A Modular Multiphase Heuristic Solver for Post Enrolment Course Timetabling

Marco Chiarandini, Chris Fawcett, Holger H. Hoos

A Competition Retrospective.
Chris Fawcett
March 12, 2008
Outline

• The ITC format.
• The problem model for track 2.
• Our development strategy.
• What actually ended up happening.
• The Algorithm.
• Parameter tuning.
• Performance.
The ITC

- Second time the competition has been run.
- First was in 2003, with a different format.
- Three independent tracks, each with different problem models.
- Finalists chosen in each track based on publicly available instances.
The ITC

• Five finalists chosen based on algorithm performance on fourteen public instances.

• The problem? The performance is self-reported, along with the random seed used.

• Verification of that seed is performed by the track organisers.

• Our solution? Thousands of runs, of course!
Track 2

- Meant to describe the university timetabling problem where students pick courses before they are scheduled.
- Timetable constructed based on the course choices for all students.
- Each course (event) must be given a time slot and a room.
Track 2 Specifics

- $n$ events to be scheduled into 45 slots (9 slots x 5 days).
- $r$ rooms, each with a capacity.
- $f$ “room features”
  - Satisfied by rooms.
  - Required by events.
Track 2 Specifics

• $s$ students, each attending a set of events.
• Each event has a set of available timeslots.
• Possible precedence constraints between pairs of events
  • “A must be in an earlier slot than B”.
Hard Constraints

• No student can attend two events at the same time.

• The room chosen for a particular event must be big enough and have the right features.

• One event per room in each time slot.
Hard Constraints

- Each event cannot be assigned to a time slot that is not in its “available” set.
- Events cannot violate the given precedence constraints.
- Can leave events unscheduled to prevent hard constraint violations.
Soft Constraints

• Try not to schedule events in the last time slot of each day.

• Students shouldn’t attend three or more events in successive time slots in one day.

• Students shouldn’t have only a single event on a given day.
Valid vs. Feasible solutions

• A valid timetable is one with no hard constraint violations, but where some events have been left unscheduled.

• A feasible timetable is a valid timetable with no events unscheduled.

• All solutions returned must be valid.
Development Strategy

• Try and integrate the automated parameter tuning process earlier in development.

• Expose as many parameters as possible, let ParamILS sort it out.

• Iterate based on the tuning results.
What Actually Happened?

- ~1 month of development and tuning.
- Some success using this model.
- Pressed for time, so in the end things were quite rushed.
- Not quite enough time for all of the tuning.
- Some parameters dropped in order to have faster tuning runs.
Our Algorithm

• Builds on work by Marco Chiarandini in the 2003 competition.

• Three phases:
  • Construction.
  • Hard constraint satisfaction.
  • Soft constraint satisfaction.
Construction

• Generates valid solutions, with possibly many events left unscheduled.

• Unscheduled events are iteratively placed into with feasible time slot that is available to the fewest unscheduled events.

• A topological order is used to make sure precedence constraints are satisfied.
Hard Constraint Solver

- Tabu search
- At each iteration, an unscheduled event is inserted into the best non-tabu time slot.
  - Selected by looking at the number of students involved.
- All events now causing violations are removed from the timetable.
Soft Constraint Solver

- Simulated annealing over several neighbourhoods.
  - 1-exchange
  - 2-exchange
  - Swap of time slots
  - Kempe chains.
Soft Constraint Solver

- The soft constraint neighbourhoods can introduce hard constraint violations.
- If a quick run of the hard constraint solver can’t repair them, revert.
Parameter Tuning

- During development, many tuning runs used to see how heuristics performed, as well as combinations of heuristics.
- Final tuning used 8 parameters with reasonably discretised domains, for time reasons.
- Each instance run took five minutes, with 16 instances in the training set.
Parameter Tuning

• 80 ParamILS runs performed on Arrow, with each run lasting approximately 24 hours.

• Several parameters ended up being set to the same value in all 80 final configurations.

• Others were set to several close values in their domains.
Public Instances

• Sixteen instances in total, with seven released two weeks before the competition deadline.

• 8 had only 200 events and were generally trivial to solve the hard constraints for.

• 8 had 400 events and were quite a bit harder.
Public Instance Performance

• 541 runs on each of the 16 instances, using the final parameter configuration.

• Best solutions found for each instance were feasible.

• For several 200-event instances, the soft constraint violations were brought to zero or very close to zero.
• This is the empirical SQD for instance 2-10 with the final parameter configuration.

• This is arguably the hardest public instance we had, our submission had quality 1364.
Conclusions

• We were selected as finalists, so hopefully that means the approach was at least decent.

• ParamILS was extremely helpful.

• There is no way we could have manually tuned without taking a lot of time away from development.
Questions?