An Enhanced Weighted Graph Model for Examination/Course Timetabling

Julie R. Carrington[*], Nam Pham[†], Rong Qu[†], Jay Yellen[*†]

[*]Department of Mathematics and Computer Science, Rollins College
Winter Park, Florida, USA
{jcarrington, jyellen}@rollins.edu

[†]Automated Scheduling, Optimisation and Planning Research Group
School of Computer Science, University of Nottingham, Nottingham, UK
{nxp, rxq, jzy} @ cs.nott.ac.uk

Abstract

We introduce an enhanced weighted graph model whose vertices and edges have several attributes that make it adaptable to a variety of examination and course timetabling scenarios. In addition, some new vertex- and colour-selection heuristics arise naturally from this model, and our implementation allows for the use and manipulation of various combinations of them along with or separate from the classical heuristics that have been used for decades. We include a brief description of some preliminary results for our current implementation and discuss the further development and testing of the ideas introduced here.

Introduction

Background

Using graph colouring to model timetabling problems has a long history (e.g., Broder 1964, Welsh and Powell 1967, Wood 1968, Neufeld and Tartar 1974, Brelaz 1979, Mehta 1981, and Krarup and de Werra 1982). Several survey papers have been written on this topic (e.g., Schmidt and Strohlein 1980, de Werra 1985, Carter 1986, Schaerf 1999, Burke, Kingston, and deWerra 2004, and Qu et al. 2006).

In a standard graph representation for a timetabling problem, the events to be scheduled are represented by vertices. A constraint (conflict) between two events indicating that they should be assigned different time slots is represented by an edge between the two corresponding vertices. In our case, the events are exams (or courses) and the constraints might be that some students are enrolled in both exams or the same professor is giving both courses. Ideally, then, such exams (courses) would be assigned different time slots. If we associate each possible time slot with a different colour, then creating a conflict-free timetable is equivalent to constructing a *feasible* (or *proper* or *valid*) colouring of the vertices of the graph, that is, a vertex colouring such that *adjacent* vertices (two vertices joined by an edge) are assigned different colours.

Given that vertex colouring is *NP-Hard* (Papadimitriou and Steiglitz 1982), the development of heuristics and corresponding *approximate algorithms*, which forfeit the guarantee of optimality, has been a central part of the research effort.

Two events with a constraint between them are generally prohibited from being assigned the same time slot, i.e., the edge represents a ***hard constraint***. In some university timetabling scenarios, another objective is to minimize the number of students that have to take exams close together (or courses far apart). This ***proximity*** restriction is generally regarded as a ***soft constraint***.

The weighted graph model introduced in 1992 (Kiaer and Yellen 1992a) was designed to handle timetabling instances for which the number of available time slots (colours) is smaller than the minimum needed to construct a feasible colouring. (This minimum number is called the *chromatic number* of the graph.) For instance, in course timetabling, there is likely to be a limited number of time slots that can be used during the week, and a conflict-free timetable may not exist. If conflicts are unavoidable, then a choice must be made on which ones to accept.

Distinguishing among conflicts

Clearly, certain conflicts are worse than others. If two exams (or courses) require the same professor to be present or use the same equipment that cannot be shared, then those two exams must not be scheduled at the same time. On the other hand, if two exams happen to have one student in common, then scheduling those two exams in the same time slot may need to be considered acceptable. In fact, there may be situations where the distinction between hard and soft constraints becomes less clear. For instance, a timetable having a single student scheduled to take two exams in the same time slot (forcing some special accommodation) may actually be preferred to one that has 50 students taking back-to-back exams.

Scope of Paper

This paper introduces an extension of the weighted graph model of Kiaer and Yellen (1992a). This enhanced model holds and keeps track of more of the information relevant to the two sometimes opposing objectives – minimizing total conflict penalty (or keeping it zero) and minimizing total proximity penalty. A natural byproduct of this approach is the emergence of some new heuristics that appear to hold promise for their use, separately or in combination, in fast, one-pass, approximate algorithms.

Such algorithms can prove useful in a number of ways. Because solutions are produced quickly, they can be used within a flexible, interactive decision-support system that can be adapted to a variety of timetabling scenarios.

These solutions can also be used as initial solutions in local search and improvement based techniques, (e.g., Tabu Search, Simulated Annealing, Large Neighborhood Search, Case-Based Reasoning), or as upper bounds for a branch-and-bound algorithm (Kiaer and Yellen 1992b). Recent research has demonstrated that these algorithms, when hybridized effectively or integrated with other techniques such as meta-heuristics, are highly effective on solving timetabling problems (Qu et al. 2006).

Also, because the model lends itself to using various combinations of heuristics for vertex and colour selection, it may prove useful in the context of *hyper-heuristics* (Burke et al. 2003) and/or in an evolutionary computation approach that might involve automatic generation of combinations and switching from one combination to another as the colouring progresses (see Burke et al. 2007).

For an up-to-date survey that includes a broad overview and extensive bibliography of the research in this area in the last ten years (see Qu et al. 2006).

Description of the Model

Although we restrict our attention for this paper to examination timetabling, our model is also applicable to course timetabling. Moreover, it incorporates more of the problem information at input and keeps track of more information pertaining to the partial colouring during the colouring process than do existing timetabling models. These features led us to the design of some new vertex- and colour-selection heuristics, which we introduce in this paper.

Each vertex in the graph corresponds to an exam to be scheduled and each colour corresponds to a different time slot. Accordingly, assigning colour *c* to vertex *v* is taken to mean that the exam corresponding to *v* is scheduled in the time slot corresponding to *c*.

We represent various components of a typical instance of an Examination Timetabling problem using a weighted graph model. Each vertex and each edge are *weighted* with several attributes, some that hold information from the problem instance and others that hold and update information that helps guide the colouring process.

Associated with each vertex is the set of students who must take that exam. Two vertices are joined by an edge, and are said to be *adjacent* or *neighbors*, if it is undesirable to schedule the corresponding exams in the same time slot. Each edge carries information that tells us how undesirable it would be for the corresponding exams to be scheduled in the same time slot or in time slots near each other. In particular, each edge has two attributes: the set of students taking both exams (*intersection subset*); and a positive integer indicating the conflict severity if the exams are scheduled in the same time slot. This second attribute is currently tied to the size of the intersection subset. However, it can also reflect factors not tied to this intersection. For instance, if the same professor is assigned to both exams, then the severity is likely to be set at a high level.

To illustrate our model, suppose there are four available time slots, 0, 1, 2, and 3 for five exams, E1, E2, E3, E4, and E5. The set of students taking each of the exams is as follows:

E1: {a, b, …, j}
E2: {k, l, …, z}
E3: {a, e, k}
E4: {b, c, d, x, y, z}
E5: {a, c, e, g, i, j}

Each edge in the graph shown in Figure 1 has the subset of students enrolled in both exams corresponding to the endpoints of that edge.

In general it may be undesirable to assign the same time slot (colour) to a given pair of exams for a variety of reasons. For this example, however, we consider two vertices to be adjacent only if there is at least one student taking both exams.

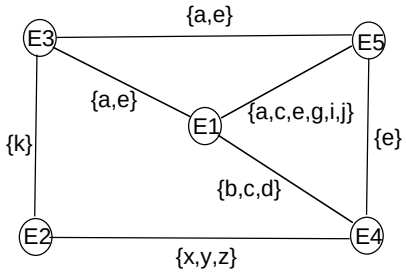

Figure 1: Student intersections for pairs of exams.

For our example, we set the conflict severity equal to 1, 5, or 25, according to the size of the intersection. In particular, we set the conflict severity to 1 if the intersection size is 1 or 2, to 5 if the intersection size is 3 or 4, and to 25 if the intersection size is 5 or greater (see Figure 2). We emphasize that these thresholds for conflict severity are arbitrarily chosen here. If a conflict-free timetable is a requirement, as it is in the University of Toronto problem instances (Carter, Laporte, and Lee 1996), then all

conflict severities can simply be set to one since all conflicts are regarded as equally bad.

Of course, as mentioned, there will be many situations in which the conflict severity depends on other factors. In these situations, an edge might exist even when it corresponds to an empty intersection of students.

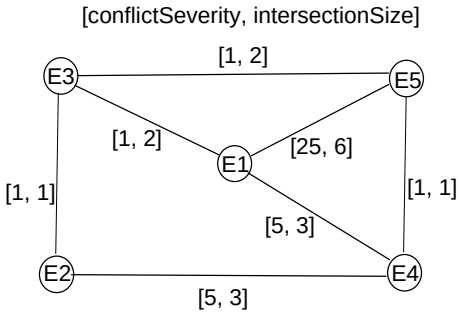

Figure 2: Additional edge attributes.

The proximity penalty of assigning colours c_i and c_j to the endpoints of an edge is a function of how close c_i and c_j are and the size of the intersection. For the Toronto problem instances, where the time slots are simply $c_i = i$, $i = 0,1,\ldots$, the intersection size is multiplied by a *proximity weight* that equals $2^{5-|i-j|}$ when $|i-j| \leq 5$ and 0, otherwise. Our implementation uses this same evaluation for comparison purposes with the Toronto benchmark results. However, if the time slots are specified by a day, a start time, and a duration, then our colour attributes can easily be modified to allow for the appropriate change in the proximity evaluation function.

Our overall objective is to produce colourings (timetables) with minimum total conflict (zero may be required) and minimum total proximity penalty.

Knowing the conflict severity and size of the intersection for each edge makes it straightforward to keep track of the two kinds of penalties as the colouring progresses. When a vertex gets coloured *c*, that colour becomes less desirable (or forbidden) to its neighbors, as do colours in proximity with colour *c*.

Our model keeps track of these two kinds of colour undesirability as follows. Each vertex *v* has a *colour-penalties vector* that indicates the undesirability of assigning each colour to that vertex with respect to conflict penalty and proximity penalty. That is, the component of the colour penalties vector corresponding to colour *c* has two values, one is the conflict penalty incurred if *v* is coloured *c*, and the other is the resulting proximity penalty.

Using our example and a simplified proximity function, we illustrate how the colour-penalties vectors change as the graph is coloured. Suppose that any two colours *i* and *j* of the colours 0, 1, 2, and 3 are *within proximity* if they differ by 1, then the proximity penalty incurred when the colours of the endpoints of an edge differ by 1 equals the intersection size. Suppose further that the colour-penalties vectors for all of the vertices are initialized with [0, 0] for all of their colour components. Figure 3 shows the result of colouring vertex E1 with colour 1.

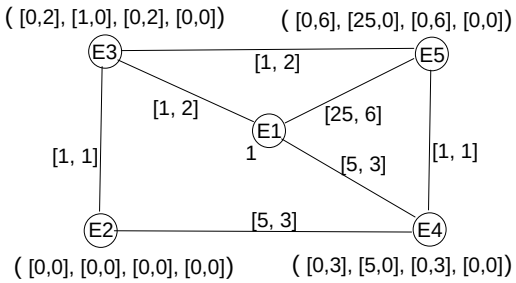

Figure 3: Colour-penalties vectors after E1 is coloured 1.

There may be other factors that make certain time slots undesirable for an individual exam. For instance, if professor X is assigned to exam A and cannot be on campus before noon. So any colour corresponding to a morning time slot for exam A would be given a prohibitively large conflict penalty value before the colouring begins.

As each vertex is coloured, its adjacent vertices' colour-penalties vectors are updated. The ease with which we are able to keep track of both hard and soft constraints as the colouring progresses creates new opportunities for the use of more sophisticated heuristics tied to this readily accessible information.

The Basic Approximate Algorithm

Our basic algorithm consists of two steps, select a vertex and then colour that vertex. We repeat these two steps until all vertices are coloured. Notice that while our model will easily accommodate more computation-intensive algorithms involving backtracking, local improvement, etc., we chose for this first phase of our research to concentrate on producing fast, essentially one-pass colourings.

Summary of the Model Features and Parameters

In preparation for the next section's discussion of heuristics, we list the key features and parameters on which the heuristics are based. The two edge attributes, conflict severity and intersection size, give rise to two different versions of the traditional concept of weighted degree of a vertex.

- *Conflict severity* (of an edge) – a measure of how undesirable it is to assign the same colour to both endpoints of the edge. In general, this would depend on several factors, and it could be set interactively by the end-user.
- *Intersection size* (of an edge) – the size of the intersection of the two sets corresponding to the endpoints of the edge. In exam timetabling, this is simply the number of students taking both exams.
- *Conflict degree* (of a vertex) – the sum of the conflict severities of the edges incident on the vertex.

- *Intersect degree* (of a vertex) – the sum of the intersection sizes of the edges incident on the vertex.
- *Bad-conflict edge* – an edge whose conflict severity exceeds a specified *threshold* value. If a conflict-free timetable (i.e., a feasible colouring) is required, then this threshold is set to zero, as we do for the Toronto problem instances.
- *Bad-intersect edge* – an edge whose intersection size exceeds a specified threshold. In our current implementation, this threshold is a function of the average of the intersection sizes of all edges; specifically, we use the average intersection size times some constant multiplier.
- *Conflict penalty* (for the colour assignment of a vertex) – a measure of how undesirable it is to assign that colour to the vertex. This will depend on the colour assignments of the vertex's neighbors and the conflict severities of the relevant edges, but it could also depend on other factors (e.g., professor, room, or equipment constraints).
- *Proximity* (of two colours) – a measure of how close together (in the case of exam timetabling) or spread apart (for course timetabling) the two colours are. This is often a secondary objective to optimize in school timetabling and is typically referred to as a *soft constraint.*
- *Proximity penalty* (for the colour assignment of a vertex) – the sum of the proximity penalties resulting from that colour assignment and the colour assignments of all neighbors of that vertex (determined by the function described immediately following Figure 2).
- *Colour-penalties vector* (of a vertex) – indicates for each colour the conflict penalty and proximity penalty of assigning that colour to the vertex. When a vertex is coloured, the colour-penalties vector of each of that vertex's neighbors must be updated accordingly.
- *Bad-conflict colour* (for a vertex) – a colour whose conflict penalty for that vertex exceeds some specified threshold (also set to zero for the Toronto instances since feasible colourings are required).
- *Bad-proximity colour* (for a vertex) – a colour whose proximity penalty for that vertex exceeds some specified threshold. Similar to the bad-intersect-edge threshold, we use average intersection size times a (possibly different) constant multiplier.

The thresholds for badness are easily adaptable to the requirements of the problem, and, in a decision support system, they could be specified by the end-user interactively. Part of this ongoing research is to study the effect that the values of the thresholds have on the quality of the solution and to identify features of a problem instance that determine that effect.

Heuristics

Vertex selection and color selection are the two key components of our simple, constructive algorithm, and our strategies for both are flexible in the varied ways they use new heuristics and variations of the traditional ones. Our current implementation uses 10 'primitive' heuristics for selecting the next vertex to be coloured and four to select a colour for that vertex.

Ten Primitive Vertex-Selection Heuristics

Our colouring strategies are based on the classical and intuitive idea that the most troublesome vertices should be coloured first. Some of the commonly used heuristics based on this idea have been largest saturated degree, largest degree, and largest weighted degree.

We use variations of these, and we introduce some new ones that focus more on the number of bad edges and the number of bad colours. Some of these new heuristics rely on the information kept in each vertex's colour-penalties vector, while others use information tied to the edges incident on each vertex. The primitive heuristics on which our vertex selectors are based are:

0. *Maximum number of bad-conflict edges to uncoloured neighbors* – vertices having the most bad-conflict edges among their incident edges to uncoloured neighbors.
1. *Maximum number of bad-conflict colours* – vertices having the most bad-conflict colours. For the Toronto data set, this heuristic reduces to largest saturation degree.
2. *Maximum number of bad-proximity colours* – vertices having the most bad-proximity colours.
3. *Maximum conflict sum* – vertices with the largest sum of their conflict colour penalties.
4. *Maximum proximity sum* – vertices with the largest sum of their proximity colour penalties.
5. *Maximum conflict degree to uncoloured neighbors* – vertices whose incident edges to uncoloured neighbors have the largest sum of the conflict severities.
6. *Maximum number of bad-conflict edges* – vertices having the most bad-conflict edges among their incident edges. For the Toronto data set, this reduces to largest degree (since *every* edge is considered a bad-conflict edge).
7. *Maximum number of bad-intersect edges to uncoloured neighbors* – vertices having the most bad-intersect edges among their incident edges to uncoloured neighbors.
8. *Maximum intersect degree to uncoloured neighbors* – vertices whose incident edges to uncoloured neighbors have the largest sum of the intersection sizes.
9. *Maximum number of bad colours* – a consolidation of heuristics 1 and 2; a bad colour is one whose conflict penalty or whose proximity penalty exceeds its respective threshold.

Observe that heuristic 7 may be better at evaluating the difficulty of a vertex than its sum counterpart, heuristic 8. To illustrate, suppose that the edge weights in Figure 4 represent intersection size and that all neighbors of vertices v1 and v2 are uncoloured. Then heuristic 8 would select v1, whereas, for any bad-intersect-edge threshold greater than one, heuristic 7 would select v2, which ap-

pears to be more difficult. A similar observation can be made for heuristic 2 versus heuristic 4.

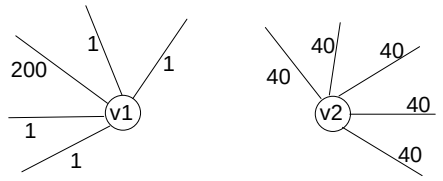

Figure 4: Heuristic 8 would select v1 before v2.

Four Primitive Colour-Selection Heuristics

Given a vertex v that has been selected, the primitive heuristics that we use to choose a colour for v are:

0. *Minimum conflict penalty* – a colour that has minimum conflict penalty for vertex v.
1. *Minimum proximity penalty* – a colour that has minimum proximity penalty for vertex v.
2. *Least bad for neighbors with respect to conflict penalty* – a colour which when assigned to v causes the fewest good-to-bad conflict penalty switches for the uncoloured neighbors of v.
3. *Least bad for neighbors with respect to proximity penalty* – same as heuristic 2 but with respect to proximity penalty.

Combining Heuristics

One of the innovations of our model and implementation is the ability to combine any number of the primitive heuristics to form compound vertex selectors and compound colour selectors. A compound vertex selector starts with one of the 10 primitive vertex-selection heuristics listed above. Typically there will be several vertices identified as the most difficult with respect to that heuristic. This subset of vertices is then narrowed down by applying a second primitive heuristic, and so on. Thus, a compound vertex selector consists of a sequence of primitive heuristics, where all but the first one in the sequence, is regarded as a tiebreaker for the ones before it. Once the subset of vertices is pared down by the combination of heuristics, some vertex is chosen from the subset (typically the first one in the list). Compound colour selectors are similarly constructed from the four primitive colour-selection heuristics listed above.

Switching Selectors in the Middle of a Coloring

Another feature of our model is the ability to switch from one combination of heuristics to another at various stages of the colouring. Including this feature was motivated by the general observation that the effectiveness of a heuristic is likely to change as the colouring progresses. The primitive vertex-selection heuristic 1 is perhaps the simplest illustration of this behavior. As we mentioned earlier, this heuristic is essentially the traditional saturation degree, which has proven to be among the most preferred heuristics for classical graph colouring. However, applying heuristic 1 in the very early stages of a colouring will produce a huge number of ties. Moreover, early in a colouring, the only vertices with any bad-conflict colours will tend to be those few that have neighbors that have already been coloured. Thus, until several vertices are coloured, the order in which they are selected will tend toward a simple breadth-first order and not be an effective predictor of the difficult-to-colour vertices.

Accordingly, the compound vertex selectors used early in the colouring process begin with a primitive heuristic based on the weights of incident edges (e.g., heuristic *0*). Then, after a designated number of vertices have been selected and colored, we switch to a compound selector that begins with heuristic 1 when it is more likely to be a stronger predictor of the difficulty of a vertex.

Vertex Partitioning

A final innovation involves a preprocessing step that partitions the vertex set and allows us to reduce the amount of computation without incurring additional conflict penalties. The preprocessing is based on the following simple observation. If *v* is a vertex with degree less than *k*, and *v* initially has *k* colours available, then *v* can safely be left until last to colour, since it will always have at least one non-conflict colour available, independent of how its neighbors are coloured and how heavy the edge-weights are between *v* and its neighbors.

The preprocessing uses an iterative partitioning algorithm that places all vertices whose colouring can be done last into the easiest-to-colour subset, say S_1. Next, for each vertex in S_1, we calculate a reduced (quasi-) degree of each of its neighbors and put all vertices whose reduced degree is less than the number of colours available into the next-easiest-to-colour subset, S_2. Again, as long as a vertex in S_2 is coloured before any of its neighbors in S_1, it can safely be left uncoloured until its other neighbors are coloured. The process continues until no additional vertices can be removed from the 'hardest' subset and the vertices in that last subset of the partition must be coloured first using the specified selection criteria.

As long as the subsets are done in order (last to first), vertices in all subsets except for the hardest one can be selected arbitrarily with no possibility of incurring a conflict penalty. One simply chooses an available colour, whose existence is guaranteed by the construction. Thus, in a fairly sparse graph, computation can be considerably reduced. Notice that because any penalties that result from the colouring occur in the process of colouring the hardest cell, any local improvement algorithms could be applied only to that set of vertices before moving on to colour the rest of the graph, again without incurring additional penalties at a later stage.

Another potential advantage to this partitioning strategy is that the vertex-selection process after the hardest subset has been coloured can be based solely on proximity considerations.

Some Preliminary Results

We present the preliminary results of applying our approach on the Toronto benchmarks, which is available at ftp://ftp.mie.utoronto.ca/pub/carter/testprob/. This dataset was first introduced in (Carter, Laporte, and Lee 1996), and since then has been extensively studied using a wide range of algorithms in the literature. We set the number of colors equal to the number of time slots in the Toronto dataset. Due to the fact that two versions of the datasets have been circulated under the same name in the last ten years, we have renamed the problems in (Qu et al. 1996). We used version I of the data in our experiments.

Testing is ongoing and much more needs to be done. However, we can make some initial observations.

Table 1 presents the best results we have obtained so far.

Although we haven't fully tested it yet, partitioning appears to improve solution quality most of the time. Except for the "sta83 I" problem instance, all results in column 2 of the table were produced using the partitioning pre-processing.

We obtained them using the following two groups of three compound vertex selectors:

vs1: 0 7 8 1 2 4 | 1 0 2 4 7 8 | 2 4 7 8
vs2: 0 7 8 9 4 | 9 0 7 8 2 4 | 2 4 7 8

The numbers refer to the primitive vertex-selection heuristics introduced earlier, and the vertical lines separate the three compound selectors that form each group. The first compound selector in a group is applied to the hardest subset until a designated fraction (the *switch fraction*) of the vertices have been selected and coloured. Then the second compound selector is applied to the rest of the hardest subset. Finally, the third selector, which consists of the four proximity-related primitive heuristics, is applied to the remaining (non-hard) vertices.

We used the following two groups of two compound color selectors:

cs0: 0 1 2 3 | 0 1 3
cs1: 0 2 3 1 | 0 3 1

The first compound selector in each group was applied to the entire subset of hardest-to-color vertices, and the second one was applied to the rest of the vertices.

As we described earlier, the thresholds for a bad-proximity color and a bad-intersect edge were set equal to the average intersection size times two different constant multipliers. In the table, PC is the multiplier for the bad-proximity color, and IE is the one for the bad-intersect edge.

The Settings column gives the values of the switch fraction and the multipliers, PC and IE, and indicates the vertex and color selectors used to produce the given result.

Problem	Best results	Settings switch | PC | IE | vs | cs	Best reported
car91 I	5.22	1/23 | 90 | 1 | vs2 | cs0	4.97
car92 I	4.40	1/13 | 126 | 2 | vs2 | cs0	4.32
ear83 I	39.28	1/5.2 | 115.5 | 1,2 | vs2 | cs0	36.16
hec92 I	12.35	1/5 | 16 | 1,2 | vs1 | cs0	10.8
kfu93 I	19.04	1/14 | 134 | 1,2 | vs2 | cs0	14.0
lse91	12.05	1/32 | 192 | 1,2 | vs2 | cs0	10.5
rye92	10.21	1/28 | 133.5 | 2 | vs2 | cs0	7.3
sta83 I	163.05	1/26.5 | 81 | 1 | vs2 | cs1	158.19
tre92	8.62	1/39 | 207 | 20 | vs2 | cs0	8.38
uta92 I	3.62	1/16 | 50 | 1,2 | vs1 | cs0	3.36
ute92	30.60	1/5 | 369 | 1,2 | vs2 | cs1	25.8
yor83 I	42.05	1/17 | 340 | 2 | vs2 | cs0	39.8

Table 1. Best results with the corresponding settings for Toronto benchmarks.

Results from Table 1 demonstrate that for vertex selection, vs2 outperforms vs1; 10 of the 12 best results were achieved using vs2. Changing threshold values for badness and changing the switch point between the first and second compound vertex selector clearly affect the performance of our algorithm.

In Table 1, we also gave the best results reported in the literature which used different constructive methods. Although our totals for proximity penalty are, on the average, 13% worse than the best ones reported, we believe our approach still holds promise, particularly in view of the fact that it is, at the moment, a one-pass algorithm without any backtracking or local improvement. The best results reported in the last column were by different approaches cited in the literature. No single algorithm outperformed others on all problems tested here.

In general, these preliminary results indicate that the performance of the algorithm is sensitive to the settings of the switch points and thresholds. Although we have some initial observations on which settings perform better on which Toronto problems, the setting of these parameters in relation to particular problems is not clear. More research effort needs to be spent to develop more intelligent mechanisms to adaptively choose these settings for different problems.

One of our future directions is to use ***heuristics*** to choose how to construct the combinations of heuristics. This ***hyper-heuristic*** approach (see Burke et al. 2003) has been applied successfully in a range of scheduling and optimization problems, including timetabling. It is well known in meta-heuristics research that different heuristics perform better on different problems, or even different instances of the same problem. One of the research challenges is concerned with the ***automatic*** design of heuristics in solving a wider range of problems. Developing an automatic algorithm that can intelligently operate on a search space of vertex and colour selectors,

switch point selectors and threshold settings will become one of our primary research efforts in the future.

Features of the Model Not Being Used Yet

There are some features of our model not used in our current implementation that add to its robustness.

Our model can handle *pre-colored* vertices, that is, exams that must be assigned to certain time slots. Furthermore, if certain time slots are forbidden for a particular exam (for example, the professor is only available on certain days and times), then this can easily be handled by setting an initial nonzero penalty for the relevant color.

As we noted earlier, each color, which represents a time slot, can have attributes associated with fairly general information, like start time, duration and/or finish time. For this paper we used only a single attribute, an integer value between zero and the maximum number of time slots in use, since we were testing our implementation on the Toronto benchmark problems.

Ongoing and Future Work

The robust model presented in this paper can be easily extended or integrated with other techniques to develop more advanced and powerful algorithms. We give below some possible (and ongoing) research directions.

- Study the effects of varying the switch points, the badness threshold values, and the use of different heuristic combinations. In the context of *hyper-heuristics*, there are a number of different search spaces to consider:
  - The set of all the combinations of one or more of the primitive vertex selectors and of the color-selectors.
  - For a given group of compound vertex selectors, the set of all switch points.
  - For a given group of compound vertex selectors, the set of threshold values for badness.
- In the context of *case-based reasoning*, test heuristic combinations, thresholds, and switch points with randomly generated problem instances that are in the Toronto format to see if certain performance patterns emerge. Previous work on using case-based reasoning (see Burke, Petrovic and Qu, 2006) to intelligently select graph colouring heuristics demonstrated that there are significant, wide-ranging possibilities for research in knowledge-based heuristic design.
- Adding a backtracking component to the algorithm is likely to lower the total proximity penalty. For instance, when every colour assignment for a selected vertex incurs a proximity penalty above some threshold, the algorithm un-colours or re-colours some other vertex in order to reduce the selected vertex's proximity penalty.
- Write an improvement method that takes a given colouring produced by our algorithm and looks for vertices whose colours can be changed to decrease the total proximity penalty while maintaining feasibility.
- With the current implementation, we have not yet made full use of the varying conflict severity of edges, nor have we allowed any trade-off between conflict penalty and proximity penalty. In timetabling situations where conflicts must be tolerated, the end-user might specify that a certain amount of conflict penalty is equivalent to a certain amount of proximity penalty, e.g., a proximity violation involving 50 students equals a conflict involving one student. This might lead naturally to a single objective function to be minimized.
- As we mentioned at the start, the model can be adapted to a variety of scenarios, in which a number of parameters would be specified interactively by the end user through an appropriate interface. Follow-up work will include building such an interface.

Acknowledgements

The research for this paper was supported by Nottingham University, UK, the Engineering and Physics Science Research Council (EPSRC), UK, and an Ashforth Grant from Rollins College, USA.

References

Broder, S., Final Examination Scheduling, Comm. of the ACM 7 (1964), 494-498.

Brelaz, D., New methods to color the vertices of a graph. Comm. of the ACM 22 (1979), 251-256.

Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S.: Hyperheuristics: an Emerging Direction in Modern Search Technology. In: Glover, F. and Kochenberger, G.: Handbook of Metaheuristics, 457-474, 2003.

Burke, E. K., Kingston, J. H., and de Werra, D., Applications to Timetabling, In: J. L. Gross and J. Yellen (eds.) The Handbook of Graph Theory, Chapman Hall/CRC Press, (2004), 445-474.

Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R.: A Graph-Based Hyper Heuristic for Timetabling Problems. European Journal of Operational Research, 176 (2007) 177-192.

Burke, E.K., Petrovic, S., and Qu R., Case Based Heuristic Selection for Timetabling Problems. Journal of Scheduling, 9 (2006) 115-132.

Carter, M. W., A Survey of Practical Applications of Examination Timetabling Algorithms, Operations Research 34 (1986), 193-201.

Carter, M. W., Laporte, G., and Lee, S., Examination Timetabling: Algorithmic Strategies and Applications, J. of the Operations Research Society 47 (1996), 373-383.

de Werra, D., An Introduction to Timetabling, Euro. J. Oper. Res. 19 (1985), 151-162.

Kiaer, L., and Yellen, J., Weighted Graphs and University Timetabling, Computers and Operations Research Vol. 19, No. 1 (1992a), 59-67.

Kiaer, L., and Yellen, J., Vertex Coloring for Weighted Graphs With Application to Timetabling, Technical Report Series – RHIT, MS TR 92-12 (1992b).

Krarup, J., and de Werra, D., Chromatic Optimisation: Limitations, Objectives, Uses, References, Euro. J. Oper. Res. 11 (1982), 1-19.

Mehta, N. K., The Application of a Graph Coloring Method to an Examination Scheduling Problem, Interfaces 11 (1981), 57-64.

Neufeld, G. A. and Tartar, J., Graph Coloring Conditions for the Existence of Solutions to the Timetable Problem, Comm. of the ACM 17 (1974), 450-453.

Papadimitriou, C. H. and Steiglitz, K., Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, 1982.

Qu, R., Burke, E.K., McCollum, B., Merlot, L. T. G., and Lee, S. Y., A survey of Search Methodologies and Automated Approaches for Examination Timetabling, Technical Report, NOTT-CS-TR-2006-4 (2006).

Schaerf, A., A Survey of Automated Timetabling, Artificial Intelligence Review 13 (1999), 87-127.

Schmidt, G., and Strohlein, T., Timetable Construction--an Annotated Bibliography, The Computer Journal 23 (1980), 307-316.

Welsh, D. J. A., and Powell, M. B., An Upper Bound for the Chromatic Number of a Graph and its Application to Timetabling Problems, The Computer Journal 10 (1967), 85-86.

Wood, D. C., A System for Computing University Examination Timetables, The Computer Journal 11 (1968), 41-47.