Generalized Scheduling Development Environment

William Fabens
BP Research
Cleveland, OH 44128

Leon Sterling
CAISR
Case Western Reserve Univ.
Cleveland, OH 44106

Abstract

A generalized scheduling development environment allows a person who composes schedules to specify both an application and a methodology, in developing a scheduling program. We have designed such a system and have built a proof-of-concept prototype. Our prototype has been tested on two fairly different application examples. It demonstrates the expressive capability of logic programming in representing the diversity of the two examples, and it exemplifies a method of achieving generality in the domain of scheduling.

1 Introduction

We are building a generalized environment for building ‘special-purpose’ schedulers. The aim is to significantly decrease the time and resources it takes to build an optimizing scheduler.

This work can be compared to ‘constraint directed’ scheduling [Fox, 1987], the wealth of operations research methods which use linear optimization methods. In [Dhar, 1990], an operation systems approach is compared with an expert system approach, as in, e.g., [Gudes, 1990].

2 Design

Our primary design goals in building the generic scheduler are

- that it be capable of backtracking,
- that backtracking be controllable both by specifying the order of evaluation and the distance to backtrack (e.g. skipping levels),
- that more-efficient problem solvers would be used when a problem is reduced to a known deterministic subproblem.

3 The Prototype System

The prototype generic scheduler has been applied to two applications, both based on actual applications: repair garage scheduling and tanker scheduling.

The Prolog program is composed of a ‘generic’ set of predicates describing the generate-and-test framework (about 10 predicates long), then the problem-specific portion consists of about a half-dozen predicates. Once the problem is specified, the interactive user manipulates the facilities of the generic part.

4 The User Interface

The architecture of the interface is given in Figure 1. We intend for two classes of user to interact with the system: the scheduling ‘informant’ and the person who would naturally run the model. Interaction with the non-informant is straightforward. Scenarios are created, the program evaluates them, and a schedule is created.

An example of informant interaction, for instance, would be that the program would be told to order the jobs in descending order of job difficulty, or to reorder the categories of resources, or to back up in special ways.
5 Discussion

To date, our experiences in building the model have been that the generate and test part of the prototype is robust and quite easy to program (in Prolog). Developing the user interface has taken longer.

Acknowledgements

The research was supported by a grant from BP America and the Cleveland Advanced Manufacturing Program. Venkatesh Srinivasan and Madhavi Ahuja have worked on the user interface. Thanks to Marc Kirschenbaum for some helpful discussions. Special thanks to Arun Jain who suggested this project in the first place.

References

