Issues in Design-to-time Real-time Scheduling [*]

Alan Garvey
Department of Computer Science
Pacific Lutheran University
Tacoma, WA 98447
Email: garveyaj@plu.edu

Victor Lesser
Computer Science Department
University of Massachusetts
Amherst, MA 01003
Email: lesser@cs.umass.edu

Abstract

Design-to-time real-time scheduling is an alternative to the many flexible computation approaches that are based on anytime algorithms. It builds schedules at runtime that dynamically combine solutions to subproblems, taking advantage of the time available to achieve the best results it can. In this paper we look in detail at a few issues related to design-to-time, including where the approximations we rely on come from, how uncertainty affects the scheduling process and the interface between the scheduler and its invoker.

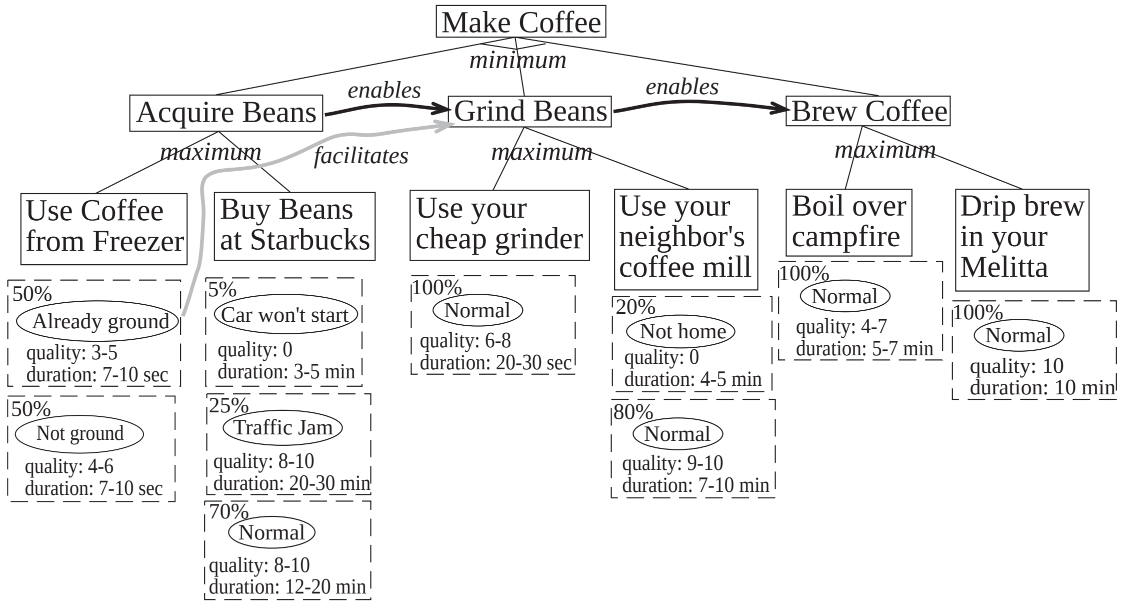
Introduction

An alternative to the standard flexible computation approach of having individual algorithms that produce better results as they are given additional runtime, is the *design-to-time real-time scheduling* [Garvey, 1996; Garvey *et al.*, 1993; Garvey and Lesser, 1993] approach that builds schedules at runtime that dynamically combine solution methods for different parts of the overall problem, attempting to maximize the quality of the solution generated. Key to this approach is the availability of multiple methods that trade off solution quality for time, as well as a clear understanding of how these methods can be combined to solve the overall problem, how these methods interact with one another, and the quality, duration and possibly cost distributions associated with the methods. In fact, this approach can also schedule the execution of anytime algorithms by choosing a set of points on the anytime performance profile and treating those as individual solution methods.

In design-to-time we frame the input problem in a generic representation of task structures known as TÆMS [Decker and Lesser, 1993]. In TÆMS we describe problems in terms of how quality is incrementally accumulated over time, what soft and hard interactions there are between tasks, and how much uncertainty there is in method quality and duration. Figure 1 shows an example of a simple TÆMS task structure that describes the problem of making coffee. Note that there are multiple methods for solving each of the three subtasks of the main problem. Several of these methods have more than one possible outcome, describing different kinds of results that can have significantly different effects on the overall solution. This example is meant to introduce the reader to the ideas of TÆMS, not to suggest that making coffee is a representative example of the complexity of problems we are interested in solving.

The TÆMS framework gives us what might be called a partially digested description of a problem. We do not have to plan completely from scratch, deciding what operators apply in a given situation and how they can be combined to solve problems. That information is available to us in the TÆMS task structure. Our job is to determine which of the many options available in that task structure we should actually pursue. Because options are dynamically combined (rather than using predefined process plans) TÆMS needs to explicitly represent the soft interactions between tasks that could normally be built in to the process plan for a task, because the exact effect of the interactions depends on the context in which the tasks are executed. While it may not be trivial to represent a problem in TÆMS we believe that the TÆMS representation is much more

[*]The work described in this paper was done while the first author was affiliated with the University of Massachusetts. This research project is sponsored by the Dept. of the Navy, Office of the Chief of Naval Research, under Grant No. N00014-95-1-1198 and by the National Science Foundation under Grants No. IRI-9523419 and IRI-9321324. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.



Make Coffee
minimum
Acquire Beans
enables
Grind Beans
enables
Brew Coffee
maximum
facilitates
maximum
maximum
Use Coffee from Freezer
Buy Beans at Starbucks
Use your cheap grinder
Use your neighbor's coffee mill
Boil over campfire
Drip brew in your Melitta
50% Already ground quality: 3-5 duration: 7-10 sec
50% Not ground quality: 4-6 duration: 7-10 sec
5% Car won't start quality: 0 duration: 3-5 min
25% Traffic Jam quality: 8-10 duration: 20-30 min
70% Normal quality: 8-10 duration: 12-20 min
100% Normal quality: 6-8 duration: 20-30 sec
20% Not home quality: 0 duration: 4-5 min
80% Normal quality: 9-10 duration: 7-10 min
100% Normal quality: 4-7 duration: 5-7 min
100% Normal quality: 10 duration: 10 min

Figure 1: An example of TÆMS task structure to be scheduled. The black lines represent task/subtask connections, the thin gray line represents a facilitates relationship and the thick gray lines represent enables relationships. The dashed line boxes represent different possible outcomes for each method. Each of these possible outcomes has a range of possible quality and duration values.

reasonable for a real-time problem-solver. Planning from first principles is often difficult in real-time scenarios. TÆMS is an abstract description of problem solving activities that allows us to make decisions about how to tradeoff among solution quality, cost and duration of alternative ways of accomplishing a task. TÆMS task structures can be generated offline for problems that are expected to come up during problem solving.

Our heuristic design-to-time scheduling algorithm consists of three main components: high-level search through the set of alternatives (unordered combinations of executable methods), a scheduler that assigns start and finish times to the methods from an alternative, and an evaluator that both remembers the best schedules found so far and looks for ways to improve the current schedule.

The algorithm consists of these three particular components because they help to reduce the overall complexity of the problem to manageable levels. Initial search is done at a higher level of abstraction, ignoring interactions between tasks and deadlines, to allow us to feasibly consider a broad cross-section of possible solution plans. Scheduling is done on only those plans that make it through the first level of the algorithm, because completely scheduling all possible plans would be computationally impossible in most situations. Evaluation is done at the end to reduce the likelihood that we are missing good plans by suggesting plans that are simple variations on existing plans that could have increased value.

In this paper we discuss issues related to design-to-time real-time scheduling. First we go into some detail about how the approximations we use can arise in real problems. Then we focus on uncertainty in the information used for scheduling and how that is handled by the algorithm. Finally we briefly describe the interface to the scheduler and how it should allow the scheduler to be usefully used to solve real problems.

Sources of approximations

One question that a study of design-to-time needs to address is the question of where approximations come from in an application and how difficult they are to construct. In our experience with building complex problem solving systems, multiple paths of control seem to occur naturally. Approximations, often in the form of variations on a standard algorithm, are a standard part of building AI applications. We can identify at least three general sources of approximation in problems. They are the use of approximate data, the use of approximate algorithms, and (related to the second) the dynamic creation of approximate algorithms through the rearrangement and skipping of individual problem steps.

Examples of these kinds of approximations can be seen in a pair of large, complex AI applications with which we are familiar: the Distributed Vehicle Monitoring Testbed (DVMT) [Decker *et al.*, 1990], an application that identifies and tracks vehicles moving through a region, and the Integrated Processing and Understanding of Signals system (IPUS) [Lesser *et al.*, 1995].

An example of the use of approximate knowledge from the DVMT is an approximation that can replace the multiple steps involved in interpreting low level sensor data and using it to identify and track vehicles moving through the domain. Figure 2 shows an example of an approximation from the DVMT that is used in vehicle identification. In this case the two ways to solve the problem are one that uses the grammar on the left and has individual methods for propagating information from the lowest level to the intermediate level and for propagating information from the intermediate level to the top level, and one that uses the grammar on the right and has one individual method that goes from the lower level to the upper level. Using the approximation nearly halves the runtime required for a solution, at the expense of reducing the certainty and precision of the identification of the vehicle.

In IPUS the task is to identify the sources of various acoustic signals that can be detected. Fourier transforms are used to isolate signals into particular spectral bands. Within the IPUS project, recent work has looked at approximating these transform calculations [Nawab and Dorken, 1993]. Either these approximations or standard Fast Fourier Transform algorithms can be used, depending on the time requirements and the importance of the particular calculations.

Data approximations can appear in the DVMT in the form of time skipping and clustering. In time skipping, instead of using data from every sensor interpretation cycle, some cycles are skipped. The same algorithms are used to process the data, but when it is processed less frequently runtime is saved that can be applied to other tasks. Clustering in the DVMT involves treating groups of data points as if they were a single point for processing. Again this has the effect of reducing solution quality while saving significant runtime.

Related to algorithm approximation is the use of algorithms where the form of solutions to at least some intermediate problems is shared, allowing some intermediate problem solving steps to be skipped or rearranged to save time. Representations of this type are common in iterative refinement approaches (such as anytime algorithms) where a solution must always be available. However, not all approximations of this form are straightforward anytime algorithms. It may be that different small groups of intermediate problem steps share different solution forms or that none of these intermediate solutions can stand alone as complete problem solutions. Approximations of this form appear in several places in the IPUS control structure. This occurs because IPUS uses iterative techniques to solve many of its subtasks. One example is the basic IPUS control loop, which uses a bottom-up approach to process low level signal data, then verifies what the bottom-up processing concludes by doing top-down processing. But this entire verification step can be bypassed at the potential cost of low quality solutions (e.g., increased uncertainty about the correctness of the signal identification.) Within the top-down verification process IPUS looks for data to verify its conclusions, diagnoses why expected signals were not found and reprocesses the data to verify that its diagnoses correctly identified what was happening. Again both the diagnosis step and the reprocessing step can be bypassed to save time.

As these examples suggest, approximations can often arise naturally in complex problem solving systems. We believe that as applications become more complex, the desirability of having multiple paths of control through them will become apparent.

As mentioned briefly above, the design-to-time scheduling algorithm can also be used to schedule anytime algorithms. An interesting example of how approximations can appear even with anytime algorithms comes from the work of Crites [Crites and Barto, 1996]. He uses simulated annealing to learn good control plans for elevator operation. His use of simulated annealing has a definite anytime character, as more time is spent learning, better plans are generated. However, his results show that if you know in advance how much time is available for annealing, it is possible to achieve better results than those achieved by just stopping the execution of a longer annealing process, because of the parameter settings associated with the annealing. This suggests that it is useful to have a set of different anytime algorithms for the annealing (with different parameter settings associated with different expected completion times) and choosing from among those algorithms at runtime in a design-to-time manner.

Uncertainty

Our design-to-time algorithm relies on being able to accurately predict runtime information about the tasks being scheduled. However, because the kind of

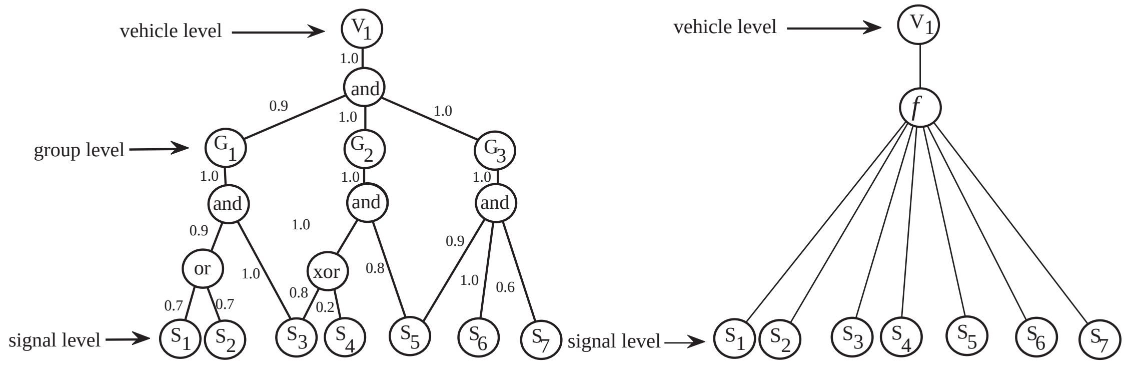


Figure 2: Both complete (on the left) and approximating (on the right) grammars describing the hypotheses necessary to identify a vehicle of type V_1 in a sensor interpretation application.

AI tasks that we are interested in scheduling often involve search, it is usually not possible to completely predict the runtime and solution quality of the tasks. In this section we describe how we model this uncertainty in TÆMS and how the scheduler takes the uncertainty into account when producing schedules.

Uncertainty in TÆMS takes the form of uncertainty concerning what kind of outcome a task will produce, what the solution quality of the outcome will be, how long it will take to achieve the outcome, whether a soft interaction exists between tasks, and what the power of existing soft interactions are.

At the lowest level there is a distribution of possible kinds of outcomes for a task. Each outcome has a solution quality distribution and a duration distribution. Different outcomes can represent different kinds of solutions to a problem, e.g., success versus failure or different ranges of value. Note that the outcome of a task is not directly under the control of the problem solving system. Outcomes just allow us to represent the different kinds of answers we could get after executing a task. Different outcomes can have different interactions with other tasks. For example, a positive outcome might facilitate another task while a negative outcome does not. Associated with each interaction is a likelihood that the interaction actually exists and distributions for the parameters of the effect. For example, associated with a facilitation interaction is a likelihood that facilitation will actually take place (say 50 percent) and distributions for the power parameters that describe the effect on solution quality and duration of the facilitated task.

Given these kinds of uncertainties, the scheduler needs to produce schedules that are likely to achieve the kind of results desired. It does this by scheduling using likely bounds of expected task performance (rather than just expected values), possibly scheduling redundant tasks to increase certainty, monitoring the performance of executing tasks and rescheduling when necessary.

Rather than just schedule using the expected values from solution quality and duration distributions, the scheduler actually combines distributions, and uses upper and lower bounds to predict the results of computations. This allows best and worst case expectations to guide the decisions about which schedules to choose to execute. Our use of bounds in scheduling has been influenced by the work of Fujita [Fujita and Lesser, 1996a; Fujita and Lesser, 1996b].

One aspect of this is that it is sometimes beneficial to schedule redundant tasks to increase the likelihood that a good result is generated. For example, we may choose to use each of two methods for computing an intermediate result, to allow us to choose the best actual outcome at runtime. This is particularly useful for the kind of AI search tasks that we are most interested in scheduling, because they are most likely to have significant amounts of uncertainty in their predicted results.

Also important to ensuring that schedules perform as expected is the monitoring of the execution of tasks. This involves setting appropriate expectations for task execution, actually checking on the progress of tasks as they execute to see if they are meeting expectations, and possibly rescheduling if expected performance is being either not met or exceeded.

Scheduler interface

One important aspect of a scheduling system such as ours is that it does not work in isolation. It needs to interact with the other components in a problem solving system. We have spent a significant amount of time working out the details of an input/output specification between the scheduler and a "decision maker" that uses the schedule output to decide what to do and what information to communicate to other problem solving agents [Garvey *et al.*, 1994].

At the heart of the scheduler interface is the idea of *commitments*, which constrain the scheduler to try to produce schedules with particular properties. When the scheduler can commit to satisfying some particular constraint, that commitment can be communicated to other agents, who can rely on it in their computations.

The scheduler supports three kinds of commitments:

- *Do commitment* – that commits the scheduler to completing a particular task as part of problem solving,
- *Deadline commitment* – that commits the scheduler to completing a particular task by a deadline (presumably a deadline that is earlier than the hard deadline associated with the task),
- *Earliest-start-time commitment* – that commits the scheduler to not starting a particular task before an earliest start time (again, presumably an earliest start time that is later than the hard earliest start time associated with the task)

The scheduler is invoked with a set of commitments that the decision maker would like satisfied. Deadline and earliest start time commitments can have either specific times associated with them or, for deadlines an indication that the *earliest* possible finish time is desired and for earliest start times an indication that the *latest* possible start time is desired. This allows the scheduler to build schedules with realistic values for these parameters rather than forcing the invoker to guess what reasonable times might be. The scheduler attempts to satisfy all commitments, and when this is not possible it tries to satisfy the most important commitments. When commitments are not satisfied, the scheduler tries to suggest alternate commitments that it can satisfy.

The result is a set of schedules that satisfy particular commitments or suggest alternates. The decision maker can then choose to communicate information about which tasks will be completed and when to other agents, which can in turn rely on that information when producing their own schedules. Another use for commitments is to allow the scheduler to be easily controlled by a higher level agent that uses the scheduler to understand the lower level effects of high level decisions as described in the information gathering work of Zilberstein and Lesser [Zilberstein and Lesser, 1996].

Also part of the invocation of the scheduler is an indication of what the scheduler objectives should be. This allows priorities to be set indicating what combination of maximizing solution quality, increasing certainty and finishing as soon as possible is desired. For example, one invoker of the scheduler might want to maximize solution quality, while another may want a schedule that is most likely to achieve solution quality above a particular threshold.

Associated with each schedule is a detailed summary of what solution quality can be expected for every task in the TÆMS task structure, information about the uncertainties associated with schedule elements (including how likely it is that particular schedule elements will not complete execution before their deadline), and monitoring points at which the scheduler suggests ranges of performance that are acceptable.

Conclusions

We have described recent work in the area of design-to-time real-time scheduling. First we outlined how approximations of the sort we require tend to arise in applications. Then we described how uncertainty is represented and how it affects the scheduler. Finally we described parts of the interface between the scheduler and its invoker. Together these threads begin to show how the design-to-time approach can be used to solve real world problems in a reasonable, efficient way.

References

Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*. MIT Press, Cambridge, MA, 1996.

Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4):215–234, December 1993. Special issue on "Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior".

Keith S. Decker, Victor R. Lesser, and Robert C. Whitehair. Extending a blackboard architecture for approximate processing. *The Journal of Real-Time Systems*, 2(1/2):47–79, 1990.

S. Fujita and V. Lesser. Cooperative tasks in coarse grain search problems. CS Technical Report 96–28, Univ. of Massachusetts, 1996.

S. Fujita and V.R. Lesser. Centralized task distribution in the presence of uncertainty and time deadlines. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, Japan, 1996.

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

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

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

Alan Garvey. Design-to-time real-time scheduling. Ph.D. Dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA, February 1996.

V. Lesser, S. H. Nawab, and F. Klassner. IPUS: An architecture for the integrated processing and understand of signals. *Artificial Intelligence*, 77(1), 1995.

S. H. Nawab and E. Dorken. Efficient STFT computation using a quantization and differencing method. In *Proceedings of the 1993 IEEE International Conference on Acoustics,Speech, and Signal Processing*, volume 3, pages 587–590, Minneapolis, MN, April 1993.

S. Zilberstein and V. Lesser. Intelligent information gathering using decision models. CS Technical Report 96–35, Univ. of Massachusetts, 1996.