

TTComp02: Algorithm Description

Tomáš Müller

Charles University
Department of Theoretical Computer Science
Malostranské náměstí 2/25, Praha 1, Czech Republic
muller@kti.mff.cuni.cz

Algorithm Description

The implemented algorithm uses forward based search that extends a partial feasible solution towards a complete solution. The next solution is derived from the previous solution by allocating some event and removing conflicting events from the partial schedule. Every step is guided by a heuristic that aims minimal violation of soft constraints requirements.

The algorithm works in iterations. It uses two basic data structures: a set of events that are not scheduled and a partial feasible timetable. At each iteration step, the algorithm tries to improve the current partial timetable towards a complete timetable. The timetabling starts with an empty partial timetable (which is a feasible timetable), i.e. all the events are in the list of non-scheduled events. Then it goes repeatedly from one partial solution to another feasible solution until all the events are scheduled.

Look now what is going on in the iteration step. First, the algorithm selects one non-scheduled event and computes the locations where the event can be allocated (locations with the hard constraint are not assumed). Every location is described by a start time (the number of the slot for the event) and by a selected room. Then every location is evaluated using a heuristic function over the current partial timetable. Finally, the event is placed to the best location that may cause some conflicts with already scheduled events. Such conflicting events are removed from the timetable and they are put back to the list of non-scheduled activities.

The below algorithm schema (Fig.1.) is parameterised by two functions: event selection and location selection. We can see these functions in a broader view of constraint programming as variable selection and value selection functions. Techniques used for activity and location selection and for escaping from a cycle are described in next sections.

Fine-tuning

Because for the given set of input instances, this algorithm finds a solution very quickly (several seconds). The rest of the available time is used for improving the found solution. This fine-tuning works again in iterations. The above described algorithm is repeatedly executed on the previous solution with several events unscheduled (see Fig. 2.).

```

procedure solve(unscheduled, timetable, time_limit)
  // unscheduled is a list of non-scheduled events
  // timetable is a partial timetable (empty at the beginning)
  iterations=0;
  while unscheduled non empty & time_limit not reached
    & non user interruption do
      iterations ++;
      event = selectEventToSchedule(unscheduled);
      unscheduled -= activity;
      location = selectPlaceToSchedule(timetable, event);
      unscheduled += place(timetable, event, location)
      // place the event and return violated activities
    end while;
  return timetable;
end solve

```

Fig. 1. A kernel of the scheduling algorithm

```

procedure improve(timetable, time_limit)
  backup = timetable;
  while time_limit not reached do
    // backup previous solution
    unscheduled = unscheduleSomeEvents(timetable);
    // remove some events with lots of violated
    // soft constraints from timetable
    timetable = solve(unscheduled, timetable, time_limit);
    if (timetable is feasible &
        timetable has less soft conflict than backup)
      backup = timetable; // backup achieved solution
      tighten location selection heuristics;
    else
      timetable = backup; // restore previous solution
      slack location selection heuristics;
    end while;
  return backup;
end improve

```

Fig. 2. Fine-tuning algorithm

Before the solver starts searching for the new solution from the previous one, some of the events are unscheduled. These events are selected randomly (uniformly, each with probability 0.2) from all events enrolled in soft conflicts.

After the rescheduling of the unscheduled events (which can cause rescheduling of other events), the found timetable is committed only when it has less violated soft constraints than the previous one. The solution is declined otherwise. When the timetable is accepted and thus used for subsequent improvement, the parameters of location selection heuristic are tightened a bit. In reverse, when the timetable is declined and thrown away, the parameters of location selection heuristics are slacked back.

Event Selection (a variable selection criterion)

As mentioned above, the proposed algorithm requires a function that selects a non-scheduled event to be placed in the partial timetable. The first-fail principle is being used here.

In our algorithm we use the following criteria when selecting the event:

- Number of enrolled students (weight -5)
- Number of rooms where an event can be placed (weight 10)
- Number of previous removes of the event from timetable (weight 1)

A weighted-sum of above criteria is used. Event with the minimal heuristic value is selected. Notice that the first criterion is used with a negative weight, this is because a larger value means worse event. Using such formula gives us more flexibility when tuning the system. Because of the randomization of this heuristic, we select one of

$\min \{10, \text{number of unscheduled events} / 3 \}$
events with the lowest weighted sum randomly.

Location Selection (a value selection criterion)

After selecting the activity we need to find a location where to place it. The best-fit advice is used here.

To find the best place for the event we explore the space of all possible locations. We evaluate each such location using the following criteria:

- Number of events in a hard conflict with this location (these events have to be unscheduled when the location is selected) (weight 20)
- Sum of previous removes of the events in a hard conflict with this location (weight 1)

- Number of enrolled students when the location is in the last slot of a day, zero otherwise (initial and minimal weight 2; when fine-tuning succeeds increased by one, decreased by one otherwise)
- Number of students, for who this event will be the only one event in a day if the location is selected (initial and minimal weight 4; when fine-tuning succeeds increased by one, decreased by one otherwise)
- Number of students, who will have more than two events consecutively, if the location is selected (caused by this event) (initial and minimal weight 4; when fine-tuning succeeds increased by one, decreased by one otherwise)

Again, we use a weighted sum of above criteria to evaluate the location. Location with the minimal heuristic value is selected. To remove cycling, this location selection procedure is also randomized. One of top 10 best locations is selected randomly.

Conclusion

Since the above presented algorithm can be executed from any feasible solution, its major fulfilment is in an interactive timetabling. As an example, please visit <http://pipin.ics.muni.cz:8888/ttcomp02/index.jsp>, where this algorithm is used in an interactive way on the input instances of this competition.

The above described algorithm is a variant of an algorithm we presented in papers [1 - 4] listed below.

References

- [1] T. Müller and R. Barták. *Interactive Timetabling*. In Proceedings of the ERCIM Workshop on Constraints, Prague, June 2001
- [2] T. Müller and R. Barták. *Interactive Timetabling: Concepts, Techniques, and Practical Results*. In Proceedings of PATAT'02, Gent, 2002.
- [3] T. Müller. *Interactive Heuristic Search Algorithm*. In Proceedings of the CP'02 Conference - Doctoral Programme, Ithaca, September 2002
- [4] T. Müller. *Some Novel Approaches to Lecture Timetabling*. In Proceedings of the 4th Workshop of Constraint Programming for Decision and Control, CPDC'2002, pp. 31-37, Gliwice, September 2002