Hierarchical Timetable Construction

Jeffrey H. Kingston

School of Information Technologies,
The University of Sydney, NSW 2006, Australia
jeff@it.usyd.edu.au
http://www.it.usyd.edu.au/∼jeff

Abstract. A hierarchical timetable is one made by recursively joining smaller timetables together into larger ones. Hierarchical timetables exhibit a desirable regularity of structure, at the cost of some limitation of choice in construction. This paper describes a method of specifying hierarchical timetables using mathematical operators, and introduces a data structure which supports the efficient and flexible construction of timetables specified in this way. The approach has been implemented in KTS, a web-based high school timetabling system created by the author.

1 Introduction

The basic timetable construction problem is to assign times and resources (students, teachers, rooms, etc.) to a set of meetings so that the resources have as few timetable clashes as possible. To this basic problem many other constraints are typically added, such as that the times allocated to a meeting be spread evenly through the week, that workload limits placed on some resources not be exceeded, and so on. Timetable construction is an NP-complete problem with an extensive literature [4,5,6,7,8].

Informally, a *regular timetable* is one in which a pattern may be discerned which makes the timetable easy to understand and remember. Regularity may take many forms, but this paper will be chiefly concerned with regularity in the choice of times. For example, North American universities commonly require all courses to occupy three hours per week, offered in one of the sets of time slots Mon–Wed–Fri 9–10AM, or Mon–Wed–Fri 10–11AM, and so on, producing a very regular timetable.

Even when such a strict rule as this is not possible, still some regularity might be achievable, perhaps by attempting to minimize the number of pairs of meetings that share at least one time, in addition to the usual objectives.

Regular timetables are easy to assign resources to. For example, in the North American university system, each meeting can meet in the same room for all three of its times. This point is particularly significant in high school timetabling, the area of timetabling which has inspired this paper, since teachers are assigned as well as rooms. Teacher assignment is the main area where the author's previous work in high school timetabling [10,13] is deficient. Thus, regularity is more than just an aesthetic consideration.

E.K. Burke and H. Rudová (Eds.): PATAT 2006, LNCS 3867, pp. 294–307, 2007.

This paper introduces a method of specifying regular timetables hierarchically, using *timetable expressions* analogous to algebraic expressions. This seems to be the first paper to find a use for arbitrarily deep hierarchies, although two-level hierarchies have been used occasionally. For example, Fizzano and Swanson [11] group together pairs of meetings, where one occurs on Mon–Wed–Fri and the other on Tue–Thu, and Adriaen et al. [1] aggregate sets of university meetings that occur in disjoint weeks of the semester.

As will be seen, the particular kind of hierarchical specification introduced in this paper imposes significant hard constraints on the times assigned to the meetings of the hierarchy. This is appropriate for a method used in high school timetabling, where the constraints tend to be relatively hard because entire classes of students suffer under any deficiencies, rather than individual students as in, for example, examination timetabling.

One way to handle these constraints would be to express them in a general-purpose constraint programming language. A number of papers have taken this approach [9,16]. However, this paper takes the special-purpose route, introducing a data structure, the *layer tree*, which represents timetable expressions and efficiently supports a particular set of constraints relevant to timetabling. The special-purpose route, while costly in development time, has some advantages. In particular, some of the algorithms used here, for example weighted bipartite matching, do not seem to be available in any existing constraint programming system [3,17], although some recent research into the *all_different* constraint [12], which implements unweighted bipartite matching, is a step in that direction.

Our focus is on the efficient implementation of some basic assignment and deassignment operations (and the resulting constraint propagation), rather than their use with any particular timetable construction algorithm. If these operations are efficient, many algorithms, including construction heuristics, tree searches, and local searches, become available. Although efficiency is a key goal, it has not been considered useful to report running times, since the operations to be presented are all polynomial time, and running times say more about the algorithms built on these operations than the operations themselves.

Layer trees are particularly effective when sets of meetings can be identified that must be disjoint in time. In high school timetabling, each set of meetings attended by a given student group satisfies this condition. This author's KTS timetabling system [14], a free, public web site for high school timetabling, uses layer trees. KTS typically produces a good timetable in about ten seconds [15], showing that layer trees can support practical timetabling.

The algorithms used here have appeared in previous timetabling work by the author and others [10,13,18]. This paper's contribution is to show how these algorithms can be incorporated into a flexible, efficient, hierarchical constraint framework. Section 2 introduces timetable expressions, and Section 3 introduces the layer tree data structure. Section 4 analyses the problem of efficiently propagating constraints related to time through this data structure as assignments and deassignments occur, and Section 5 does the same for resource constraints. Section 6 surveys some other, less fundamental features implemented in KTS.