An Automatically Configured Modular Algorithm for Post Enrollment Course Timetabling

Chris Fawcett, Holger H. Hoos
Department of Computer Science
University of British Columbia

Marco Chiarandini
Department of Mathematics and Computer Science
University of Southern Denmark

1 Introduction

Course and examination timetabling is a resource-constrained scheduling problem encountered by universities and other educational institutions, involving scheduling a set of events into given rooms and timeslots. The resulting schedule is subject to various feasibility constraints and preferences derived from the availability of rooms, student enrollment in the events, and precedence relations between events. While the feasibility constraints must be strictly satisfied, creating a satisfaction problem, preferences should not be violated whenever possible, which gives rise to an optimisation problem.

We have developed a new state-of-the-art solver for a particular version of this problem, presented in track two of the Second International Timetabling Competition held in early 2008. Our main contribution lies in the novel automated approach used for designing our new solver, as well as the resulting solver itself. This automatically-configured solver represents a substantial improvement over the previous best algorithm for the problem.

2 Post Enrollment Course Timetabling

As presented in track two of the Second International Timetabling Competition, the Post Enrollment Course Timetabling Problem is defined by:

- 45 timeslots, corresponding to five days with nine one-hour timeslots per day.
- A set of rooms, each with a given seating capacity.
- A set of events, each of which is to be assigned a timeslot and room.
- A set of room features that are satisfied by rooms and required by events, such as a whiteboard or video projector.
- A set of students, each needing to attend a subset of the events.
- A set of *available* timeslots for each event.
- A set of precedences between events, each of the form "Event A must be scheduled in an earlier timeslot than Event B".

There are **five hard constraints** defining the feasibility of event assignments:

- No student can attend two events in the same timeslot.
- Only one event can be scheduled into a given room in a single timeslot.
- The room an event is assigned to must be large enough to accomodate all students, and must satisfy all of the room features required by the event.
- Events must be assigned to one of its *available* timeslots.
- Precedences between events must be respected.

Solvers are permitted to leave some events unscheduled in order to satisfy these hard constraints. Such schedules are defined to be *valid*, but only schedules where all events are assigned can be *feasible*. In addition to the hard constraints, there are **three soft constraints** (or preferences) used to differentiate feasible schedules:

- Students prefer not to have an event scheduled in the last timeslot of a day.
- Students prefer not to attend three or more events in successive timeslots.
- Students prefer not to have only one event on a given day.

We define the *distance to feasibility* of a valid schedule to be the number of students attending events that are unscheduled. To break ties (or for feasible schedules), the *soft constraint violations* is the total number of broken soft constraints in the schedule.

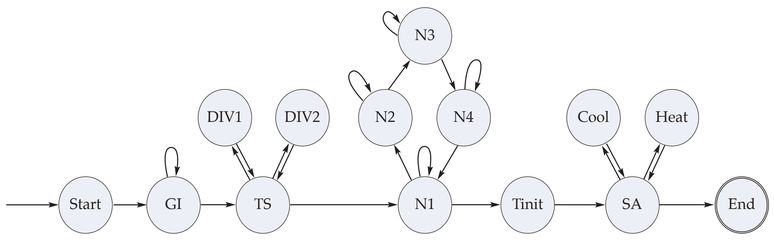
3 Our Automated Design Approach

In recent years there has been a considerable amount of methodological research devoted to the issue of tuning the parameters of optimisation algorithms, especially for heuristic algorithms. Contrary to the traditional approach of trying to minimise the number of user-configurable algorithm parameters, in the presence of new tools for automated configuration it is becoming clear that developers should parameterise as much of their algorithm functionality as possible!

Our approach is to extend this reasoning, and to use these powerful automated algorithm configuration tools to search for a performance-optimised design within a very large space of candidate solvers. Some examples of the design choices that can be left open are:

- Which diversification strategies to use, and at what times
- Which search type to use (i.e. between iterated local search, tabu search, and simulated annealing)
- Should a greedy construction be used to generate an initial solution, or should a random starting point be used?
- Which neighbourhoods and local moves should be used during search?

This approach allows a developer to concentrate on implementing as many modular components as possible, without the tedious overhead of determining how best to connect them together. Out of the available procedures for automated algorithm configuration we selected FocusedILS, as it is the only procedure we are aware of that has been demonstrated to be effective in dealing with very large, discrete design spaces.



A generalised local search machine (GLSM) representation of our configured solver. GLSMs offer a convenient formal framework for representing stochastic local search methods as a combination of independent components linked by conditional or probabilistic transitions. For more information see Chapter 3 of *Stochastic Local Search: Foundations and Applications*, by Holger H. Hoos and Thomas Stützle.

Start - Common initial state
GI - Greedy Initialisation
TS - Tabu search for hard constraints
DIV1 - Diversification by removing some events
DIV2 - Diversification via soft constraint solver
N1 - Two-exchange neighbourhood
N2 - One-exchange neighbourhood
N3 - Swap-time-slots neighbourhood
N4 - Kempe chains neighbourhood
Tinit - Temperature initialisation via sampling
SA - Simulated annealing using N1 and N3
Cool - Geometric cooling
Heat - Additive reheating after idle iterations
End - Finish, after cutoff or optimality

4 Modular Solver Framework

The design space of algorithms for the problem considered in this work is defined by a modular solver framework based on stochastic local search (SLS) methods, and builds on several ideas from the timetabling and graph colouring literature, as well as work done for the first international timetabling competition held in 2003.

Parameter Name	Type	Default	Domain Values
numConstructionSolutions	Hard	10	{1, 5, **10**, 25, 50, 100}
hardTSTabuDelta	Hard	2.0	{1.0, 1.2, **1.5**, 2.0, 2.5, 3.0}
maxNonImprovingIterations	Hard	10000	{1, **10**, 50, 100, 500, 1000, 5000, 10000, 100000, 1000000}
hardTSIdleItersMultiplier	Hard	50	{1, 2, 5, **10**, 25, 50, 75, 100}
resetHowManyAfterImprovement	Hard	disabled	{enabled, **disabled**}
howManyIncrement	Hard	2	{1, 2, 3, 4, 5, **6**, 7, 10, 15, 20}
howManyModulus	Hard	0.2	{0.01, 0.02, 0.05, 0.1, **0.2**, 0.3, 0.4, 0.5}
saAlpha	Soft	0.99	{0.90, **0.91**, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.0}
saIniTempFactor	Soft	0.40	{0.20, 0.40, 0.60, 0.80, 1.00, 1.20, 1.40, 1.60, 1.80, 2.00, 2.20, 2.40, 2.60, 3.00, 3.25, **3.50**, 3.75, 4.00, 4.25, 4.50}
saSamplesForInitialTemperature	Soft	100	{1, 10, 50, **100**, 150, 200}
saIterationCutoffCooling	Soft	50	{1, 10, 50, 100, 1000, 3000, 5000, 7000, 9000, 11000, 13000, 15000, 17000, 19000, 21000, 23000, 25000, 27000, 29000, 31000, 33000, 35000, **37000**, 39000, 41000, 43000, 45000}
saIdleIterationCutoffReheating	Soft	25	{1, 2, 5, 10, 15, 20, 25, 30, 35, 40, 45, **50**, 55, 60, 65, 70, 75}
saStagnationStrategy	Soft	reheat_additive	{**reheat_additive**, reheat_geometric, no_reheating}
saAdditiveBeta	Soft	0.50	{0.1, 0.25, 0.50, 0.75, 1.0, 1.25, **1.50**, 1.75, 2.00, 2.25}
saGeometricBeta	Soft	N/A	{0.01, 0.05, 0.10, 0.20, 0.50, 0.70, 0.90, 1.00}
saSTSSelectionProbability	Soft	0.1	{0.0, **0.1**, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
stsIdleCountForEarlyExit	Soft	1.0	{0.001, 0.005, **0.01**, 0.05, 0.10, 0.25, 0.50, 0.75, 1.0}
twoexIdleCountForEarlyExit	Soft	1.0	{0.001, 0.005, 0.01, 0.05, 0.10, 0.25, 0.50, 0.75, **1.0**}

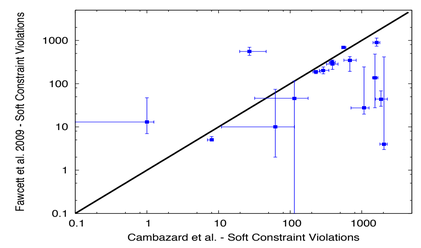
The eighteen exposed parameters in the current version of our solver. "Hard" and "Soft" indicate which phase of the solver the parameter affects. The parameter values of the final solver configuration are indicated in bold.

This framework currently contains 18 configuration parameters and design choices, allowing for on the order of 10^{13} possible unique algorithm instantiations. The backbone of this framework is the idea that one should first solve the feasibility problem, and then tackle the optimisation problem by restricting the subsequent search to moves between feasible timetables. Tabu search is used to tackle the feasibility problem, with various configuration options for initialisation and diversification, while several options are available for optimisation of the soft constraints. The configured solver uses simulated annealing with additive reheating as a diversification strategy.

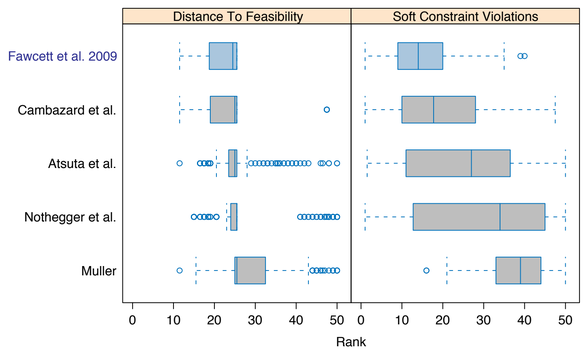
5 Experimental Design and Results

Based on the multi-phase architecture underlying our solver, we applied FocusedILS to first configure the parameters and behaviour of the hard constraint solver and then to configure the parameters and behaviour of the soft constraint solver.

Algorithm configuration and analysis was performed using multiple independent runs of FocusedILS on a cluster of identical machines running SUSE linux 10.1, each with a dual-core Intel Xeon 3.2Ghz CPU with a cache size of 2MB per core. In total, 360 CPU hours were used for configuration, with an algorithm runtime cutoff of 600 CPU seconds. Of the 24 available competition benchmark instances for this problem, the sixteen "public" instances were used for configuration while the eight "private" instances were reserved for testing purposes.



This is a scatter plot comparing the soft constraint solver performance of our configured solver with the competition-winning solver of Cambazard et al. Our solver is superior for all but three instances, in some cases outperforming the solver of Cambazard et al. by an order of magnitude or more.



This figure shows the relative performance of our solver against the solvers of the other four competition finalists. Ten runs were performed on each of the 24 instances for each solver, on our hardware with a runtime cutoff of 600 CPU seconds, and the resulting solutions were placed into rank order.

Using 100 runs on each of these 24 instances, each 600 CPU seconds in length, our configured solver achieves better median soft constraint violation values than any other finalist from the compeition. Additionally, we also beat the top-ranked solver of Cambazard et al. using the rank-based metric used in the competition. As can be seen in the figures, this clearly represents a substantial performance improvement compared to the previous state of the art.

6 Future Research Directions

We are currently working with the classroom services department at the University of British Columbia, in order to generate each semester's exam schedule at the Vancouver campus. This problem is approximately two orders of magnitude more difficult than the benchmark instances currently available, so this is certain to be an exciting project.

Additionally, we are working to make our solver framework even more general, and we will be attempting to automatically configure additional solvers for the problems of the other two tracks of the 2008 timetabling competition. These problems are quite different in flavor than the version presented in this work, and examining the differences in the generated solvers will hopefully yield interesting results.

Get the newest version of FocusedILS:
http://www.cs.ubc.ca/labs/beta/Projects/ParamILS

References:

1. Lewis, R., Paechter, B., and McCollum, B. (2007). Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Working Papers in Accounting and Finance A2007-3, Cardiff Business School, Cardiff University.
2. Hoos, H. H. (2008). Computer-aided design of high-performance algorithms. Technical Report TR-2008-16, University of British Columbia, Department of Computer Science.
3. Hutter, F., Hoos, H. H., and Stützle, T. (2007). Automatic algorithm configuration based on local search. In Proc. of the Twenty-Second Conference on Artificial Intelligence (AAAI '07), pages 1152-1157.