Design-to-time Scheduling and Anytime Algorithms *

Alan Garvey and Victor Lesser
Computer Science Department
Lederle Graduate Research Center A243
University of Massachusetts
Amherst, MA 01003
Email: {garvey,lesser}@cs.umass.edu

Abstract

Design-to-time real-time scheduling is an approach to solving time-sensitive problems where multiple methods are available for many subproblems. It is an alternative to the anytime algorithm approach, scheduling discrete methods rather than anytime algorithms with the goal of maximizing the value of the scheduled computation. In this paper we briefly introduce the design-to-time approach, describe how design-to-time can be used to schedule anytime algorithms including some experimental results, and examine anytime characteristics of our design-to-time scheduling algorithm.

Introduction

Design-to-time[Garvey and Lesser, 1993; Garvey *et al.*, 1993; Garvey *et al.*, 1994] is an approach to problem solving that involves designing a solution plan dynamically at runtime that uses all of the time available to find as good a solution as it can. Because the problems it is solving are generally intractable and because time spent finding solution plans is time that could otherwise be spent solving the actual problem, it is a satisficing approach. In our design-to-time work, problem solving is modeled as a set of interrelated computational tasks, with alternative ways of accomplishing the overall task and not a single "right" answer, but a range answers of different qualities, where the overall quality of a problem solution is a function of the quality of individual subtasks. Because of the choices available at all levels of task scheduling, design-to-time combines a simple form of planning (deciding *what* to do) with scheduling (deciding *when* to do it). A major focus of our work on design-to-time is on taking interactions among subproblems into account when building solution plans, both "hard" interactions that must be heeded to find correct solutions (e.g., hard precedence constraints), and "soft" interactions that can improve (or hinder) performance (e.g., facilitates constraints[Decker and Lesser, 1993]).

An example of a problem to be solved by the design-to-time scheduling algorithm is given in Figure 1. This representation of a task structure is based on the TÆMS modeling framework[Decker and Lesser, 1993]. In a TÆMS task structure the leaves of the graph represent executable computations (known as *methods*) and the nonleaf nodes represent tasks that achieve quality as a function of the quality of their subtasks. Each separate graph is known as a *task group* and represents a single independent problem to be solved. Each task group has a deadline by which all computation on that task group must be completed. Non-parent-child connections between tasks and methods represent interactions, such as *enables* (Task A must have quality greater than a threshold before Method B can correctly begin execution) and *facilitates* (if Task A has quality greater than a threshold, then Method B will have reduced duration and/or increased quality).

Given task structures of this form, the job of the design-to-time scheduling algorithm is to dynamically build schedules with a preference for schedules that (in order of importance) achieve nonzero quality for all task groups, maximize the sum of the qualities of all task groups, and minimize the total duration of method executions. The result of this scheduling algorithm is a schedule that specifies what methods to execute, when to execute them, and what values are expected from that execution. Figure 2 shows a schedule for the task structure in Figure 1.

A detailed description of our most recent design-to-time scheduling algorithm can be found in a recent paper[Garvey and Lesser, 1995]. That paper also discusses the incorporation of uncertainty in method duration/quality and relationship parameters into the model. Uncertainty leads to the need for both scheduling with uncertainty in mind (for example, by building contingencies into schedules and taking the likelihood of failure into account when evaluating schedules) and monitoring the execution of methods to allow rescheduling when progress is not acceptable. In the remainder of this paper we first describe our extensions to design-to-time to allow the scheduling of anytime algo-

*This paper has not already been accepted by and is not currently under review for a journal or another conference. Nor will it be submitted for such during IJCAI's review period. This material is based upon work supported by the National Science Foundation under Grant No. IRI-9208920, NSF contract CDA 8922572, DARPA under ONR contract N00014-92-J-1698 and ONR contract N00014-92-J-1450. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

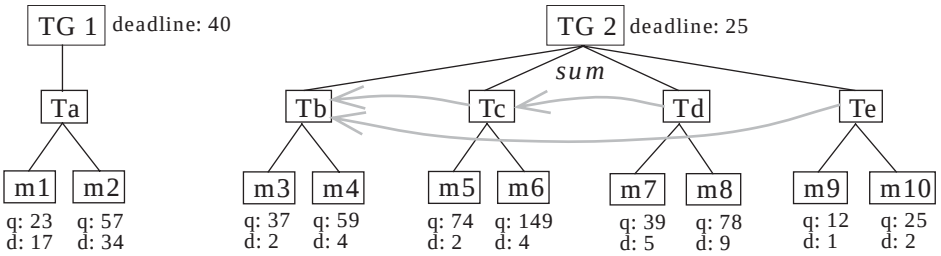

Figure 1: An example of TÆMS task structure to be scheduled. The black lines represent task/subtask connections, while the gray lines represent facilitates relationships.

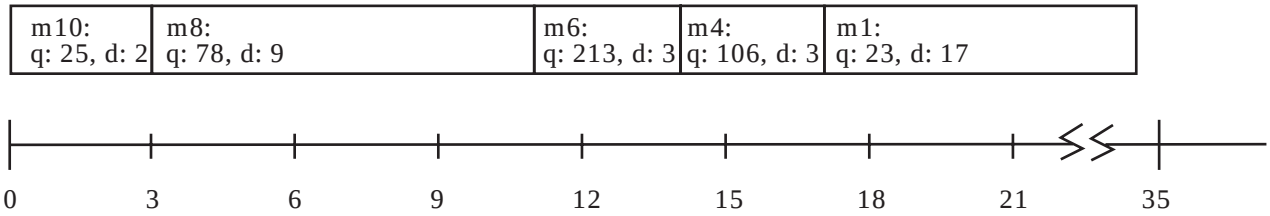

Figure 2: A schedule that solves the problem from Figure 1. The runtimes and qualities for $m6$ and $m4$ are not as indicated in Figure 1, because both methods are facilitated, thus reducing duration and increasing quality.

rithms, then discuss the anytime performance characteristics of our design-to-time scheduling algorithm.

Design-to-time Scheduling of Anytime Algorithms

In the example of design-to-time given above, each method had an expected duration and an expected quality. Each of these methods achieves zero quality unless it is executed to completion, at which point it achieves full quality. This can be thought of as a performance profile with a single large step at completion.

We have recently extended our design-to-time scheduling algorithm to schedule methods that are anytime algorithms. This extension involves extending our method execution model to allow methods that accumulate quality as they execute (before completion), representing the performance profiles of these anytime algorithms, modeling the effect of hard and soft relationships on these performance profiles, and effectively scheduling the execution of these anytime algorithms.

Modeling Anytime Algorithms

Our base representation of performance profiles is similar to one described by Zilberstein [Zilberstein, 1993]. This representation uses a table where each row in the table represents a particular duration, each column represents a quality value, and entries in the table store the probability that the given quality will be achieved in the given duration. This provides a discrete distribution of quality values for each possible duration.

We extended this representation to allow the modeling of the effects of hard and soft relationships. Hard and soft relationships can adjust the maximum achievable quality (and the corresponding duration to achieve that quality) for a method, that is, they adjust the point in the performance profile where maximum quality is achieved. We chose to model this by normalizing our performance profile quality and duration values to this point (i.e., maximum quality has value 1 as does the duration to achieve this quality). Our table consists of these normalized values, so, when the actual maximum quality changes because of the effect of a relationship, the entries in the table can remain unchanged. Zilberstein achieves a similar effect by having different performance profile tables for different input quality values. His approach is more general (because he doesn't assume that the profiles keep the same general shape as we do), but is less space efficient. However, because his approach is more general it may better model particular anytime algorithms.

Given this normalized table representation, we can calculate the expected quality for any particular duration by finding the normalized duration that is closest to the given duration (possibly through interpolation), looking up the associated normalized qualities and multiplying them by the current maximum achievable quality.

Scheduling Anytime Algorithms

Our basic approach to scheduling anytime algorithms with our design-to-time scheduler is to choose a few duration/quality points on each performance profile and treat those as discrete methods. For example, Figure 3 shows a performance profile with a set of discrete duration/quality points. Once this set of points is chosen, each of them can be modeled as a possible method to execute, and design-to-time scheduling can be done as it is normally. (It might be desirable to have special scheduler features for anytime al-

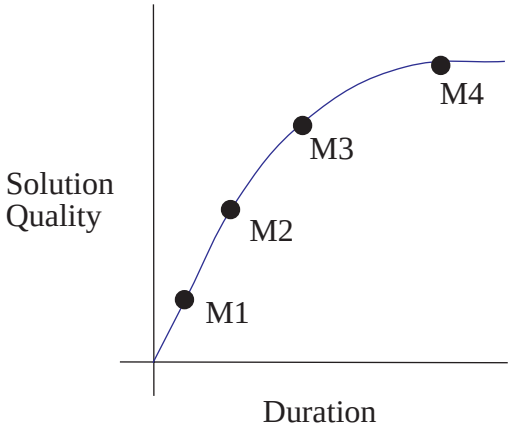

Figure 3: An anytime algorithm performance profile with a set of discrete duration/quality points.

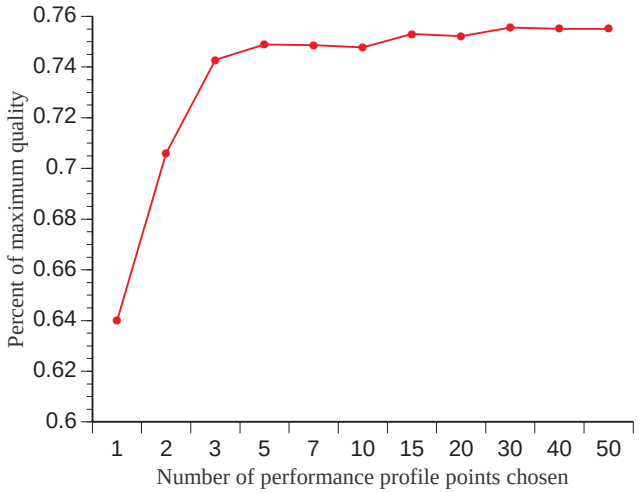

Figure 4: The percent of maximum quality of the best schedule as a function of the maximum number of points chosen from each anytime algorithm performance profile.

gorithm methods, such as the ability to dynamically choose intermediate points on a performance profile to maximize performance or to delay the execution of anytime methods because their highly predictable duration allows them to safely execute right up to a deadline, but we do not believe such extra features would significantly improve system performance, because they would be outweighed by the uncertainties of method performance.)

Interesting questions include how to choose the discrete points on the performance profile and how many points should be chosen. We have implemented an algorithm for choosing the discrete points that tends to choose points that allow close modeling of the performance profile function. More points tend to be chosen in areas of rapid change in duration or quality. The algorithm starts with a large set of points (initially a point for every integer duration value) and each cycle removes the point with the smallest duration/quality area between it and the next point. Points that are close to each other in duration and quality will have small areas; at points of transition in quality or duration this area will be large. The algorithm completes when the desired number of points are all that remain.

We chose a more empirical approach to investigating the number of points to be chosen. Figure 4 is a graph showing the percent of maximum quality achieved for a range of values for the number of points chosen. In these experiments we ran the system with each possible number of points on the same 100 randomly generated problem instances. When the number of possible points is 1 our options are to either execute the anytime algorithm to completion or not execute it at all. As the number of points increases the number of duration/quality choices increases. In these experiments, performance improved up to about 5 choices, then leveled off. This is probably related to the fact that the average duration of a method in these experiments was 10.

Anytime Design-to-time Scheduling

Our design-to-time scheduling algorithm is heuristic and among the decisions that it makes heuristically are which combinations of methods (known as alternatives) to try to schedule and how many of these combinations to try. These heuristics tend to cause the algorithm to produce better schedules as it is given additional time to schedule (that is, as a greater number of alternatives are considered.)

Figure 5 shows the sum quality of the best schedule found (so far) as more alternatives are considered for the example problem from Figure 1. This algorithm has a few parame-

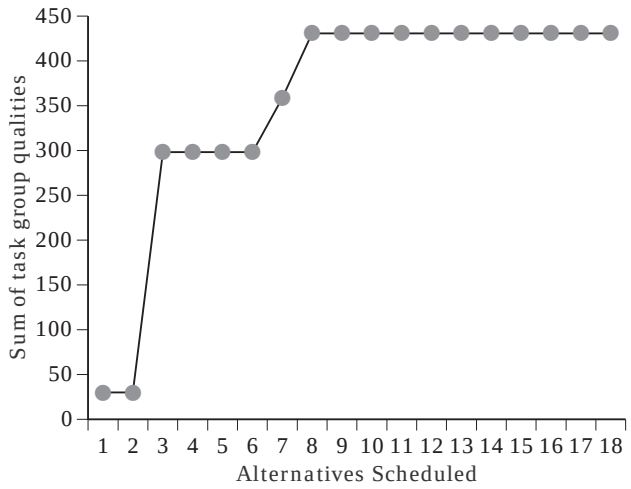

Figure 5: The sum quality of the best schedule as a function of the number of alternatives considered for the task structure from Figure 1.

ters that can externally control the behavior of the algorithm. Judicious use of these parameters can result in anytime per-

formance by the algorithm. Unfortunately it would probably be quite difficult to calculate an *a priori* performance profile for the scheduling of a particular task structure. It should (in principle) be possible to predict the bounds on runtime for the algorithm given a particular task structure, but it is very difficult (probably impossible) to predict the expected quality and duration of the resulting schedule without actually doing a minimum amount of actual scheduling.

Conclusions

We have described the design-to-time approach to real-time problem solving. We have shown how that approach can be extended to effectively schedule anytime algorithms. We have also shown how the performance of the scheduling algorithm itself is anytime in character, producing better schedules as more alternatives are considered.

We would like to extend this work by doing more detailed experiments comparing our scheduling approach to other anytime algorithm scheduling approaches, and by understanding in more detail the anytime character of the design-to-time scheduling algorithm, possibly leading to an algorithm that is better able to control its own performance.

Acknowledgments

We would like to thank Satoru Fujita for his help on the design-to-time scheduling algorithm implementation and for the idea and initial implementation of the code for choosing discrete points on performance profiles. We would like to thank Keith Decker for his work on the TÆMS task modeling framework.

References

Decker, Keith S. and Lesser, Victor R. 1993. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington. 217–224.

Garvey, Alan and Lesser, Victor 1993. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics* 23(6):1491–1502.

Garvey, Alan and Lesser, Victor 1995. Design-to-time scheduling with uncertainty. CS Technical Report 95–03, University of Massachusetts.

Garvey, Alan; Humphrey, Marty; and Lesser, Victor 1993. Task interdependencies in design-to-time real-time scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C. 580–585.

Garvey, Alan; Decker, Keith; and Lesser, Victor 1994. A negotiation-based interface between a real-time scheduler and a decision-maker. CS Technical Report 94–08, University of Massachusetts.

Zilberstein, Shlomo 1993. Operational rationality through compilation of anytime algorithms. Ph.D. Dissertation, Department of Computer Science, University of California at Berkeley, Berkeley, CA.