Improving a Lecture Timetabling System for University-Wide Use

Ben Paechter, R. C. Rankin, and Andrew Cumming

Department of Computing
Napier University
219 Colinton Road
Edinburgh
Scotland
EH141DJ
benp@dcs.napier.ac.uk

Abstract. During the academic year 1996/97 the authors were commissioned by their institution to produce an automated timetabling system for use by all departments within the Faculty of Science. The system had to cater for the varying requirements of all the departments, be easy to use, robust, expandable, and timetable 100% of events fully automatically within a reasonable time. The timetables produced had not only to be workable, but also had to be 'good' with respect to management defined criteria. The work was intended as a pilot study for later extension to the whole institution. This paper describes the enhancements to the user interface and timetabling engine that were found to be necessary to meet the more extensive needs of a faculty.

1. User Interface

Specifying timetabling constraints can be difficult for anyone, and often this task is to be completed by administration staff with little knowledge of mathematics or computing. It is important therefore, that the user interface is easy to use, and that the concepts employed related to the real world rather than to abstract mathematical structures and operations.

In designing the interface some concepts were defined to help users to specify the constraints on the timetable.

1.1 Features

A *feature* is a property *satisfied* by some resources or events in the system and *required* by others. For example certain rooms may be "accessible" by those with mobility problems. The room is made to *satisfy* a feature called "access" that is *required* by a student-group. The constraint being represented here is that any event that the student group is required to attend can only be allocated to a room possessing the "access" feature. Another constraint might be that a certain room is reserved for senior students. This could be represented by making the room *require* the feature "senior" and the senior student groups *satisfy* it.

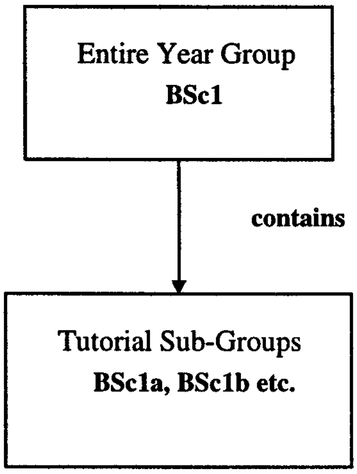

Fig. 1. The *contains* relationship for groups.

There is no restriction on what can be represented as a feature; the most common use in a computing department is to manage rooms according to function and equipment and software availability. Features need not be singular: a room may satisfy or require several features. For example, a pair of computer laboratories for data-communications classes may have the same equipment and software but one may be more accessible than the other. Both would have the feature "comms", but only the latter would also have the feature "access". Events requiring "comms" could be placed in either laboratory, but "comms" events that also required "access" could only be placed in the latter.

1.2 Containers

A *container* is a resource that can hold other similar resources, including other containers. The *contains* relationship can be seen as a *parent-child* relationship giving us a directed graph. If a particular resource is busy then all its descendants are also busy along with all of the descendant's ancestors.

This has been applied to student-groups and rooms. A common example is show in Figure 1, where a student group sub-divides for tutorial classes or practical classes.

The container concept allows for flexible use of rooms, especially when different features are associated with each level in the container hierarchy. For example, a general purpose classroom may have the feature "GP-room" and a nearby computer laboratory may have the feature "PC-lab". Although there is no physical relationship between these rooms, a container may be created that encompasses both and has the abstract feature "GP+PC". Any events of a subject or module that need flexible access to both types of accommodation will require the feature "GP+PC". These events will be timetabled into the logical container room making both physical rooms busy at that time. The composite room becomes unavailable when either of its components is in use.

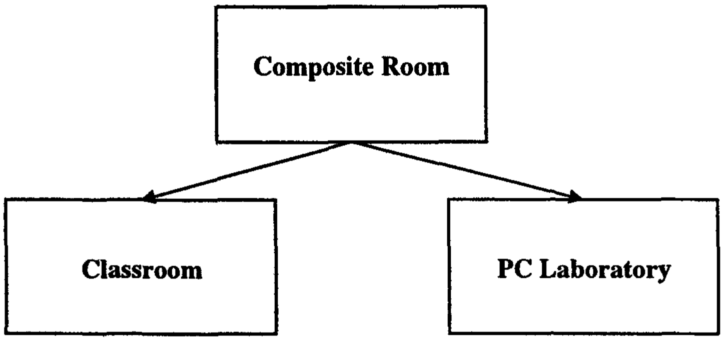

Fig. 2. The *contains* relationships for rooms

The container relationship can also be applied to rooms with moveable partitions which allows the various components, or the whole room, to be used at will but prevents booking of one element causing clashes with others. There is no limit to the depth of the container tree and its branches may be unbalanced.

A novel use of the container function is to deal with rooms of variable capacity. For example a PC laboratory may have 16 computers. For one class it may be desirable to have one student per machine, while for another, students may benefit from working in pairs. One room is created with size 16 and the feature "PC-lab" and another with size 32 and the feature "crowded-PC-lab". One room is made to contain the other. Events that allow one student to each machine will require a "PC-lab" and those that allow two students per machine will require a "crowded-PC-lab". As before if the parent is busy the child is unavailable and vice-versa, so clashes do not occur.

A more specialised use of the container facility occurs where an event requires use of different types of accommodation at different times in the semester. For example, if a room is only required by an event on odd weeks of a semester that event can be made to require "odd-ness". A physical room is made to contain two identical rooms, one with the feature "odd-ness" and the other with the feature "even-ness". Because siblings can be assigned independently, the allocation of the "odd" room to an event does not preclude the use of its sibling by an event requiring the "even" room. This technique is unnecessary where the system deals with separate timetables for each week, but is of importance where the greatly increased complexity and processing load of supporting weekly timetables needs to be avoided.

2. Evolutionary Engine

The evolutionary engine employs a memetic algorithm using an indirect representation, heuristic seeding, directed mutation and targeted mutation. This algorithm is described in detail in [1]. Changes to the algorithm, and further results are given below.

2.1 Room Assignment

In previous versions of the algorithm, genetic representations encoded information about the timeslot that each event should take place in, and rooms were allocated using a greedy algorithm which allocated the best fitting room available to each event. However, new evaluation criteria concerning the movement of people around and between buildings has made it necessary to extend this genetic representation to include information about the room in which an event should take place.

The specified room for an event is chosen from an ordered list of possible rooms. A room is deemed "possible" if the combination of event, student groups, lecturers and that room, leaves all the required features of any of them satisfied by at least one of them, and the room capacity is greater than the combined sum of the sizes of the student groups attending. The list is ordered by a heuristic that takes into account features that the room satisfies that are not required, and unused capacity of the room. This helps to ensure that events do not claim rooms that are bigger or satisfy more features than are necessary.

When evaluating a chromosome, if the room specified for an event is not available (because it has been used by some other event), then a search through the ordered list of possible rooms is conducted. If a suitable room is found, then that room can be written back to the genetic representation.

2.2 Local Search Results

There has been some debate as to the need for local search (memetics) in an evolutionary approach to timetabling. Local search can be computationally very expensive and it could be argued that the advantages of local search are outweighed by the disadvantage of processing far fewer chromosomes in the same time. Some methods do not use local search e.g. [2], some use local search throughout e.g. [1], [3], others use local search to seed an initial population e.g. [4].

In order to explore this issue, experiments were conducted using different amounts of local search. The data for these experiments was that for the Computer Studies Department at Napier University. In one experiment local search was always used, in another in was only used to seed the initial population, and in other experiments local search was used when evaluating a randomly selected percentage of chromosomes. Each experiment was given the same elapsed time to run, and so experiments using less local search were able to perform a significantly larger number of evaluations. The results are given in Figure 3 and show the weighted sum of penalty points over 12 competing objectives, averaged over 25 runs. These results show that significant improvement can be made to the algorithm by including local search not just at the population seeding stage, but throughout the whole algorithm.

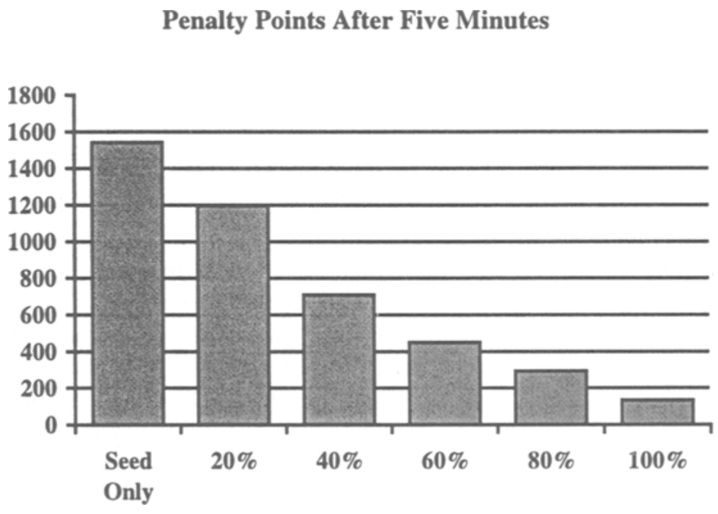

Fig. 3. Effect of varying the amount of local search

Some systems, such as that described in [5], use a method known as delta evaluation. This method cuts the processing necessary for timetable evaluation by re-evaluating only the parts of the timetable that are changed through recombination or mutation. Using local search throughout the evolution makes it impossible to efficiently implement delta evaluation. While the authors expect that delta evaluation cannot compensate for the lack of local search, it should be noted that experiments have not been conducted to examine this.

2.3 Lamarckism

If local search is used then the results of this can be written back into the chromosome giving Lamarckian evolution. There is some debate as to the relative benefits of Lamarckian evolution compared to simply making use of the Baldwin Effect (using local search but not writing the results back) [6].

Experiments were conducted, in a similar fashion to those above, to measure the effect of varying the percentages of chromosomes that have the results of local search written back to them. It should be noted that even in the experiments where no direct writeback occurs, there is still a Lamarckian effect, since the directed and targeted mutation operators have a Lamarckian nature. The results given in Figure 4 show that for this real world problem Lamarckism has a significant advantage over use of the Baldwin Effect. As soon as some writing back is done there is a big improvement. Increasing the amount of writeback does not have a large effect because if there is some writeback in the system then each chromosome will eventually have its local search results written back even if it has to wait some generations for this.

The main reason that writing back is important is that it decreases epistasis in the chromosomes. The local search considers each of the events in turn and tries to place the event in the timeslot and room specified by the chromosome. If the timeslot and room are not available then others are tried. Without writeback, the timeslot and room used by an event at the last evaluation are not necessarily the ones specified in the chromosome. This means that choosing the same timeslot and room is dependent on the slots and rooms tried earlier being unusable - i.e. the resources they need have already been used by some event considered earlier. So the timeslot and room for an event is dependent on the slots used by previously considered events. If this event gets a different slot then so might others considered afterwards. Writing back the results of local search avoids this, so decreasing epistasis in the chromosomes, making then less brittle.

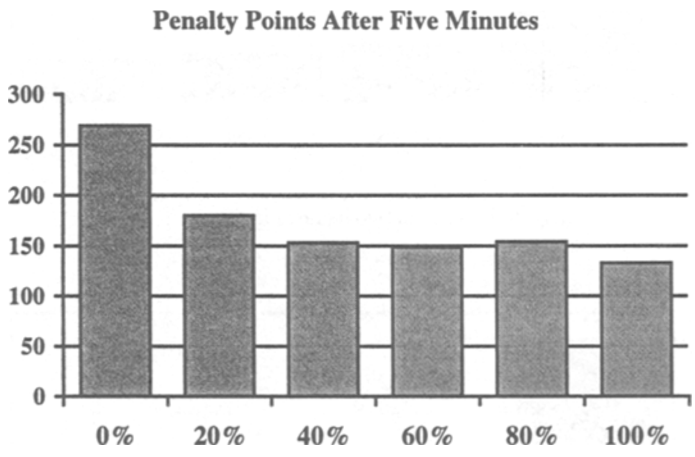

Fig. 4. Effect of varying the amount of writeback

3. Evaluation Criteria and Results

The Napier University Science Faculty, was timetabled using 12 competing objectives in addition to the objective of placing all events. The results obtained from a typical run are shown in Figures 5-16. Results show penalty points for each evaluation criterion are given for run time of 10 minutes, 1 hour and 12 hours on a Pentium Pro 200 computer system, along with the figures for the manually produced timetable.

It can be seen from the results that the pilot study was successful. The system produces feasible timetables in a few seconds. If the system is left to run overnight then timetables can be produced that are better than manually produced timetables in all measured criteria.

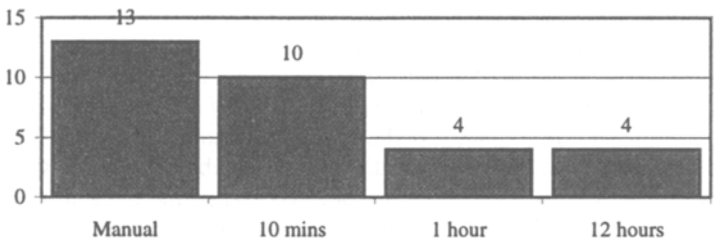

Fig. 5. Lecturers without a teaching free day.

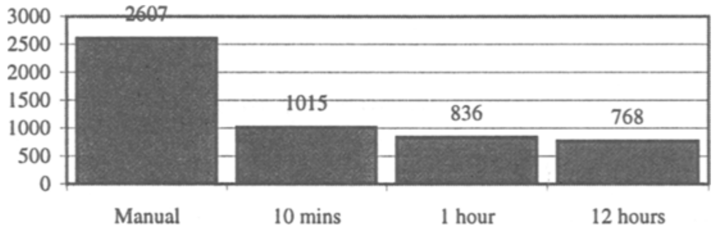

Fig. 6. More than two lectures in a row

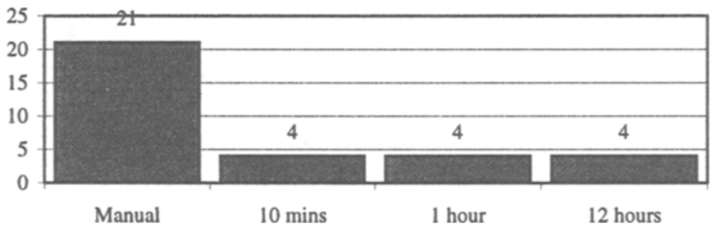

Fig. 7. More than three hours class contact in a row

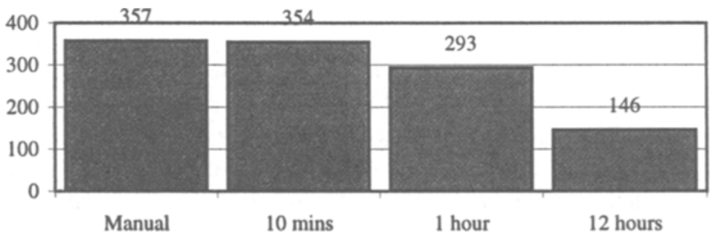

Fig. 8. Gaps of more than three hours in a student's day.

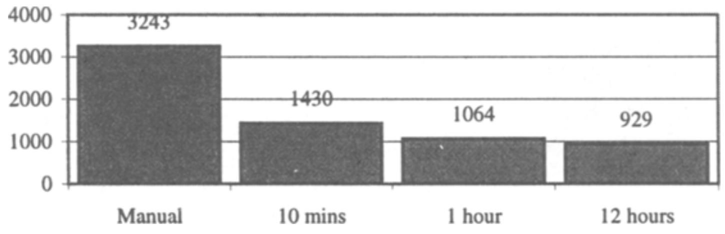

Fig. 9. Wednesday Afternoon Classes.

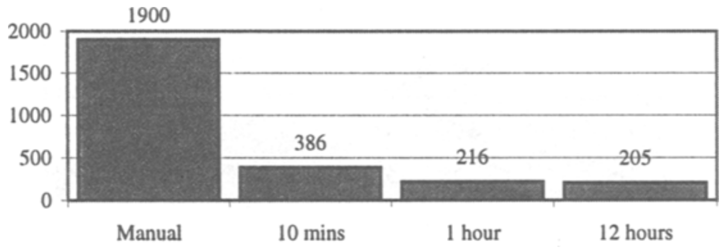

Fig. 10. Five o'clock classes.

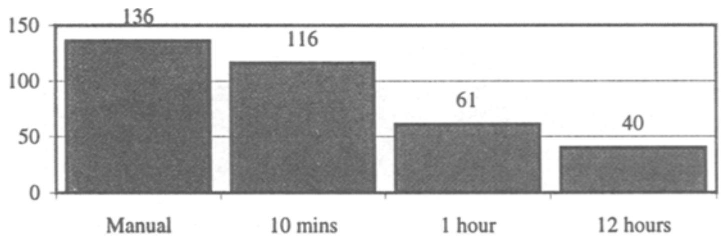

Fig. 11. Single classes on a student's day.

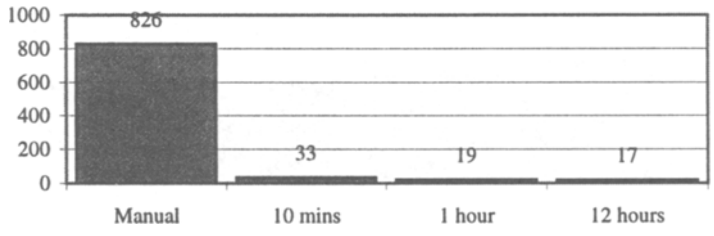

Fig. 12. Site changes during the day.

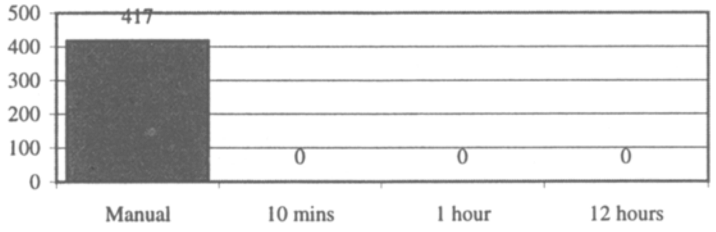

Fig. 13. Instantaneous site changes during the day.

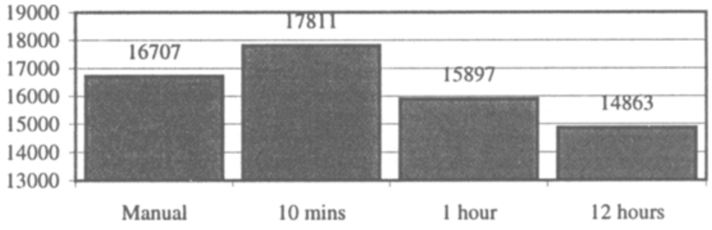

Fig. 14. Location changes within a site.

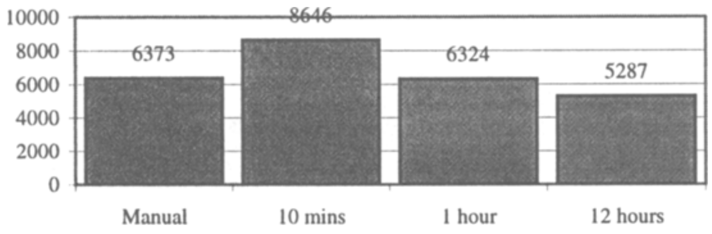

Fig. 15. Room changes within a location.

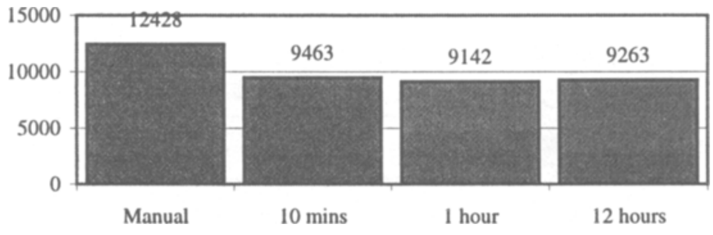

Fig. 16. Seat wastage.

4. Conclusions

We have defined two concepts, *features* and *containers* and shown how they can be used to help define constraints in timetabling problems. We have shown that the policy of using a memetic (local search) algorithm with Lamarckian evolution work well for this problem.

A system has been produced which is easy to use, quickly produces feasible timetables, and which given an overnight run can be produce timetables that are better than those manually produced in all measured criteria.

References

[1] Paechter B., Cumming, A., Norman, M.G. and Luchian, H., "Extensions to a Memetic Timetabling System", in Practice and Theory of Automated Timetabling, Eds., Burke, E. and Ross, P., Springer-Verlag Lecture Notes in Computer Science 1153, Berlin 1996.

[2] Corne, D., Ross, P. and Fang, H, "Fast Practical Evolutionary Timetabling", in Evolutionary Computing, Ed. Fogarty, T, Springer-Verlag Lecture Notes in Computer Science 865, Berlin 1994.

[3] Burke, E. Newall, J.P., and Weare, R.F., "A Memetic Algorithm for University Exam Timetabling", in Practice and Theory of Automated Timetabling, Eds., Burke, E. and Ross, P., Springer-Verlag Lecture Notes in Computer Science 1153, Berlin 1996.

[4] Corne, D and Ross, P., "Peckish Initialisation Strategies for Evolutionary Timetabling", in Practice and Theory of Automated Timetabling, Eds., Burke, E. and Ross, P., Springer-Verlag Lecture Notes in Computer Science 1153, Berlin 1996.

[5] Ross, P., Corne, D. and Fang, H., "Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation" in Parallel Problem Solving from Nature ii, Ed. Davidor, Y., Springer-Verlag, Berlin 1994.

[6] Turney, P, Whitley, D. and Anderson, R. "Evolution Learning and Instinct: 100 Years of the Baldwin Effect" in Evolutionary Computation, Volume4, Number 3, MIT Press 1996.