Task Interdependencies in Design-to-time Real-time Scheduling *

Alan Garvey and Marty Humphrey and Victor Lesser
Computer Science Department
University of Massachusetts
Amherst, MA 01003
EMAIL: garvey@cs.umass.edu

Abstract

Design-to-time is an approach to real-time scheduling in situations where multiple methods exist for many tasks that the system needs to solve. Often these methods will have relationships with one other, such as the execution of one method enabling the execution of another, or the use of a rough approximation by one method affecting the performance of a method that uses its result. Most previous work in the scheduling of real-time AI tasks has ignored these relationships. This paper presents an optimal design-to-time scheduler for particular kinds of relationships that occur in an actual AI application, and examines the performance of that scheduler in a simulation environment that models the tasks of that application.

Introduction

One of the major difficulties in the real-time scheduling of AI tasks is their lack of predictable durations. This difficulty occurs in non-AI systems, but it is especially prominent in AI problem-solving because of the inherent nondeterminism of most AI problem-solving techniques due to their extensive use of search. For this reason, most AI systems use some form of approximation to reduce the nondeterminism and make system performance more predictable.

At least two broadly different kinds of approximation algorithms have been examined. They are:

- *Iterative refinement*—where an imprecise answer is generated quickly and refined through some number of iterations. There are several variations including *milestone methods* where a procedure explicitly generates intermediate results as often as is deemed useful, and *sieve functions* where intermediate results are refined by running them through a series of functions (known as sieves) that improve the results [Liu *et al.*, 1991].
- *Multiple methods*—where a number of different algorithms are available for a task, each of which is capable of generating a solution. These algorithms emphasize different characteristics of the problem, which might be applicable in different situations. These algorithms also make tradeoffs of solution quality versus time.

The scheduling problem for approximate algorithms is to decide how to allocate processing time among approximations for different tasks so as to optimize the total performance of the system. Several approaches to this scheduling problem have been described in the literature [Dean and Boddy, 1988; Liu *et al.*, 1991; Russell and Zilberstein, 1991]. Nearly all of these approaches assume that tasks are either totally independent or have only hard precedence constraints between them. However, often AI applications do not consist of independent tasks, but rather of a series of interrelated subproblems whose consistent solution is required for an acceptable answer. The importance of taking these relationships into account in scheduling decisions has been observed in our work in sensor interpretation [Garvey and Lesser, 1993; Lesser and Corkill, 1983]. One important reason why other work has not focused on relationships is undoubtedly the difficulty of scheduling related tasks efficiently. While we don't offer a proof of it here duc to space limitations, it is evident that the scheduling problems we are investigating fall into the class of NP-Hard problems, as others have shown for similar problems not involving task interrelationships [Graham *et al.*, 1979; Liu *et al.*, 1991]. As we will discuss below, we have developed a scheduling algorithm for a specific class of approximation algorithms and task structures that in the worst case has exponential performance, but, in practice, is able to schedule tasks effectively.

Our new scheduling algorithm that exploits task interrelationships is appropriate for what we have called the *design-to-time* approach to real-time problem-solving [Decker *et al.*, 1990; Garvey and Lesser, 1993]. Design-to-time (a generalization of what we have previously called approximate processing [Lesser *et al.*, 1988]) is an approach to solving problems in domains where multiple methods are available for many tasks and satisficing solutions are acceptable. These methods make tradeoffs in solution quality versus execution time, and may only be applicable in particular environmental situations.

*This work was partly supported by NSF contract CDA 8922572, ARPA 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.

The methodology is known as design-to-time because it advocates the use of all available time to generate the best solutions possible. It is a problem-solving method of the type described by D'Ambrosio [D'Ambrosio, 1989] as those which "given a time bound, dynamically construct and execute a problem solving procedure which will (probably) produce a reasonable answer within (approximately) the time available."

Design-to-time can only be successful if the duration and quality associated with methods is fairly predictable. The predictability issue was investigated in detail in a previous paper [Garvey and Lesser, 1993] with the result that the predictability necessary for execution times is based on a complex set of factors that include how busy the agent is and how difficult it is for the agent to determine when a method is not performing as expected. An agent can tolerate uncertainty in its predictions if

- monitoring can be done quickly and accurately, so that when a task will not meet its deadline enough time remains to execute a faster method, or
- intermediate results can be shared among methods, so that when it is necessary to switch to a faster method the intermediate results generated by the previous method can be used, or
- there exists a fall back method that quickly generates an minimally acceptable solution.

The next section presents a model of task structures that supports satisficing real-time tasks. The following section describes a particular class of task structure, then presents an algorithm for scheduling that class and gives an example of that algorithm scheduling a set of task groups. That is followed by a section that examines the performance of that algorithm in a simulation environment. Finally, we summarize our results and discuss future directions.

Task Structures

This section defines a model of task structures that has the complexity necessary to describe the unpredictability of tasks and the interactions among tasks[1]. In this model a problem consists of a set of independent task groups. Each task group contains a structured set of dependent tasks. Task groups $\mathcal{T}$ occur in the environment at some frequency, and induce tasks T to be executed by the agent under study. Each task group has a deadline $D(\mathcal{T})$.

In this model the value of performing a task is known as the *quality* of the task. The term quality summarizes several possible properties of actions or results in a real system: certainty, precision, and completeness of a result, for example [Decker *et al.*, 1990]. Task group quality ($Q(\mathcal{T})$) is based on the *subtask* relationship. In the experiments described in this paper tasks accumulate quality using *minimum* or *maximum* functions, i.e., a task's quality at time t is either the minimum or maximum of the qualities of each of its subtasks at time t. This quality function is constructed recursively; each task group consists of tasks, each of which consists of subtasks, etc., until individual executable tasks (known as *executable methods*) are reached. Executable methods have a base level quality and duration, which in this work are generated randomly for the experimental evaluation, but are correlated with one another (i.e., higher quality methods tend to take longer than lower quality methods).

Besides task/subtask relationships, tasks can have other relationships to tasks in their task group. Many such relationships are possible including:

- *enables* constraints — Task A must be executed before Task B. This is usually a hard constraint.
- *facilitates* relationships [Decker and Lesser, 1991] — If Task A is executed before Task B, then Task B will have increased quality and/or decreased duration. This could result, for example, from Task A performing part of the work that would have been done by Task B.
- *hinders* relationships — If Task A is executed before Task B, then Task B will have decreased quality and/or increased duration. This could result, for example, from Task A using an approximation that reduces the precision with which Task B can be performed.

These relationships can affect the base level quality and duration of affected methods. In this work we have examined task structures that have acyclic enables and hinders constraints.

For each task in a task structure there may be multiple sets of subtasks that can be combined to solve the task, although a particular scheduling algorithm may not enumerate all such combinations. Each of these sets is known as a *method* for solving the task. At least some of these methods may involve approximations and thus be satisficing.

The scheduling problem for sets of task groups is to find an ordered set of executable methods that

- generate non-zero quality for each task group, T,
- maximize the total quality, $Q(\mathcal{T})$, of all task groups added together (possibly weighted by the *importance* of the task group, although that is not examined in this paper),
- do not execute any executable methods after the deadline of their task group, $D(\mathcal{T})$.

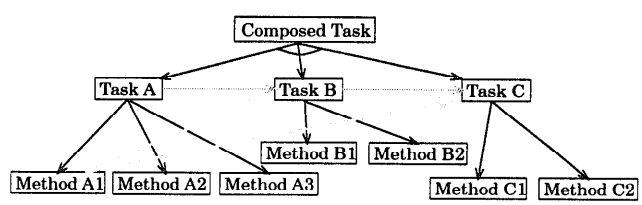

Figure 1: An example task group. The dark lines indicate subtask relationships. The thin gray lines represent enables constraints. The thick gray lines represent hindering constraints. The standard notation for minimum as *and* and maximum as *or* are used.

Figure 1 is an example of a simple task group. In this task group *Composed Task* is solved by solving each of

[1] A more detailed mathematical description of this model can be found in [Decker *et al.*, 1992].

Task A, *Task B* and *Task C* in order. (In order because of the enables constraints.) Each of these tasks has multiple solution methods available for solving it, where increasing method number means longer, more complete method. The thick gray lines represent hindering constraints from *Method A1* to each of *Task B* and *Task C*. This means that if *Method A1* (presumably a fast, imprecise method) is used to solve *Task A* then *Tasks B* and *C* will take longer to complete and/or produce lower quality results.

A Design-to-time Scheduler

This section describes an algorithm for scheduling the execution of executable methods in environments where:

- The task/subtask relationship forms a tree with a single root for each task group. This means that each task and method has exactly one supertask.
- Tasks generate quality using one of *minimum* (AND) or *maximum* (OR).
- Enables relationships may exist among the subtasks of tasks that accumulate quality using minimum. The enables relationships are mutually consistent (i.e., there are no cycles). This corresponds to the situation where there is a body of work that must be completed to satisfy a task and this work must be done in a particular order.
- Hinders relationships may exist in situations where enables may exist and an enabling subtask has a maximum quality accumulation function. In this situation there may be a hinders relationship from the lowest quality method for solving the subtask to the tasks that the subtask enables. This corresponds to the situation where using a crude approximation for a task can have negative effects on the behavior of tasks that use the result of the approximated task.

These environmental characteristics closely model characteristics seen in a sensor interpretation application. In particular, the enables relationships appear as requirements that low level data be processed before high level interpretations of that data are made, and the hinders relationships appear in the situation where fast, imprecise approximations of low level data processing can both increase the duration and decrease the precision of high level results [Garvey and Lesser, 1993; Lesser and Corkill, 1983].

The Algorithm

Briefly, this algorithm recursively finds all methods for executing each task in the task structure, pruning those methods that are superseded by other methods that generate greater or equal quality in equal or less time. In calculating the expected quality of a method it takes enables and hinders constraints into account. When it has found all unpruned methods for every task group, it orders the task groups by deadline and finds the combination of methods for the task groups that generates the highest total quality while meeting all deadlines. It then schedules the execution of each individual executable method using a simple algorithm that ensures that no enables constraints are violated and avoids hinders constraints if possible. If no schedule can be found that generates quality for all task groups, the scheduler returns a schedule that generates some quality for as many task groups as possible.

This algorithm works its way up from the leaves of the tree. In all examples in this paper, those leaves are executable methods, however, there is no reason why they could not be higher level tasks (with an estimated duration and quality) whose detailed execution is scheduled at a later time.

The optimality of this algorithm follows from the fact that it is effectively generating all possible alternatives, then choosing the ones that generate the maximum quality possible without missing any deadlines. As the alternatives for each task are generated, ones that could never be chosen because other alternatives exist that are *always* better are pruned. This pruning is effective only because we can make this determination locally, because of the constraints on where relationships can occur.

In the worst case this algorithm takes time exponential in the number of tasks, but, in practice, pruning and a clustering effect (to be described) usually make it much more efficient. The pruning reduces the number of alternatives that need to be considered for each task. In fact, pruning can reduce the number of alternatives for a task to no more than the number of distinct quality values possible. In the case where quality is a real number, this is not particularly helpful. However, if quality values are symbols from a small set of possible values or are somehow otherwise limited, this can significantly reduce the number of alternatives considered. As illustrated by the example in the next section, our experiments achieved a clustering effect by using small integer values for quality and combining them using minimum and maximum, resulting in a small set of possible quality values.

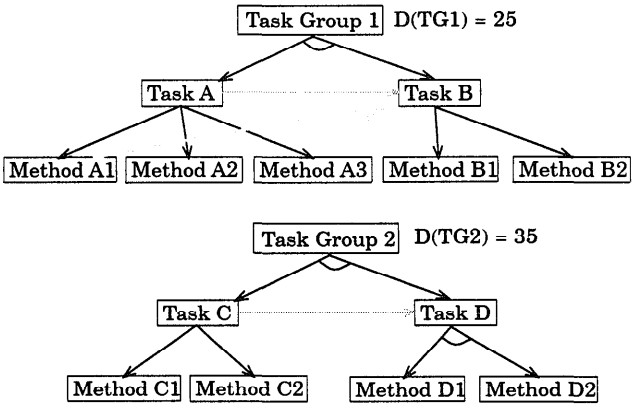

Figure 2: An example problem consisting of two task groups. The dark lines indicate subtask relationships. The thin gray lines represent enables constraints. The thick gray line represents a hinders constraint.

Method	Duration	Quality
Method A1	5	5
Method A2	7	7
Method A3	10	9
Method B1	9	8
Method B2	12	12
Method C1	4	4
Method C2	7	6
Method D1	6	5
Method D2	5	5

Table 1: Duration and quality for the executable methods in the example problem.

An Example

To describe the details of the algorithm more specifically we now show how it would schedule the two task groups shown in Figure 2 with associated durations and qualities shown in Table 1. Task group 1 (TG1) has both a hinders and an enables relationship, while TG2 has only an enables relationship. TG1 and TG2 have deadlines of 25 and 35 respectively. The hinders relationship has the effect of reducing quality by 50% and increasing duration by 25%.

First the algorithm recursively finds all alternatives for each element of the task structure. Each executable method has exactly one alternative, the method itself. Task's A, B, and C each accumulate quality using maximum, so they only need to execute one of their subtasks, giving them 3, 2, and 2 alternatives respectively[2]. Task D accumulates quality using minimum, so it has only one alternative, that which executes both of its subtasks. No pruning is possible in any of these situations. Finally, the algorithm finds the alternatives for each task group by combining alternatives from the associated subtasks. The possible alternatives for TG1 shown in Table 2.

In this case alternatives 1,2 and 4 can be pruned (as indicated by the lines through them) because other alternatives exist that can generate equal or higher quality in equal or shorter time. Note that the effects of the hinders relationship from Method A1 to Task B are shown in the reduced qualities and increased durations of Methods B1 and B2 in alternatives 1 and 2. Similarly the possible alternatives for TG2 (neither of which can be pruned) are shown in Table 3.

Finally, the alternatives for the entire set of task groups are shown in Table 4.

Alternatives 4, 5, and 6 can be pruned because TG2 does not meet its deadline of time 35. Alternative 3 can be pruned because it is redundant with Alternative 2. The scheduler chooses Alternative 2, which generates the maximum possible quality while meeting all deadlines. It then finds an ordering for the chosen alternative that meets all enables constraints, for example it could choose the schedule: A2, B1, C2, D1, D2.

[2]There is no need to consider alternatives that involve executing more than one of these subtasks, because no possible gain could result. However, in cases where such gain could result, for example when quality is accumulated in an additive fashion, all possible subgroupings must be considered.

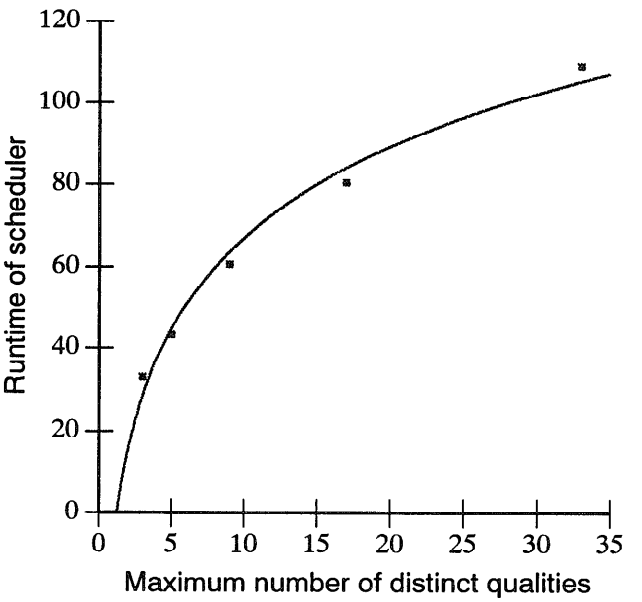

Figure 3: Maximum number of possible quality values versus the average runtime of the scheduler.

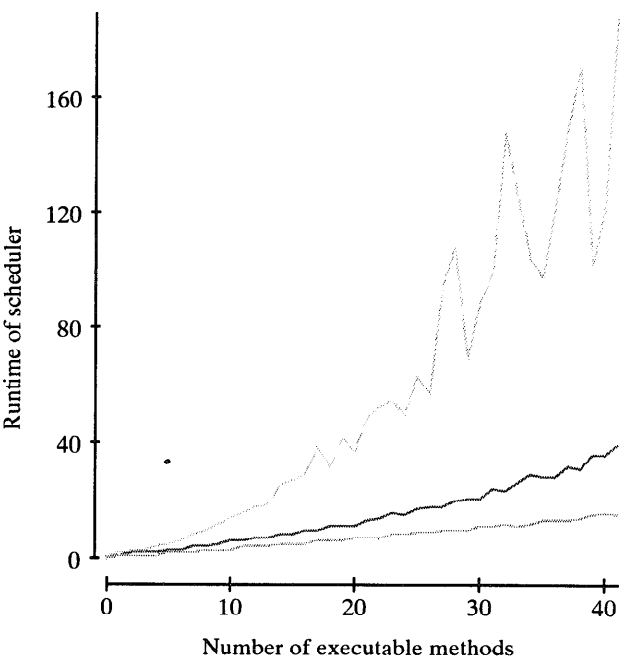

Figure 4: Number of executable methods versus the average runtime of the scheduler. The middle line is the median runtime. The upper and lower line are the 95th and 5th quartiles respectively.

ID	Set of methods	Expected quality	Expected duration
~~1~~	~~{Method A1, Method B1}~~	~~$\min(5, 0.5 * 8) = 4$~~	~~$5 + 1.25 * 9 = 16.25$~~
~~2~~	~~{Method A1, Method B2}~~	~~$\min(5, 0.5 * 12) = 6$~~	~~$5 + 1.25 * 12 = 20$~~
3	{Method A2, Method B1}	$\min(7, 8) = 7$	$7 + 9 = 16$
~~4~~	~~{Method A2, Method B2}~~	~~$\min(7, 12) = 7$~~	~~$7 + 12 = 19$~~
5	{Method A3, Method B1}	$\min(9, 8) = 8$	$10 + 9 = 19$
6	{Method A3, Method B2}	$\min(9, 12) = 9$	$10 + 12 = 22$

Table 2: Alternatives for Task Group 1.

ID	Set of methods	Expected quality	Expected duration
1	{C1, D1, D2}	$\min(4, 5, 5) = 4$	$4 + 6 + 5 = 15$
2	{C2, D1, D2}	$\min(6, 5, 5) = 5$	$7 + 6 + 5 = 18$

Table 3: Alternatives for Task Group 2.

Experimental Results

In order to be practically useful, a design-to-time scheduling algorithm needs to be subject to the same kind of controls that it expects from domain level tasks. In particular it needs to be able to tradeoff the quality of its schedules as a function of the time devoted to scheduling.

This section describes two measures of the performance of our scheduling algorithm as a function of the task structures it is scheduling. The first experiment measures the effect of the number of distinct possible quality values on the performance of the scheduler. The second experiment measures the effect of the size of the task structure (as reflected in the number of executable methods) on the performance of the scheduler.

Our experiments were conducted on randomly generated sets of task groups with enables and hinders relationships of the form described above. In the first experiment the number of task groups varied from 1 to 4 (to vary the size of the problems significantly); in the second experiment there was always 1 task group (to isolate the effect of the number of methods on scheduler performance). We controlled the size of the trees generated by having a maximum branching factor and a maximum depth—in these experiments the maximums were set to 5. We also controlled the likelihood that enables and hinders relationships would appear in situations where they were possible—these values were 50% and 100% respectively.

Figure 3 shows the effect of the maximum number of distinct quality values on the performance of the scheduler. This experiment was conducted by generating a task structure, then randomly assigning quality values to the executable methods by choosing them uniformly from the set of possible quality values. As this graph shows the runtime of the scheduler appears to increase in a logarithmic fashion as the number of possible quality values increases.

Figure 4 shows the effect of the number of methods in the task structure on the performance of the scheduler. This experiment was conducted by generating random task structures, scheduling them using the design-to-time scheduling algorithm, recording a number of statistics including both the runtime of the scheduler and the number of executable methods in the task structure. We then collected together all of the data from each of several thousand runs and found the average runtime for each distinct number of executable methods. This suggests that the performance of the scheduler is polynomial in the number of executable methods, and that performance becomes significantly less predictable as the number of methods increases.

The results of these experiments suggest that a design-to-time scheduler could control its own performance by dynamically modifying the task structures it is scheduling. The result relating to the number of possible quality values suggests that a scheduler could reduce its runtime by reducing the number of distinct quality values in the task structure it is scheduling. It could do this by bucketizing the quality values into a smaller set of buckets and treating all quality values in the same bucket as identical. This approximation will have the effect of reducing the precision of the final schedule, because the scheduler will not consider fine-grained distinctions among methods. However, because it does not throw away any methods, the scheduler will always find a schedule in those situations where it would have found a schedule originally; it just might not be as good a schedule.

The result concerning the number of methods suggests that if a scheduler could reduce the number of methods it had to consider it could reduce its runtime. It could do this by reducing the number of methods considered for tasks that generate quality in a maximum fashion. It is probably best to not remove the fastest method or the highest quality method, but methods in between can be ignored. This approximation will have the effect of reducing the completeness of the schedule. Not all possible schedules will have been considered, so the best schedule may not be found. However, if the scheduler does not throw away the fastest methods, it will always be able to find a schedule in those situations where it could find one originally.

Another approximation that we have thought of, but not

ID	Set of methods	Expected quality	Expected finish times
1	{A2, B1, C1, D1, D2}	$7+4=11$	$16, 16+15=31$
2	{A2, B1, C2, D1, D2}	$7+5=12$	$16, 16+18=34$
~~3~~	~~{A3, B1, C1, D1, D2}~~	~~$8+4=12$~~	~~$19, 19+15=34$~~
~~4~~	~~{A3, B1, C2, D1, D2}~~	~~$8+5=13$~~	~~$19, 19+18=37$~~
~~5~~	~~{A3, B2, C1, D1, D2}~~	~~$9+4=13$~~	~~$22, 22+15=37$~~
~~6~~	~~{A3, B2, C2, D1, D2}~~	~~$9+5=14$~~	~~$22, 22+18=40$~~

Table 4: Alternatives for set of task groups.

yet investigated carefully, is to schedule without considering hinders relationships. Preliminary investigation suggests that this has the positive effect of reducing the runtime of the scheduler, but the negative effect of having the scheduler occasionally produce schedules that do not meet deadlines (because the scheduler mis-estimates the duration of executable methods). One approach to this problem is to monitor the execution of methods. For a more detailed discussion of monitoring see [Garvey and Lesser, 1993].

We intend to investigate these issues and build schedulers that take their own performance into account when scheduling. This should result in schedulers for design-to-time tasks that are themselves design-to-time in character.

Conclusions and Future Work

Previously we have examined the scheduling of tasks with multiple methods, but few task interdependencies, in both the Distributed Vehicle Monitoring Testbed (DVMT) and in a simulation environment [Garvey and Lesser, 1993]. Currently we are working on developing a more sophisticated scheduler that efficiently schedules more complex task structures that include additional types of relationships between tasks such as *facilitates*, another relationship that occurs in the DVMT environment. We are also looking at scheduling for distributed agents that are cooperating to solve complex, real-time problems. Finally, we intend to study this scheduler in a sound understanding application [Lesser *et al.*, 1993].

More generally, we would like to investigate the issues raised in the Experimental Results section by moving in the direction of building design-to-time schedulers that can control their own performance. These schedulers should be able to trade off the quality of the schedules they produce with the time it takes to produce them. This will have the effect of creating schedulers for design-to-time tasks that have a design-to-time character.

References

D'Ambrosio, Bruce 1989. Resource bounded-agents in an uncertain world. In *Proceedings of the Workshop on Real-Time Artificial Intelligence Problems*, IJCAI-89, Detroit.

Dean, T. and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota. 49–54.

Decker, Keith S. and Lesser, Victor R. 1991. Analyzing a quantitative coordination relationship. COINS Technical Report 91–83, University of Massachusetts. To appear in the journal *Group Decision and Negotiation*, 1993.

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

Decker, Keith S.; Garvey, Alan J.; Lesser, Victor R.; and Humphrey, Marty A. 1992. An approach to modeling environment and task characteristics for coordination. In Petrie, Charles J. Jr., editor 1992, *Enterprise Integration Modeling: Proceedings of the First International Conference*. MIT Press.

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

Graham, R.L.; Lawler, E. L.; Lenstra, J. K.; and Kan, A. H. G. Rinnooy 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. In Hammer, P. L.; Johnson, E. L.; and Korte, B. H., editors 1979, *Discrete Optimization II*. North-Holland Publishing Company.

Lesser, Victor R. and Corkill, Daniel D. 1983. The distributed vehicle monitoring testbed. *AI Magazine* 4(3):63–109.

Lesser, Victor R.; Pavlin, Jasmina; and Durfee, Edmund 1988. Approximate processing in real-time problem solving. *AI Magazine* 9(1):49–61.

Lesser, Victor; Nawab, Hamid; Gallastegi, Izaskun; and Klassner, Frank 1993. IPUS: An architecture for integrated signal processing and signal interpretation in complex environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*.

Liu, J. W. S.; Lin, K. J.; Shih, W. K.; Yu, A. C.; Chung, J. Y.; and Zhao, W. 1991. Algorithms for scheduling imprecise computations. *IEEE Computer* 24(5):58–68.

Russell, Stuart J. and Zilberstein, Shlomo 1991. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia. 212–217.