exams in the permutation lists, following a successful Kempe chain interchange. We use the grouping operation of Culberson and Luo, as illustrated in Figure 1 (c), to do this.

Kempe chain mutation proceeds as follows: first, a connected edge (i, j) is selected at random. This simply involves choosing a pair of examinations where at least one student is

taking both. The routine then constructs a connected subgraph to include all linked exams currently assigned to the time slots occupied by exams i and j, $t(i)$ and $t(j)$ respectively. Next, all the time slots in this subgraph are relabelled, so that $t(i)$ becomes $t(j)$ and vice versa. For the uncapacitated problem, new timetables resulting from such Kempe interchanges will always be conflict free, as long as the original timetable was conflict free. If additional constraints are involved however, further checks will probably be needed. For the capacitated problem, for example, rearrangement of examinations into different time slots may produce a violation in seating capacity, with too many students and not enough seats for a time slot. Thus, it is essential to check for such constraint violations before any Kempe interchanges are made permanent. Should a Kempe interchange be rejected on the ground of capacity (or other constraint) violation, the process is repeated until a suitable candidate is found.

4.7 Local Search using Kempe Chain Reductions

In addition to its use for mutation, the Kempe chain interchange process is also applied at the end of our process to try to improve the proximity costs of the non-dominated individuals in the final population. However, this time the Kempe interchanges are seeded repeatedly and methodically, instead of selecting a single edge at random on a 'one off' basis. Each connected edge in the original graph becomes the focus of attention for our simple hillclimber, and Kempe interchanges that result in improvements to the proximity cost are saved, provided that no constraint is violated by the new candidate timetable. Poor and illegal solutions are discarded. The stopping criterion that we use involves at most three iterations through the above routine. It has been noticed that improvements become progressively smaller as time goes on. Furthermore, the local search is halted even earlier, following a full iteration in which no permanent changes resulted.

5 Characteristics of the Data Sets

The Toronto benchmarks [14] have been chosen for the present work. As pointed out in [29], however, there are two versions of this data in circulation. The benchmarks identified as version I in [29] is the most popular of the two sets, and this version has been used in the present paper. However, for all but six of these instances a maximum seating capacity for the sessions is not defined in the original data. In addition, no session restrictions are included. To model these additional constraints, we use data created by L. Merlot and his colleagues, members of the Operational Research Group at the University of Melbourne. This data can be found at:

http://www.or.ms.unimelb.edu.au/timetabling/ttframe.html?ttexp4.html

Merlot *et al* obtained capacity values for the Toronto instances, for which this quantity was undefined in the original data, by first dividing the total number of student-exams by the number of time slots used in the research at Melbourne. They then added another 5 % capacity in order to allow a small amount of slack. For session restrictions the timetable length was also fixed in advance, and approximately 10 % of exams were selected to have their number of sessions restricted. For each of the chosen exams, a random integer between one and five was generated to determine the level of restriction to be imposed. Finally, specific time slots were chosen to match the restrictions.

The main characteristics of the data are summarized in Table 1. Note: the instance pur-s-93 has been omitted from the study, because of time limitations (it is very much larger than all the other Toronto instances). The present author included results for pur-s-93 in the earlier conference paper [28], as part of a much smaller study involving the capacitated problem and just 6 data sets. The first five columns of Table 1 are self explanatory, and

Table 1: Characteristics of the Toronto Instances

Instance	exams	students	edges	seats	GCP slots	BPP slots	UB_{av} Prox
car-f-92	543	18419	20305	2000	28	28	44
car-s-91	682	16926	29814	1550	28	37	54
ear-f-83	190	1125	4793	350	22	24	164
hec-s-92	81	2823	1363	650	17	17	64
kfu-s-93	461	5349	5893	1955	19	13	90
lse-f-91	381	2726	4531	635	17	18	69
rye-s-93	486	11483	8872	2055	21	22	70
sta-f-83	139	611	1381	465	13	13	230
tre-s-92	261	4362	6131	655	20	23	55
uta-s-92	622	21266	24249	2800	30	22	39
ute-s-92	184	2750	1430	1240	10	10	77
yor-f-83	181	941	4706	300	19	21	140

Column 6 lists the best known solutions to the underlying graph coloring instances. The uta-s-92 best is taken from [12]. All the other graph coloring solutions can be found in [14]. Column 7 presents solutions to the underlying bin packing instances, as calculated by the present author, [28]. Interestingly, the seating capacity limitation is identical to the weight capacity constraint for the bin packing problem: the items of various sizes (bin packing) being replaced with examinations having various numbers of candidates (timetabling). Thus, a feasible solution to the timetabling problem that avoids all clashes and seats all students requires the simultaneous solution to the underlying graph coloring and bin packing problems. Column 8 specifies an upper bound for the proximity cost for each instance. This is evaluated by generating the worst possible examination schedule for each individual student, and then adding together the corresponding proximity costs. Assuming no clashes, pathological schedules for individual students consist of a sequence of his/her exams with no gaps. The total pathological proximity costs are divided by the number of students to give an average, UB_{av}, as quoted in Column 8. This UB measure simply assumes that each student has all of his/her exams in one continuous sequence. Instances with the highest values in Column 8 (i.e., sta-f-83, ear-f-83 and yor-f-83) correspond to universities where the students have the most examinations to sit.

6 Incorporating the Hard Constraints

Hard constraints need to be checked carefully by the greedy algorithm when it is assigning examinations to time slots, and the algorithm should not allocate a time slot to an exam if it produces an infeasible timetable that violates one or more of the hard constraints.

6.1 The Uncapacitated Problem

The uncapacitated problem imposes only one hard constraint: no student is allowed to sit more than one examination at a time, i.e., the 'clash' constraint. Thus, the greedy algorithm will only have to check for clashes, with each attempt to allocate a time slot to an exam.

6.2 The Capacitated Problem

In addition to the 'clash' constraint above, the capacitated problem restricts the number of students per session (i.e., applies a 'bin packing' constraint), in order to ensure that each student has a seat in an examination room. With this version of the problem it is necessary for the greedy algorithm to ensure that there are no clashes and that there are enough seats for all the students, with each allocation of exam to time slot.

6.3 The Capacitated Problem with Session Restrictions

In this final version of the problem, the greedy decoder has three separate constraints to check: clashes, seats, and restricted sessions. As mentioned above, the restricted sessions data obtained from the University of Melbourne fixed the timetable lengths in advance (usually allowing more sessions than the minimum), then selected 10 % of exams to have their number of sessions restricted.

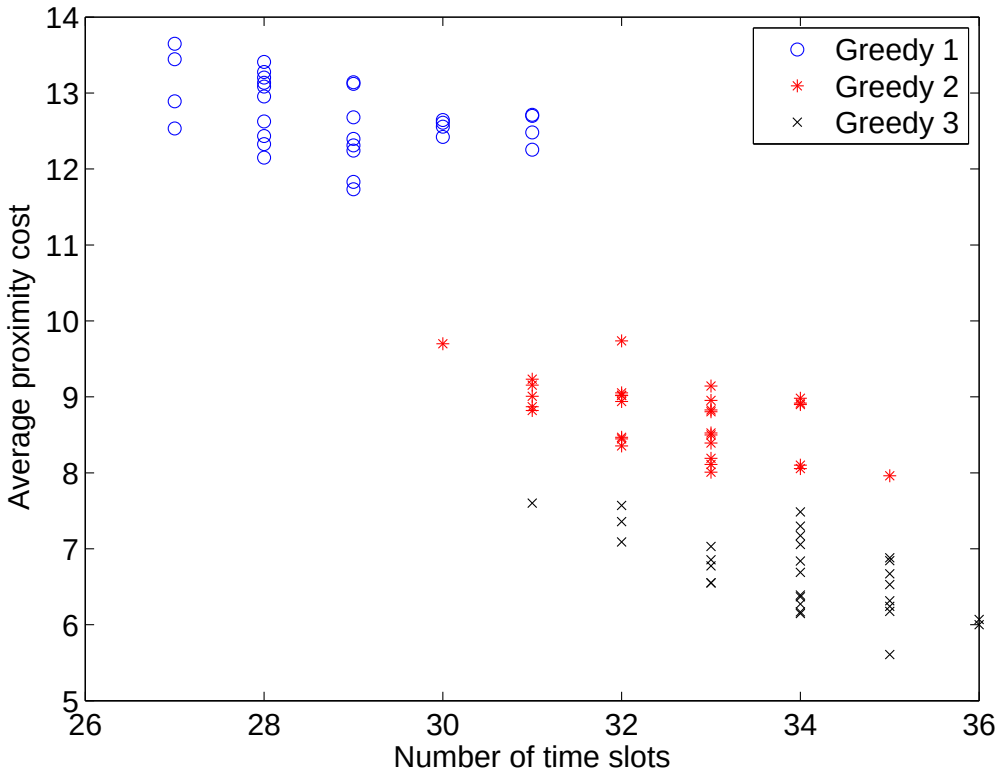
Unfortunately there is a potential problem with timetable feasibility, if restricted sessions are imposed. Although hard constraints generally pose no problem for order-based (i.e., sequential) approaches using greedy decoders, restricted sessions prove to be an exception. Consider an ordering of examinations and a greedy algorithm that has progressed part way through the list, to produce a partially constructed timetable. Assume that the next exam encountered on the list is one of the 10 % with restricted sessions. It is entirely possible that none of the allowable time slots is feasible for the exam currently under consideration - it may clash with other exams already allocated to those time slots, or perhaps produce a violation of seating restrictions. Recall that each exam with restricted sessions, has only between 1-5 time slots available to it. If none of those sessions work for a particular exam, given previous allocations to those time slots, the current exam will remain unscheduled, and the greedy decoder will fail to produce a solution.

To try to help matters, the initial population is seeded with permutation lists in which the exams with restricted sessions appear first, so that these exams can be assigned before the others (at least in the initial population). More robustly, the feasibility issue is addressed by relaxing the 'restricted sessions' constraint, to allow exams with restricted sessions to be allocated to time slots late on in the timetable, if they cannot be accommodated in one of their specified slots. To discourage this from happening, however, we impose a penalty whenever an exam has to be placed in one of these 'late slots'. In the present context, timetables with 'late slots' simply refer to timetables that are longer than those fixed in the University of Melbourne study, for example hec-s-92 is limited to 19 time slots, and sta-f-83 to 14, and so on.

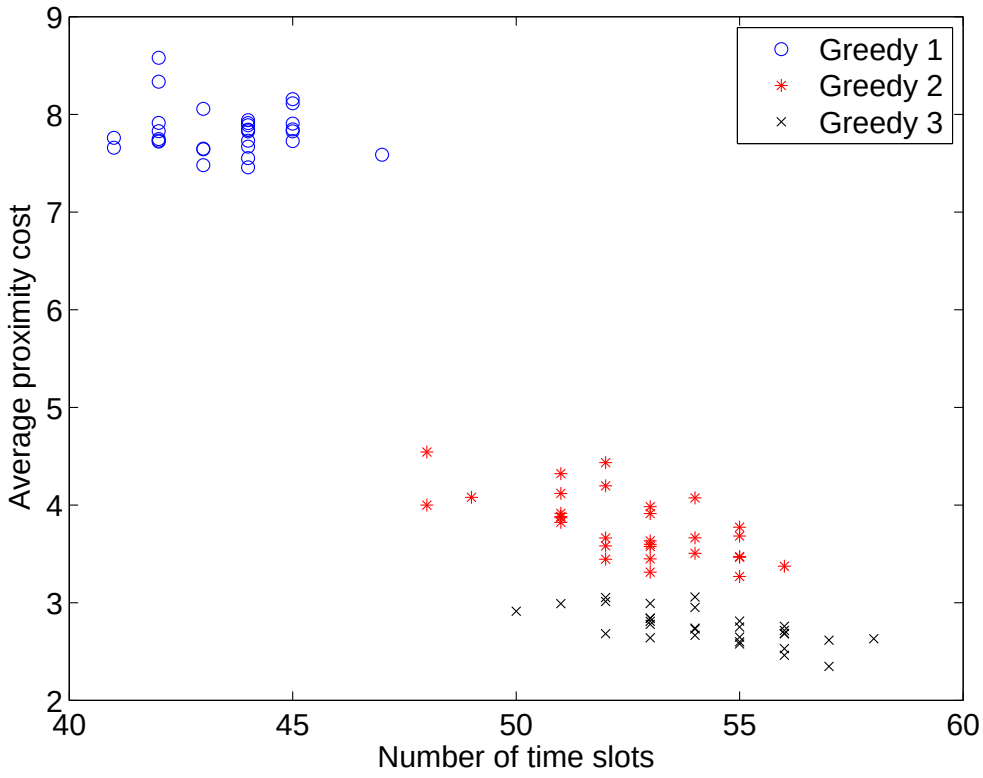
The penalty we impose involves counting the number of exams with restricted session violations in a particular solution, then scaling this value according to OB_1. Thus, we use either $OB_1 \leftarrow P + P \times N_{ex}$ or $OB_1 \leftarrow T + T \times N_{ex}$ where N_{ex} denotes the number of exams with violated restrictions. As an example consider the instance hec-s-92, which has 19 session imposed in the University of Melbourne database. Suppose that the greedy algorithm in the present order-based approach is unable to allocate a particular exam having restricted sessions to any of the slots specified for that exam. To avoid a solution with an unassigned exam, the restrictions will be relaxed to allow the greedy algorithm the earliest time slot, occurring later than time slot 19, that it is able to choose without violating clash or seating constraints.

7 Performance of the Greedy Algorithms on Random Orderings

It is a useful exercise to compare the results produced by the three greedy algorithms applied to random orderings. Figures 2(a) and 2(b) show the dual objective plots obtained by running each algorithm on 30 random orderings of two benchmark problems, **tre-s-92** and **car-f-92** respectively, taken from the Toronto data sets. Clearly, greedy 1 produces the shortest timetables with the worst proximity costs and greedy 3 the longest timetables with the best proximity costs, and the greedy 2 results in between the other two.



(a) **tre-s-92**



(b) **car-f-92**

Figure 2: Comparing greedy 1, greedy 2 and greedy 3, each run 30 times on random orderings for tre-s-92 and car-f-92 on the uncapacitated problem

8 The Greedy Algorithms in a Multiobjective Framework

Section 7 illustrates how a different profile of results can be obtained if the greedy decoder considers the proximity costs when assigning time slots to random orderings of examinations. We shall now turn our attention to examine how well the three greedy algorithms perform when applied to *intelligent orderings* obtained by *iterative improvement* in a multiobjective framework. We will begin by incorporating the CL grouping and reordering heuristics into a multiobjective framework, and then we shall examine the added advantage of including mutation operators and the final local search. Experiments with crossover operators have been omitted from the present study. Although successful order-based crossovers (POP and MIS) for the related graph coloring problem [27] have been developed by the present author, and one of them (POP) used for multiobjective timetabling in [28], further tests failed to demonstrate their effectiveness in the present context.

8.1 Multiobjective Framework with Grouping and Reordering Heuristics

The first set of experiments involve the use of grouping and reordering heuristics as described in Section 4.1. Details of the iterative procedure are presented below in Function 1. As can be observed, one reordering heuristic is chosen at random in the given proportion, from {insert, reverse, decreasing-total-degree} for each iteration, and then the greedy decoder is applied (either greedy 1, greedy 2 or greedy 3). Finally, the exams are regrouped into their consecutive time slot sequence. Following the application of the operators, the new individual will replace a parent, replace another individual or die, following the precise conditions stated in Algorithm 1. Please note that OB_1 switched between P and T once 25 % of the generations were completed, as stipulated in Algorithm 1.

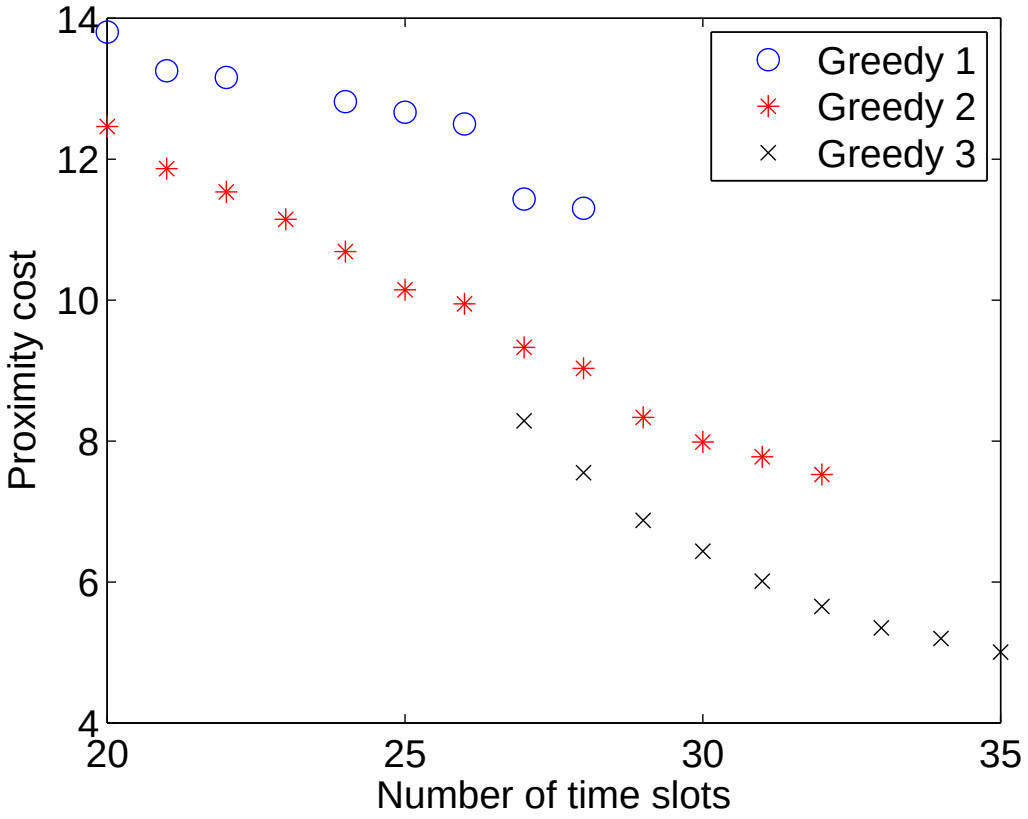
Function 1 ApplyOperations: Single iteration of grouping and reordering

Choose a reordering heuristic at random (**insert** 50 %, **reverse** 25 % or **decreasing-total-degree** 25 %) and apply it to create a single offspring
Apply the greedy decoder
Group the exams into a consecutive time slot sequence

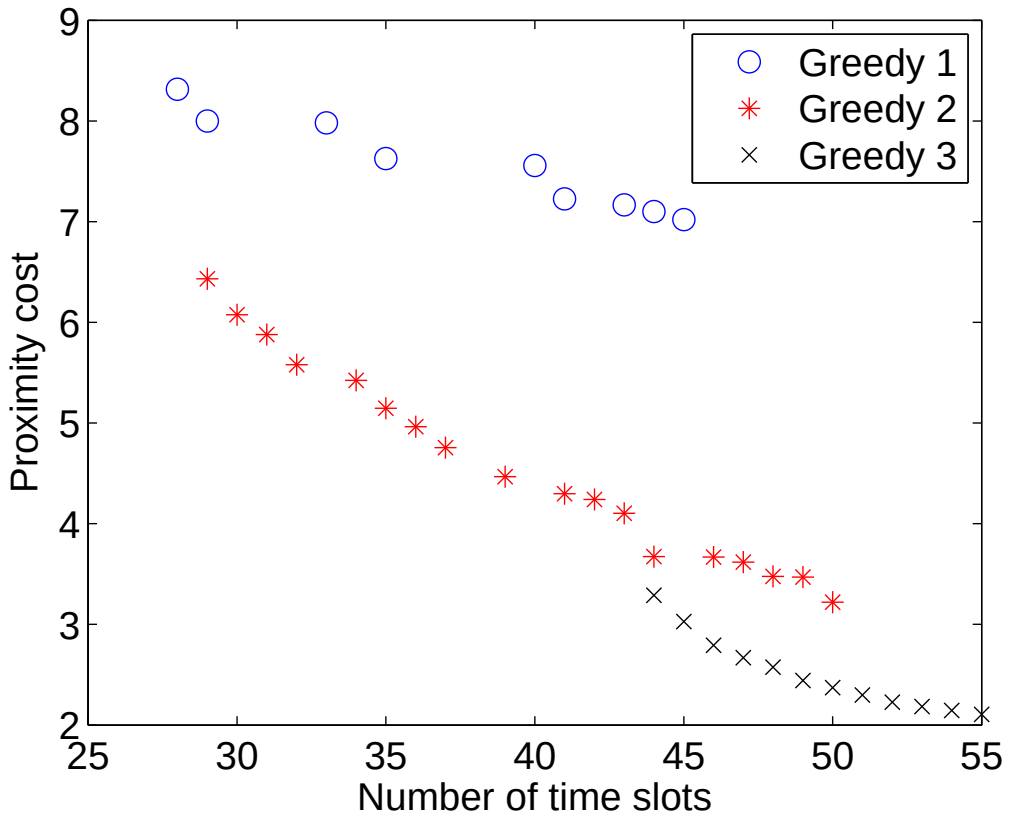
The traces in Figures 3(a) and 3(b) represent the non-dominated solutions extracted from five replicate runs of the multiobjective framework incorporating the grouping and reordering heuristics, on tre-s-92 and car-f-92 respectively. A population size of 400 was used for these experiments and the algorithms run for 2,000 generations (note: no limitation on the range of timetable lengths is imposed on phase *MOO 2* for these experiments). From the results we can see that that the greedy 1 algorithm produces the shortest timetables and the highest proximity costs, while the greedy 3 operator favors lower proximity costs, but produces a poor spread of timetable lengths. Greedy 2, on the other hand, appears to produce a good set of compromise solutions. Notably, each set of results is evenly spread indicating uniform coverage of the approximate Pareto trade-off surfaces.

8.2 Grouping and Reordering with Mutations and Final Local Search

For this set of experiments the same timetabling instances were used. This time the mutations were added and a local search applied to the best individuals in the final population. Once again, the non-dominated results were extracted from 5 replicate runs and graphs plotted. A population of 400 was used. This time, however, the MOEA was allowed to run for 5000 generations, because the addition of mutation operators produced a slower convergence rate. OB_1 switched from P to T 1/4 way through, as before (once again, no limitation on the range of timetable lengths is imposed on phase *MOO 2* for these experiments). The exact mechanism for selecting the operators is detailed in Function 2.

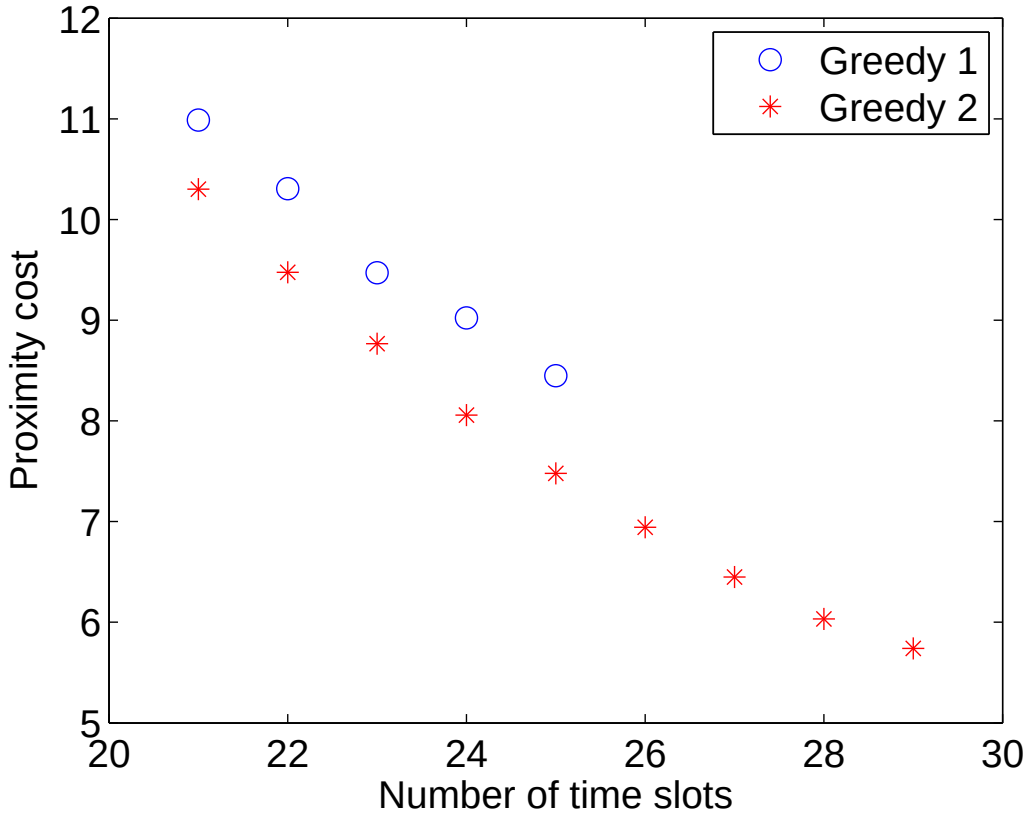


(a) **tre-s-92** with grouping and reordering

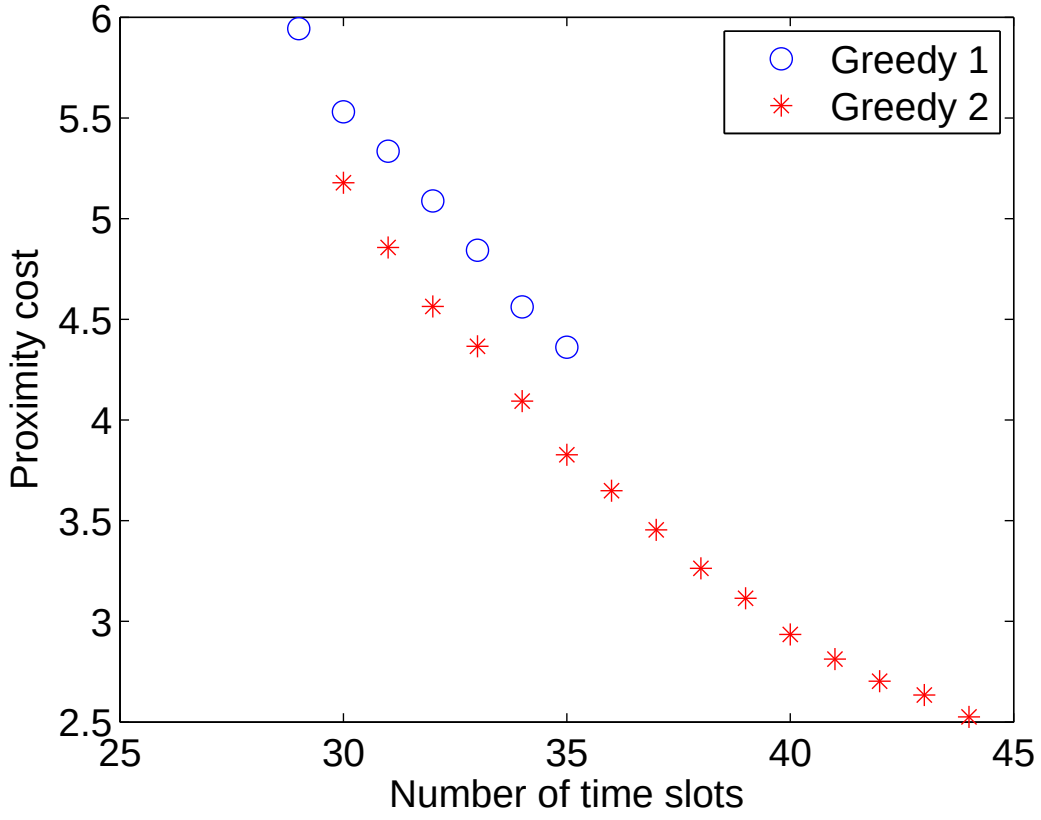


(b) **car-f-92** with grouping and reordering

Figure 3: Comparing greedy 1, greedy 2 and greedy 3 in a multiobjective framework for tre-s-92 and car-f-92 on the uncapacitated problem, incorporation grouping and reordering heuristics



(a) **tre-s-92** with the MOEA



(b) **car-f-92** with the MOEA

Figure 4: Comparing greedy 1 and greedy 2 on tre-s-92 and car-f-92 on the uncapacitated problem in a multiobjective framework with grouping, reordering, mutation and local search

Function 2 ApplyOperations: Single iteration, choosing grouping or mutation

Choose either reordering ($Prob_{reord}$) or mutation ($P_m = 1 - Prob_{reord}$) {$Prob_{reord}$ = 100%, at start of MOEA, and 0% at the end, decreasing evenly throughout the execution of the MOEA}
if reordering is chosen **then**
    Choose a reordering heuristic at random (**insert** 50 %, **reverse** 25 % or **largest-total-degree-first** 25 %) and apply it to create a single offspring
else
    Choose a mutation (**insertion** 50 % or **Kempe** mutation 50 %)
Apply the greedy decoder
Apply grouping

In Function 2 the reordering heuristic is selected from the same menu as before, with the same relative probabilities. With mutation, the proportion is 50 % : 50 % for **insertion** and **Kempe mutation**. For each iteration only one operator is selected, either a grouping operator or a mutation. If grouping is chosen, then mutation is not applied, and vice versa. The relative rate of application of grouping versus mutation is allowed to vary, depending on the stage of execution of the MOEA algorithm. At the start, grouping is applied at 100 % and mutation at 0 %, but at the end these proportions are reversed. This regime seemed to work well, although it is possible that other mixes of the operators prove equally effective. However, fine tuning these proportions is beyond the scope of the present study. For the final population, the non-dominated individuals are selected and a final local search, consisting of Kempe chain interchanges (see Section 4.7), is applied to these superior individuals. This final stage does not involve any order-based greedy operations - improvements to proximity costs are simply made where it is possible to do so without violating any hard constraints.

The plots for the MOEA are illustrated in Figure 4. This time, greedy 3 has been omitted from the experiments because it produced a poor range of timetable lengths in the previous experiments. In the current set, it is clear that a better spread of results is obtained if greedy 2 is used, rather than greedy 1. For this reason the main experiments, documented in the next section, will use the MOEA with the operations from Function 2. Once again, the results show an even spread over the approximate Pareto trade-off surfaces.

Please note that extensive tests verified that each of the four stages (*Initialization*, *MOO 1*, *MOO 2* and *local search*) of the multiobjective approach make a useful contribution to the final results. Replacing *MOO 2* with an extension of the local search stage (i.e., Kempe chain reductions), for example, produced consistently poorer results. The final local search when it follows *MOO 2*, generally improves proximity costs by about 0.2 %.

9 The Main Results

This section documents the results obtained by running the MOEA described in Algorithm 1, with the greedy 2 decoder on the Toronto benchmarks. A population size of 200 was used and the MOEA run for 10,000 generations on all instances. To demonstrate the robustness of the new method the same parameters were used throughout, with no special tuning for the different instances. Three variations of the UETP are solved, as described in Section 6: the uncapacitated problem, the capacitated problem and the capacitated problem with some exams having restricted sessions. In all cases the MOEA attempts to simultaneously minimize the timetable length and the proximity cost, producing a set of trade-off solutions. The range of timetable lengths is purposely restricted in the stage *MOO 2*, to the shortest timetable found + 4 time slots. This is to concentrate the MOEA on shorter timetables and also avoid excessive tabulation of results. Where possible the results are compared with those obtained by other recent dual objective approaches. All the current experiments were carried out using Visual C++ with a Windows XP operating system, on an Intel Pentium

Table 2: MOEA results for uncapacitated problem

Slots	Greedy 2 Ave (Bst)	Côté *et al* Ave (Bst)	Slots	Greedy 2 Ave (Bst)	Côté *et al* Ave (Bst)
	car-f-92			**rye-s-93**	
30	5.1 (5.0)	**4.9** (**4.9**)	21	**11.4** (**11.3**)	-
31	4.8 (4.7)	**4.7** (**4.5**)	22	10.3 (10.1)	**10.1** (**9.8**)
32	4.5 (4.5)	**4.3** (**4.2**)	23	9.2 (9.1)	**9.1** (**8.8**)
33	**4.3** (**4.2**)	4.5 (**4.2**)	24	8.3 (8.2)	**8.1** (**7.8**)
34	**4.0** (4.0)	4.1 (**3.9**)	25	7.5 (7.4)	**7.2** (**7.0**)
	car-s-91			**sta-f-83**	
32	6.4 (6.3)	**6.2** (**6.1**)	13	**157.1** (**157.0**)	**157.1** (**157.0**)
33	6.1 (5.9)	**5.9** (**5.7**)	14	**140.4** (140.3)	**140.4** (**140.2**)
34	5.7 (5.6)	**5.5** (**5.4**)	15	**126.0** (125.6)	**126.0** (**125.2**)
35	**5.4 (5.3)**	5.5 (5.4)	16	113.7 (112.9)	**113.2** (**112.7**)
36	**5.1** (**5.0**)	5.3 (5.2)	17	102.7 (102.1)	**101.6** (**101.4**)
	ear-f-83			**tre-s-92**	
22	**42.4** (**41.5**)	-	21	**10.3** (**10.2**)	10.5 (10.3)
23	**38.2** (**37.3**)	39.0 (38.0)	22	**9.4** (**9.3**)	**9.4** (9.4)
24	**34.8** (**34.2**)	35.6 (**34.2**)	23	**8.6** (**8.5**)	8.8 (8.6)
25	32.1 (**31.5**)	**31.9** (31.6)	24	**8.0** (**7.8**)	8.1 (7.9)
26	29.8 (29.2)	**29.7** (**28.8**)	25	7.4 (**7.2**)	**7.3** (**7.2**)
	hec-s-92			**uta-s-92**	
17	**12.1** (**12.0**)	**12.1** (**12.0**)	33	**4.3** (**4.2**)	-
18	**10.5** (**10.3**)	**10.5** (10.4)	34	**3.9** (3.8)	**3.9** (**3.7**)
19	**9.3** (**9.1**)	**9.3** (9.3)	35	3.8 (3.7)	**3.6** (**3.5**)
20	8.3 (**8.1**)	**8.2** (**8.1**)	36	3.6 (3.5)	**3.4** (**3.3**)
21	**7.5** (**7.3**)	**7.5** (**7.3**)	37	3.4 (3.3)	**3.2** (**3.2**)
	kfu-s-93			**ute-s-92**	
19	**15.8** (**15.5**)	16.2 (15.8)	10	**25.3** (**25.2**)	25.5 (25.3)
20	**14.1** (**13.9**)	14.4 (14.3)	11	**20.8** (**20.6**)	21.2 (20.7)
21	**12.5** (12.3)	12.8 (**12.1**)	12	17.2 (17.0)	**17.1** (**16.8**)
22	**11.4** (11.2)	11.6 (**11.0**)	13	-	**14.2** (**13.9**)
23	**10.3** ((10.1)	**10.3** (**10.0**)	14	-	**11.6** (**11.5**)
	lse-f-91			**yor-f-83**	
17	13.1 12.8)	**12.6** (**12.3**)	19	-	**45.6** (**44.6**)
18	11.6 (**11.3**)	**11.5** (**11.3**)	20	**40.9** (**40.6**)	41.0 (**40.6**)
19	10.2 (10.0)	**10.1** (**9.7**)	21	**37.4** (37.2)	37.6 (**36.4**)
20	**9.1** (8.8)	9.3 (**8.5**)	22	**34.2** (34.0)	34.5 (**33.8**)
21	8.2 (8.0)	**8.1** (**7.7**)	23	**31.4** (**31.1**)	31.9 (31.6)

4 PC with a 3.6 GHz processor and 1 GB RAM.

9.1 Uncapacitated Results

Results for the uncapacitated problem are presented in Table 2, where they are compared with those obtained by Côté *et al*, [18]. In both cases the average and best are recorded from five replicate runs for a range of timetable lengths. Bold entries in the table signify which of the two approaches obtain the better average or 'best' value. Run times are difficult to compare when experiments are carried out on different platforms. However, they are broadly similar and will be discussed later. Examination of the results in Table 2 shows the results of Côté *et al* are, on balance, a little better than those obtained in the current study. However, they are very close and the new MOEA equals or outperforms the other study in several places.

Table 3 compares proximity scores obtained using the current MOEA (on the Toronto data set number I) with some recent results from state-of-the-art algorithms. Apart from the present MOEA and the algorithm from Côté *et al*, all the other results have been produced using single objective approaches, with the timetable lengths fixed in advance. Thus, the new MOEA does not break any records for the uncapacitated instances, but the results are competitive, nevertheless.

Table 3: Comparing some uncapacitated results with other work: Côté *et al* [18], Yang and Petrovic [36], Abudullah *et al* [1] and Burke et al [8]. The bracketed value in Column 1 indicated the number of time slots

Data set	MOEA	Côté	Yang	Abdullah	Burke
car-f-92 (32)	4.5	4.2	**3.93**	4.4	4.0
car-s-91 (35)	5.3	5.4	**4.5**	5.2	4.6
ear-f-83 (24)	34.2	34.2	33.7	34.9	**32.8**
hec-s-92 (18)	10.3	10.4	10.83	10.3	**10.0**
kfu-s-93 (20)	13.9	14.3	13.82	13.5	**13.0**
lse-f-91 (18)	11.3	11.3	10.35	10.2	**10.0**
rye-s-93 (23)	9.1	8.8	8.53	**8.7**	-
sta-f-83 (13)	**157.0**	**157.0**	158.35	159.2	159.9
tre-s-92 (23)	8.5	8.6	7.92	8.4	**7.9**
uta-s-92 (35)	3.7	3.5	**3.14**	3.6	3.2
ute-s-92 (10)	25.2	25.3	25.39	26.0	**24.8**
yor-f-83 (21)	37.2	36.4	36.35	**36.2**	37.28

9.2 Capacitated Results

Results obtained for the capacitated problem are compared in Table 4 with the best obtained in an earlier conference paper by the present author [28], where this is possible. Unfortunately thorough benchmarking is not possible, due to an absence of work on the capacitated problem using the proximity measure of [23]. The only other similar dual objective work is by Wong et al [35]. However, they obtained timetables that were mostly much longer than is the case for the present study (see Table 5). In addition, previously mentioned issues regarding proximity cost measures caused added difficulties, and made it impossible to make comparisons in the few places where there was an overlap in ranges of timetable lengths. Clearly the results using the newer version of the MOEA are far superior to those produced in the earlier study by the same author. The earlier MOEA used crossover and the simpler greedy decoder, greedy 1, rather than the greedy 2 decoder used for the present work. Furthermore, the newer version benefits from a Kempe chain mutation and finishes off with a local search. Additionally, the fact that the timetables obtained in the present

Table 4: MOEA results for capacitated problem

Slots	greedy 2 Ave (Bst)	Mumford Bst	Slots	greedy 2 Ave (Bst)	Mumford Bst
	car-f-92			**rye-s-93**	
31	-	6.6	23	12.0 (11.6)	-
32	4.9 (4.9)	6.3	24	10.4 (9.9)	-
33	4.7 (4.6)	6.1	25	9.0 (8.9)	-
34	4.3 (4.3)	6.0	26	7.9 (7.8)	-
35	4.1 (4.0)	5.8	27	7.2 (7.1)	-
	car-s-91			**sta-f-83**	
37	-	7.2	13	167.1 (166.9)	-
38	5.6 (5.5)	6.8	14	147.8 (147.5)	-
39	5.2 (5.1)	6.6	15	131.5 (131.2)	-
40	4.9 (4.9)	6.5	16	117.9 (117.4)	-
41	4.7 (4.5)	6.3	17	106.2 (105.8)	-
	ear-f-83			**tre-s-92**	
24	45.0 (45.0)	-	24	9.3 (9.2)	10.0
25	39.8 (39.1)	-	25	8.3 (8.1)	9.5
26	35.9 (34.9)	-	26	7.5 (7.4)	9.3
27	33.0 (32.5)	-	27	6.8	9.1
28	30.0 (29.6)	-	28	6.4 (6.3)	-
	hec-s-92			**uta-s-92**	
20	9.6 (9.4)	-	33	4.2 (4.2)	5.5
21	8.5 (8.3)	-	34	4.0 (4.0)	5.4
22	7.4 (7.3)	-	35	3.8 (3.7)	5.2
23	6.6 (6.5)	-	36	3.6 (3.6)	-
24	-	-	37	3.4 (3.4)	-
	kfu-s-93			**ute-s-92**	
19	16.9 (16.9)	23.3	10	31.2 (30.8)	-
20	15.1 (14.9)	21.2	11	25.2 (25.0)	-
21	13.4 (13.3)	21.2	12	20.6 (20.4)	-
22	12.1 (11.9)	20.5	13	16.9 (16.8)	-
23	10.9 (10.8)	20.2	14		-
	lse-f-91			**yor-f-83**	
18	13.8 (13.6)	-	21	42.7 (42.7)	-
19	12.1 (11.9)	-	22	39.1 (38.2)	-
20	10.4 (10.2)	-	23	35.5 (35.3)	-
21	9.3 (9.2)	-	24	32.4 (31.6)	-
22	8.4 (8.3)	-	25	29.7 (28.7)	-

Table 5: Timetable length ranges illustrating that the current approach produces shorter timetables for the capacitated problem

Instance	Current	Wong 2004
car-f-92	32 - 35	39 - 43
car-s-91	38 - 41	51 - 55
kfu-s-93	19 - 23	20 - 24
tre-s-92	24 - 27	34 - 38
uta-s-92	33 - 37	37 - 41

study are so much shorter than those obtained in [35] is encouraging. To highlight this difference, the ranges for the two studies are compared in Table 5.

9.3 Restricted Sessions Results

Table 6: MOEA results for restricted sessions problem, '*' denotes an infeasible solutions

Slots	greedy 2 Ave (Bst)	Slots	greedy 2 Ave (Bst)
car-f-92		**rye-s-93**	
45	2.9 (2.9)	29	6.7 (6.7)
46	2.7 (2.7)	30	5.9 (5.8)
47	2.6 (2.6)	31	5.4 (5.2)
48	2.5 (2.5)	32	4.9 (4.8)
49	2.4 (2.4)	33	4.5 (4.4)
car-s-91		**sta-f-83**	
50	3.3 (3.2)*	15	133.0 (132.5)
51	3.1 (3.1)*	16	118.5 (118.2)
52	3.0 (3.0)*	17	106.9 (106.3)
53	2.9 (2.9)*	18	97.1 (96.6)
54	2.8 (2.8)*	19	88.6 (67.5)
ear-f-83		**tre-s-92**	
30	28.5 (28.0)	30	6.3 (6.1)
31	26.3 (26.0)	31	5.9 (5.8)
32	24.5 (23.8)	32	5.6 (5.5)
33	23.1 (22.6)	33	5.3 (5.2)
34	21.6 (21.2)	34	5.0 (4.7)
hec-s-92		**uta-s-92**	
21	9.6 (9.4)	46	2.5 (2.5)
22	8.2 (8.0)	47	2.4 (2.4)
23	7.2 (7.0)	48	2.3 (2.3)
24	6.3 (6.1)	49	2.3 (2.2)
25	5.6 (5.6)	50	2.2 (2.1)
kfu-s-93		**ute-s-92**	
25	10.2 (10.0)	11	26.0 (25.9)
26	9.4 (9.1)	12	21.2 (20.9)
27	8.5 (8.3)	13	17.4 (17.2)
28	7.9 (7.7)	14	14.5 (14.4)
29	7.5 (7.2)	15	12.3 (12.1)
lse-f-91		**yor-f-83**	
23	9.0 (8.7)	26	30.0 (30.0)
24	7.7 (7.5)	27	27.6 (27.3)
25	7.2 (6.8)	28	25.5 (25.2)
26	6.6 (6.3)	39	24.0 (23.6)
27	6.1 (5.8)	30	22.4 (22.1)

The final set of results, presented in Table 6, are for the capacitated problem with some exams restricted to a very limited range of time slots. No comparisons have been included for this set of results - they are merely presented as a challenge to other researchers. However, it is encouraging to note that values are very close to those presented for the capacitated problem in Table 4 where the ranges of timetable lengths overlap (see hec-s-92, sta-f-83, and ute-s-92). Although some proximity values for fixed timetable lengths are given on the Melbourne University web site, these are limited to a single result for each data set, for

which the timetable length is mostly outside the range of those obtained in the current study. In addition, the present author was a little uncertain of the precise details used to calculate the proximity measure quoted on Melbourne web site. Note that the results for car-s-91 are marked in the Table as 'infeasible'. The reason for this is that a final check on the non-dominated solutions identified exams with allocations made outside their restricted set of sessions, in all cases, for the five replicate runs on this instance. Inspection of the data revealed a higher than normal number of exams restricted to a single time slot. For all other solutions presented in the Table the restricted exams have been allocated legally.

9.4 Run Times for the MOEA

Table 7: Average run times for MOEA in minutes

Problem	# Exams	Uncap	Cap	Restrict
car-f-92	543	295	291	372
car-s-91	682	478	391	503
ear-f-83	190	37	35	41
hec-s-92	81	10	10	10
kfu-s-93	462	268	262	311
lse-f-91	381	195	180	206
rye-s-93	486	276	237	318
sta-f-83	139	22	22	30
tre-s-92	261	70	80	82
uta-s-92	622	432	452	513
ute-s-92	184	49	43	50
yor-f-83	181	35	35	36

The average run times (in minutes) for all three versions of the current MOEA are listed in Table 7. These are found to be broadly comparable with the run times obtained by Côté *et al* [18][4]

It is interesting to note that the run times for the capacitated version of a problem are for the most part faster than those for the uncapacitated version in the present study. It would appear certain that imposing the additional constraint (seating capacity) will reduce the size of the search space, probably reducing the time required by the final local search. On the other hand, addition of a further constraint, in the form of examination time slot restrictions, produces longer run times than can be observed for either of the other variations of the problem. It would be an interesting exercise to insert some check points and carry out some empirical measurements to determine relative timings.

10 Conclusions and Discussion

The current paper presents a set of ideas embedded in a multiobjective framework capable of producing a range of good trade-off solutions to heavily constrained timetabling problems. The following list identifies the key components of the new approach:

- a simple multiobjective framework,
- solutions represented as orderings,
- a greedy algorithm that successfully optimizes timetable length and proximity cost, simultaneously,

[4]Their run times varied between 25 and 816 minutes, but they used a different platform with a slower processor (2.2 GHz, compared with the 3.6 GHz used in the present study).

- effective grouping and reordering techniques for iterative improvement,
- alternative performance measures for timetable length,
- two mutations - insertion and Kempe chain interchange, and
- a local search (also based on Kempe chain interchanges), applied to the very best solutions following the MOEA.

It is well known that that sequential methods are very good at handling timetabling constraints, when simple constructive techniques are used. This present paper illustrates further possibilities for order-based methods by introducing a range of ideas capable of producing iterative improvements.

Results are presented for three versions of the university examination timetabling problem: 1) the uncapacitated problem, 2) the capacitated problem (with seating restrictions) and 3) the capacitated problem with some exams having restricted sessions. Although results for 1) compare reasonably well with recent results for multiobjective approaches in the literature, it is for the more heavily constrained problems like 2) and 3) that the new approach appears to show the most promise: timetables tend to be very much shorter than have been presented elsewhere, yet the proximity costs are not much greater than have been obtained for 1), where comparisons are possible. Furthermore, run times do not appear to become excessive, with additional constraints.

The ease with which the order-based MOEA copes with multiple constraints makes it an ideal candidate for development on bespoke examination timetabling problems: every institution has its own individual idiosyncracies. Furthermore, the approach has potential for many other real-world scheduling problems with several objectives for optimization and multiple constraints. Such problems include staff rostering, assembly line balancing, vehicle scheduling, and many more.

References

[1] S. Abdullah, S. Ahmadi, E. Burke, and M. Dror. Investigating Ahuja-Orlin's large neighbourhood search approach for examination timetabling. *OR Spectrum*, 29(2):351–372, April 2007.

[2] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003.

[3] H. Asmuni, E.K. Burke, and J.M Garibaldi. Fuzzy multiple ordering criteria for examination timetabling. In E. Burke and M. Trick, editors, *PATAT 2004*, volume 3616 of *LNCS*, pages 334–354. Springer-Verlag Berlin Heidelberg, 2005.

[4] E. Burke, Y. Bykov, and S Petrovic. A multicriteria approach to examination timetabling. In *Practice and Theory of Automated Timetabling III: Third International Conference, PATAT 2000*, volume 2079 of *LNCS*, pages 118–131. Springer, 2001.

[5] E. Burke and J. Newall. A memetic algorithm for university exam timetabling. In *Practice and Theory of Automated Timetabling: First International Conference, PATAT 1996*, volume 1153 of *LNCS*, pages 241–250. Springer, 1996.

[6] E. Burke and J. Newall. A multi-stage evolutionary algorithm for the timetabling problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.

[7] E. K. Burke, D. Elliman, P. H. Ford, and R. F. Weare. Examination timetabling in British universities: A survey. In *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling*, pages 76–90, London, UK, 1996. Springer-Verlag.

[8] E.K. Burke, A.J. Eckersley, B. McCollum, S. Petrovic, and R. Qu. Hybrid variable neighbourhood approaches to university exam timetabling. Computer Science Technical Report No. NOTTCS-TR-2006-2, University of Nottingham, 2006.

[9] E.K. Burke, B. Mccollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, January 2007.

[10] E.K. Burke and J. P. Newall. Solving examination timetabling problems through adaption of heuristic orderings. *Annals of Operations Research*, 129:107–134, 2004.

[11] E.K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266–280, 2002.

[12] M. Caramia, P. Dell'Olmo, and G.F. Italiano. New algorithms for examination timetabling. In *WAE '00: Proceedings of the 4th International Workshop on Algorithm Engineering September 05 - 08, 2000*, volume 1982 of *LNCS*, pages 230–242, London, UK, 2001. Springer-Verlag.

[13] M. W. Carter. A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34:193–202, 1986.

[14] M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: algorithms, strategies and applications. *European Journal of Operational Research*, 47:373–383, 1996.

[15] M.W. Carter and G. Laporte. Recent developments in practical examination timetabling. In E. K. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling: Selected papars from the 1st International Conference*, volume 1152 of *LNCS*, pages 3–21. Springer, 1996.

[16] S. Casey and J. Thompson. Grasping the examination scheduling problem. In E. K. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *LNCS*, pages 232–244, Berlin / Heidelberg, 2003. Springer.

[17] C. Y. Cheong, K. C. Tan, and B Veeravalli. Solving the exam timetabling problem via a multi-objective evolutionary algorithm - a more general approach. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)*, pages 165–172, 2007.

[18] P. Côté, A. Wong, and R. Sabourin. A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem. In E. Burke and M. Trick, editors, *PATAT 2004*, volume 3616 of *LNCS*, pages 294–312. Springer-Verlag Berlin Heidelberg, 2005.

[19] J. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 499–520. American Mathematical Society, 1996.

[20] L. Davis. Order-based genetic algorithms and the graph coloring problem. In *Handbook of genetic algorithms*, chapter 6, pages 72–90. Van Nostrand Reinhold, New York, 1991.

[21] D.Brélaz. New methods to color the vertices of graphs. *Communications of the ACM*, 24(4):251–256, 1979.

[22] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, United Kingdom, 2001.

[23] Gilbert Laporte and Sylvain Desroches. Examination timetabling by computer. *Comput. Oper. Res.*, 11(4):351–360, 1984.

[24] D.W. Matula, G. Marble, and J.D. Isaacson. Graph coloring algorithms. In R.C. Read, editor, *Graph theory and computing*, pages 104–122. Academic Press, New York, 1972.

[25] L.T.G. Merlot, N. Borland, B.D. Hughs, and P.J. Stuckey. A hybrid algorithm for the examination timetabling problem. In E. K. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *LNCS*, pages 207–231, Berlin / Heidelberg, 2003. Springer.

[26] C. L. Mumford. Simple population replacement strategies for a steady-state multi-objective evolutionary algorithm. In *Proceedings of the 2004 Genetic an Evolutionary Computation Conference (GECCO)*, pages 1389–1400, Seattle, Washington, USA, 2004.

[27] C. L. Mumford. New order-based crossovers for the graph coloring problem. In T. P. Runarsson, H-G Beyer, E. K. Burke, J. J. Merelo Guervós, L. D. Whitley, and Xin Yao, editors, *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference, Reykjavik, Iceland, September 9-13, 2006, Procedings*, volume 4193 of *LNCS*, pages 880–889, Berlin/Heidelberg, 2006. Springer.

[28] C. L. Mumford. An order based evolutionary approach to dual objective examination timetabling. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)*, pages 179–186, 2007.

[29] R. Qu, E. Burke, B. McCollum, L.T.G. Merlot, and S.Y. Lee. A survey of search methodologies and automated system development for examination timetabling. 2008. (To appear in *Journal of Scheduling*).

[30] A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13:87–127, 1999.

[31] J. M. Thompson and K. A. Dowsland. A robust simulated annealing based examination timetabling system. *Comput. Oper. Res.*, 25(7-8):637–648, 1998.

[32] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[33] C. L. Valenzuela. A simple evolutionary algorithm for multi-objective optimization (SEAMO). In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC2002)*, pages 717–722, Honolulu, Hawaii, 2002. (C.L. Valenzuela is now known as C.L. Mumford).

[34] D.J.A. Welsh and M.B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10:85–86, 1967.

[35] A. Wong, Pascal Côté, and R. Sabourin. A hybrid MOEA for the capacitated exam proximity problem. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1495–1501, Portland, Oregon, 20-23 June 2004. IEEE Press.

[36] Y. Yang and S. Petrovic. A novel similarity measure for heuristic selection in examination timetabling. In E. Burke and M. Trick, editors, *PATAT 2004*, volume 3616 of *LNCS*, pages 377–396. Springer-Verlag Berlin Heidelberg, 2005.