Specifying and Solving Minimal Perturbation Problems in Timetabling

Jeffrey H. Kingston

Abstract For each kind of timetable construction problem (university, high school, etc.) there is a corresponding *minimal perturbation* problem, which is the problem of adapting an existing timetable to some unexpected changes in requirements while minimizing changes to the timetable. Minimal perturbation problems are important when the requirements may change after the timetable is published, notably in university course timetabling. This paper presents a simple method of specifying minimal perturbation problems which allows arbitrary changes to requirements, yet avoids a tedious enumeration of all types of changes. The paper also simplifies solving, by showing how some repair algorithms for the construction problems can be applied with only minor changes to the corresponding minimal perturbation problems.

Keywords Timetabling · Minimal perturbation

1 Introduction

For each kind of timetable construction problem (university, high school, etc.) there is a corresponding *minimal perturbation* problem, which is the problem of adapting an existing timetable to some unexpected changes in requirements while minimizing changes to the timetable. Minimal perturbation problems are important when the requirements may change after the timetable is published (notably in university course timetabling), since changes are disruptive.

This paper presents a simple method of specifying minimal perturbation problems which allows arbitrary changes to requirements, yet avoids a tedious enumeration of all types of changes. The paper also simplifies solving, by showing how some repair algorithms for the construction problems can be applied with only minor changes to the corresponding minimal perturbation problems.

J. Kingston
School of Information Technologies, The University of Sydney, Australia
E-mail: jeff@it.usyd.edu.au

Minimal perturbation problems in timetabling have been recognized for many years. They were mentioned in the call for papers of the early PATAT conferences, at this author's initiative. But their literature is tiny, and the author knows of no previous work that addresses them generally.

The best-known previous work is that of the UniTime group [3,6]. It solves the minimal perturbation problem for university course timetabling, within a constraint programming framework. Another early paper [4] is similar.

In unpublished work from the 1990's by colleagues of the author's at the University of Sydney, intelligent backtracking is used to assign one student to sections of courses. This is done both during the construction of the original timetable, and after publication when a student's enrolment changes. Although no attempt is made to preserve any part of the student's old timetable, this work serves as an example of how some repairing solvers can be useful during both initial construction and minimal perturbation repair.

A recent paper [2] finds a perturbation by building a neighbourhood around a change, deassigning the neighbourhood, and reassigning it optimally using integer programming. Larger and larger neighbourhoods, chosen heuristically, are tried until one succeeds. A precise specification is not attempted; instead, a few common kinds of changes (late surges in enrolments, etc.) are studied.

2 Specifying and solving minimal perturbation problems

A precise specification of a minimal perturbation problem must be built on a precise specification of the corresponding timetable construction problem. For that we will use XHSTT [1], a specification of the high school timetabling problem which has become widely accepted in the last few years.

It would be better to take university course timetabling as the starting point, since it is the problem for which minimal perturbations are most needed; but no precise specification of that problem has achieved the level of acceptance of XHSTT. In any case, the two problems are similar enough, and the present work is general enough, to allow the same approach to be applied to university timetabling, and indeed to other kinds of timetabling problems.

An XHSTT instance defines four main kinds of entities: *times* (intervals of time when events may occur), *resources* (entities that attend events, typically students or student groups, teachers, and rooms), *events* (meetings between resources at specific times; the times and resources may be preassigned, or left to the solver to assign, subject to constraints), and *constraints* (conditions that solutions should obey, including the costs incurred when they do not).

A set of unexpected changes in requirements can be specified as a pair of instances, the first defining the instance before the changes, the second defining it after. Accompanying the first instance is the solution that has been rendered out of date by the changes. (XHSTT includes a format for solutions.)

As promised, this specification is fully general: the changes between the instances may be arbitrary. It is also the form in which the problem presents itself in practice. However, in this form the problem is not solvable by a repair algorithm for the corresponding construction problem. We address this issue now, using an alternative specification derived from this one.

Two entities with the same name, one from the first instance and one from the second, may be assumed to refer to the same real-world entity, since names are not changed wilfully. These names make it possible to compare the two instances to see what has changed.

However, our approach is different—luckily, since the number of kinds of ways in which two instances may differ is dauntingly large. Discard the first instance, and attempt to use the solution to the first instance as a solution to the second instance. A solution is a set of assignments of times and resources to events. Concretely, the entities involved in an assignment are represented by their names. When an assignment in the solution to the first instance names an event, time, or resource which does not appear in the second instance, or the assignment is not legal in the second instance for any other reason, omit that assignment. The result is a legal but possibly incomplete and otherwise defective solution to the second instance called the *amended solution*. Any missing assignments will attract a cost from XHSTT's *assign time* and *assign resource* constraints, which penalize missing assignments.

XHSTT offers *prefer times* and *prefer resources* constraints which express preferences for certain times and resources to be assigned to events. Where the amended solution assigns a time to an event, add a prefer times constraint to the second instance specifying that event and time; where it assigns a resource, add a prefer resources constraint specifying that event and resource. XHSTT allows constraints to be added without interfering with existing constraints; any extra costs due to the new constraints are added to the other costs. The result is an alternative specification of the minimal perturbation problem, in the form of this *amended instance*, incorporating both the unexpected changes and these extra prefer times and prefer resources constraints.

The added constraints need to be given numeric weights, saying what costs are incurred if the preferred assignments are not made. These determine how important preserving the existing solution is taken to be, relative to the other constraints. There is an unavoidable issue here: the institution must have some policy which specifies how to trade off preserving the old solution against satisfying the other constraints. The weights are determined by this policy.

Different constraints can be given different weights. Preserving times can be given higher weight than preserving rooms, for example.

There is nothing informal or incomplete in this construction. It precisely defines the perturbation problem by precisely defining the amended instance.

Assuming that the timetabling problem under study has prefer times and prefer resources constraints (as most do), the amended instance is just another instance of that problem. So a separate specification is not required, and nor are specialized solvers; any solver for the original problem will do.

In practice, however, solvers for the original problem will not be efficient on large instances. For example, it would not be efficient to re-solve a complete university course timetabling problem when one student's enrolment changes.

There seem to be two aspects to achieving efficiency on large instances when finding minimal perturbations. First, the original solution must be used as the starting point, since a complete re-solve from scratch will not be efficient. Thus, a solver capable of repairing an existing (usually incomplete) solution must be used, and passed the amended solution as its starting point.

Second, the solver must focus on the changed parts of the instance. For example, even starting from the amended solution, a simulated annealing solver, used without modification on a university course timetabling instance, would range too widely across the instance when searching for repairs.

Some repairing solvers can run efficiently both during construction and when finding minimal perturbations. Solvers which attempt to repair specific defects in the solution have this property. During construction, they can be targeted at all defects; when finding minimal perturbations, the target can be reduced to just the defects introduced by the changes, if desired. (These defects can be identified in the usual way, using entity names.)

For example, the repair algorithm in [2] repeatedly takes one defect, builds a neighbourhood around it, deassigns that neighbourhood, and reassigns it using integer programming. It is described in [2] as a minimal perturbation algorithm, but it could be used during construction as well. Ejection chain algorithms [5] also start from individual defects and can similarly be used both ways. The one in [5] is ready now to repair XHSTT defects of all kinds.

To conclude, this paper has simplified the task of specifying and solving minimal perturbation problems in timetabling. Instead of enumerating a dauntingly long list of ways an instance may change, it merely tests whether each assignment of the old solution is legal in the new instance. And instead of introducing a new problem requiring a new specification, it re-uses the existing specification and, to some extent, existing solvers.

References

1. Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngäs, Cimmo Nurmi, Gerhard Post, David Ranson, and Henri Ruizenaar, An XML format for benchmarks in high school timetabling, Annals of Operations Research, 194, 385–397 (2012)
2. Antony E. Phillips, Cameron G. Walker, Matthias Ehrgott and David M. Ryan, Integer programming for minimal perturbation problems in university course timetabling, PATAT 2014 (Proceedings of the 10th international conference on the Practice And Theory of Automated Timetabling) 366–379 (2014).
3. R. Barták, T. Müller, and H. Rudová, Minimal perturbation problem – a formal view, Neural Network World 13, 501–511 (2003)
4. Abdallah Elkhyari, Christelle Gúret, and Narendra Jussien, Solving dynamic timetabling problems as dynamic resource constrained project scheduling problems using new constraint programming tools. In Edmund Burke and Patrick De Causmaecker (eds.), Practice and Theory of Automated Timetabling, Selected Revised Papers, Springer-Verlag LNCS 2740, 39–59 (2003)
5. Jeffrey H. Kingston, KHE14: An algorithm for high school timetabling, PATAT 2014 (Proceedings of the 10th international conference on the Practice And Theory of Automated Timetabling) 269–291 (2014).
6. T. Müller and H. Rudová, Minimal perturbation problem in course timetabling, PATAT 2004 (Proceedings of the 5th international conference on the Practice And Theory of Automated Timetabling) 283–303 (2004).