An integrated Framework for Distributed Timetabling

Peter Wilke
University of Erlangen-Nuernberg, Germany,
wilke@cs.fau.de

1 The Approach

We have been experimenting with software solutions for time tabling problems. This led to a requirements list for the perfect\textsuperscript{1} framework:

\begin{itemize}
\item support of all available algorithms,
\item optimal parameter settings for the algorithms,
\item fast computation,
\item easy to use GUI.
\end{itemize}

While the last two issues are fairly obvious the first two are worth a closer look.

\begin{itemize}
\item Support of all available algorithms requires that the algorithms have to be encapsulated and share a common interface, otherwise all other components of the framework would have to be implemented more than once.
\item Setting the parameters right is the heart and soul of all optimisation algorithms. The ideal framework would do the job just by itself or with as little user interaction as possible.
\end{itemize}

Our approach is based on the following observations:

\begin{itemize}
\item timetabling algorithms are optimisation algorithms,
\item finding the right parameters is an optimisation problem.
\end{itemize}

Consequently we constructed our framework around a very general approach:

\begin{itemize}
\item the solution of a timetabling problem is found by executing an experiment.
\item An experiment consists of several, independent sequences of computation steps.
\item Each step applies an algorithms to its input and produces an output.
\item Each input is a description of some data, this could be a timetable, a rooster, or a set of parameters of some other step in the sequence.
\item Each output is a description of some data, this could be an optimised timetable, rooster or a set of parameters.
\end{itemize}
The key idea is that the sequences can modify themselves, i.e. the output of a computational step can add new steps and branching points to the sequence (fig. 1), to avoid problems no steps can be deleted once they are created.

As a result complex control structures can be created. Very common are conditional loops (fig. 2). The optimising algorithm (OA) decides whether or not to append new steps to the sequences and puts a copy of itself as new last step in the sequence.

This mechanisms allows to specify scenarios like the following:

1. generate a random timetable,
2. apply a timetable-problem-solving-algorithm using its default parameters and compute a solution,
3. stop, if the solution is good enough, continue, otherwise, and append two new steps: a parameter-optimising-algorithms and a timetable-problem-solving-algorithm to the end of the sequence,
4. apply a parameter-optimising-algorithm and compute an optimised set of parameters,
5. continue with step 3.

2 The Algorithms

Currently the following timetabling-problem-solving algorithms are available: Genetic / Evolutionary algorithms, Branch-and-Bound, Tabu Search, Simulated

---

1 It hasn’t escaped our attention that this is a highly subjective definition. But we do believe that most readers will share our opinion.
Annealing. The following algorithms will be available in the near future: Graph Colouring, Soft Computing, Ant Colony Methods, Hybrid Methods.

The optimising algorithms are quite simple: basically they evaluate a condition and compute new parameter settings.

3 The Implementation

The software is available for Windows and Linux based systems. The software is implemented in Java 5. The core consists of an application server (JBoss), a middle-ware persistence-layer (Hibernate) and a SQL-database. Distributed computing is provided by an RMI based mechanism to distribute the computation steps on a cluster of interconnected (TCP/IP) computers.

4 The Experiments

Tabu search (TS), simulated annealing (SA) and the genetic algorithm (GA) have been applied to three real world problems. All runs were timed on a 1.7 GHz P4 PC with 512 MB RAM. Average rounded times are given.

A monthly rooster for a hospital ward with 21 nurses and 4 shifts per day was planned.

A timetable for a school with 113 teacher, 100 rooms and 43 classes was generated.
Our university organises a girl-and-technology week each year to attract more female students to technical subjects. In 2002 303 girls participated in 199 half-day projects. Each girl listed up to 4 first-preference projects she would like to attend, 4 substitute projects, and a list of best-friends she would like to attend the same projects.

For this problem the results are as follows:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>time (s)</th>
<th>fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS</td>
<td>300</td>
<td>450</td>
</tr>
<tr>
<td>GA</td>
<td>3500</td>
<td>750</td>
</tr>
<tr>
<td>SA</td>
<td>3500</td>
<td>430</td>
</tr>
</tbody>
</table>

5 Summary

We described a new framework to solve timetabling problems. The key features are:

- a set of ready-to-use off-the-shelf algorithms,
- experiments for automated generation of solutions,
- optimising algorithms can influence the sequence of computation steps,
- algorithm can be applied to problems or to other algorithms.

The framework has been successfully tested with several real world problems.
6 Acknowledgement

I would like to thank Matthias Gröbner, Norbert Oster, Stefan Büttcher, Adel Habassi, Hichem Essafi, Gerlinde Gagesch, Sven-Dimo Korsch, Georg Götz, Andreas Konrad and Ronny Heft for their contributions to our software.

References