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*[1] framework:

- support of all available algorithms,
- optimal parameter settings for the algorithms,
- fast computation,
- easy to use GUI.

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

- 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.
- Setting the parameters right is the heart and soul of all optimisation algorithms. The ideal frame work would do the job just by itself or with as little user interaction as possible.

Our approach is based on the following observations:

- timetabling algorithms are optimisation algorithms,
- finding the right parameters is an optimisation problem.

Consequently we constructed our framework around a very general approach:

- the solution of a timetabling problem is found by executing an *experiment*.
- An experiment consists of several, independent *sequences* of computation steps.
- Each *step* applies an algorithms to its input and produces an output.
- 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.
- Each output is a description of some data, this could be an optimised timetable, rooster or a set of parameters.

E. K. Burke, H. Rudová (Eds.): PATAT 2006, pp. 542–546. ISBN 80-210-3726-1.

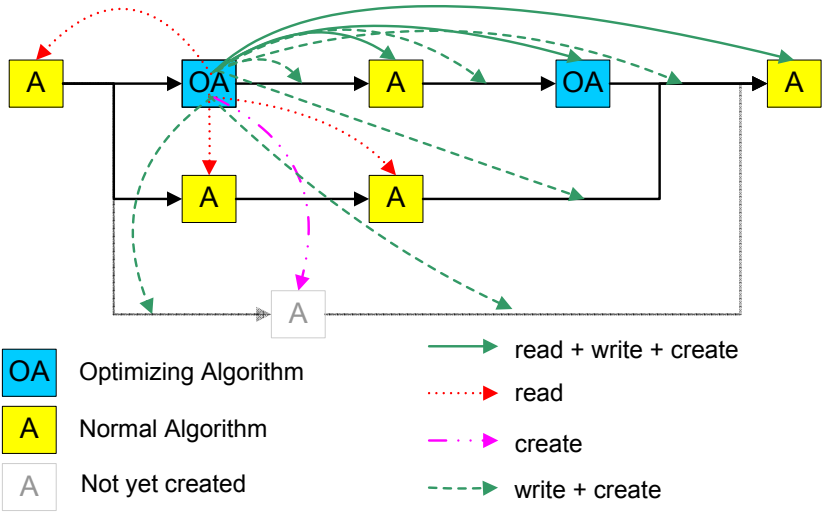

Fig. 1. Different ways an optimising algorithm can interact with the steps of his sequence.

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.

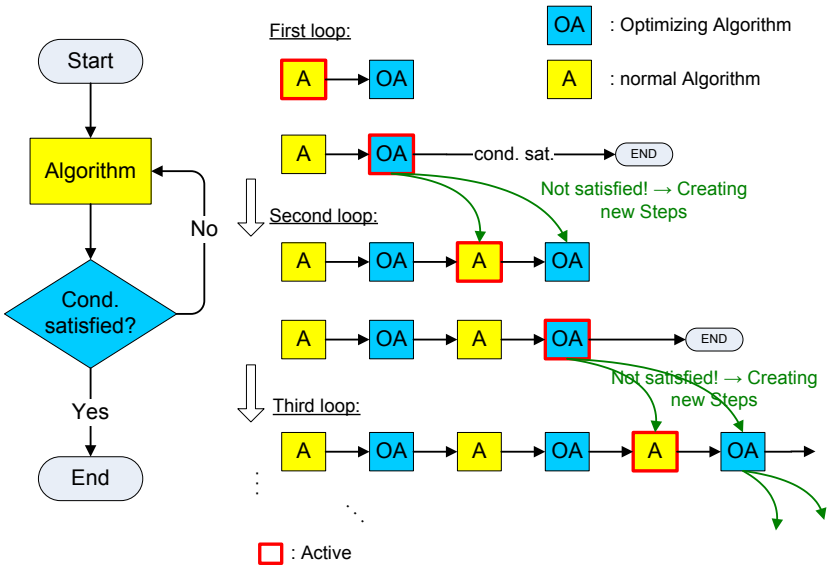

Fig. 2. Expansion of a loop in a sequence of steps.

Annealing. The following algorithms will be available in the near future: Graph Colouring, Soft Computing, Ant Colony Methods, Hybrid Methods.

The optimising algortihms 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.

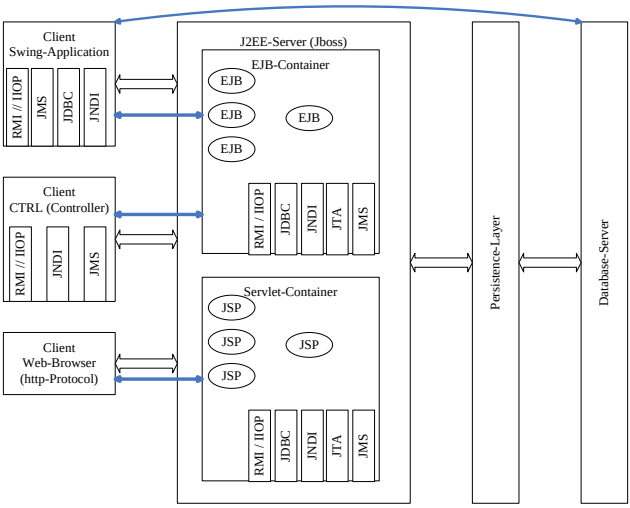

Fig. 3. The basic components of the software architecture

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:

Algorithm	time [s]	fitness
TS	300	450
GA	3500	750
SA	3500	430

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

1. Eric Taillard Alain Hertz and Dominique de Werra. Tabu search. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley and Sons, 1997.
2. Edmund Burke and Patrick De Causmaecker, editors. *Springer Lecture Notes in Computer Science*, volume 2740. Springer-Verlag, Juli 2003.
3. Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
4. Matthias Gröbner, Peter Wilke, and Stefan Büttcher. A General View on Timetabling Problems. In Burke and Causmaecker [2].
5. Bruno R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*, chapter Simulated Annealing, page 474. http://www.brpreiss.com/books/opus5/html/page474.html, 1998.
6. Peter J.M. van Laarhoven. *Simulated annealing*. D. Reidel Publishing Company, 1987.
7. William T. Vetterling William H. Press, Saul A. Teukolsky and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*, chapter 10.9, pages 444–447. Cambridge University Press, 1992.