Tradeoffs in the Design of On-Line Systems

Lloyd Greenwald
Department of Computer and Information Science
University of Pennsylvania
200 S. 33rd St., Philadelphia, PA 19104-6389
lgg@linc.cis.upenn.edu

Thomas Dean
Department of Computer Science
Brown University
Box 1910, Providence, RI 02912
tld@cs.brown.edu

Abstract

We present a general framework for analyzing tradeoffs when designing systems in which an agent with limited computational resources is required to respond in a timely manner to situations arising in a dynamic environment. These tradeoffs are characterized in terms of architectural constraints that limit the computational alternatives available to the agent. We apply the framework to a design problem in which the designer is given a set of controllers, each of which is capable of responding to a subset of situations, and a model of the run-time dynamics that can be used to predict the sets of situations that might arise over time. Exactly one controller is active at any moment, and a fixed time delay is required to activate a controller. Additional architectural constraints limit the number of controllers that are capable of being activated. The objective is to design (off-line) a strategy for activating and deactivating controllers that guarantees that there will be an active controller capable of responding no matter what situation arises. We provide an asymptotic analysis for this design problem, describe exact and approximate algorithms for the off-line design, and sketch how the framework can be applied to real-time avionics scheduling. We argue that the architectural constraints of this design problem model a rich class of planning and control systems.

Introduction

This paper is concerned with the general problem of designing a system in which an agent with limited computational resources is required to respond in a timely manner to situations arising in a dynamic and uncertain environment. Special cases include designing the controller for a robot planetary explorer or designing the avionics system for a commercial aircraft.

Existing approaches to solving such design problems differ in their characterization of the computational resources available to the system at run time and the knowledge of the environment available to the designer at design time. Even in cases in which these characterizations are precise, the resulting design problems often defy detailed analysis. We provide a general framework for characterizing these design problems that admits a careful analysis and provides insight into the tradeoffs inherent in resource-bounded decision making.

Our framework stresses three basic components:

1. *architectural constraints* that directly influence design and indirectly influence the run-time behavior of the system,
2. *behavioral requirements* that specify the desired behavior of the system, and
3. *predictive models* that are used at design time to anticipate run-time behavior.

In this paper, *design problem* refers to the problem of designing a system that meets a set of behavioral requirements subject to a set of architectural constraints, and given a predictive model. We focus on design problems in which the designer is presented specifications for each of these components and is required to configure a run-time system. In these cases, any design that satisfies the architectural constraints and meets the behavioral requirements is considered satisfactory. In the general case, the design objectives allow the designer to trade off the costs of realizing a particular design against the benefits of the resulting behavior.

To demonstrate the utility of our framework, we investigate an interesting set of architectural constraints based on an analogy to hierarchical memory models. A simple two-level hierarchy is used to explore tradeoffs involving fast reactions and slower deliberations. These architectural constraints induce specific patterns regarding how situations evolve as a consequence of the system interacting with the environment. We consider algorithms that exploit these patterns to make efficient use of computational resources.

In the remainder of this paper, we describe the general framework, introduce the restricted class mentioned above, and provide algorithms and analytical results pertaining to this class. The practical application of the framework and techniques of this paper has been demonstrated through the design and analysis of a real-time avionics scheduling solution (Greenwald 1997).

Designing On-Line Systems

A common approach to modeling dynamical systems (Dean & Wellman 1991) is to view the interaction of on-line system and environment as a feedback loop. In the remainder of this paper, we use the following terms in narrow technical sense.

- *on-line system* (or just *system*) refers to the end product of design (the "controller" in automated control or the combination of a "planner" and "executor" in AI),
- *environment* refers to the world with which the system interacts (the "plant" in conventional control systems),
- *situation* refers to a complete description of the environment at an instant in time (the "state" in dynamical systems theory),
- *output* refers to the response of the system in a given situation (an "action" in AI planning), and
- *input* refers to information regarding a situation that is available to the system at run time in choosing an output (an "observation" in dynamical systems theory).

An *on-line* system may require varying amounts of time to produce an output in response to an input or sequence of inputs. At one extreme, a "deliberative" AI planning system might require a large time delay to produce an output after receiving an input. To justify such a delay, a traditional approach is to assume that the situation does not change during deliberation. At the other extreme, a "reactive" control system might assume the situation changes rapidly and requires a rapid response to each new input. In general, systems combine fast reactive computations with slower deliberative computations.

To quantify the tradeoffs inherent in resource-bounded decision making, our framework expresses on-line computation as the interaction of *circuits*. A circuit is a unit of computation with well-specified time and space requirements and input/output behavior. Diverse computations can be consistently compared when expressed as circuits. For example, a circuit can express lookup tables, decision trees, or deliberative planning algorithms. The computational characteristics of a circuit are represented in terms of its depth (time) and number of gates (space).

We distinguish two types of circuit:

1. *control circuits* capture fast reactive computations that produce an output from an input in a bounded amount of time, and
2. *strategy circuits* capture slower deliberative computations that do not directly generate outputs; a strategy circuit indirectly affects output by manipulating control circuits, including dynamically allocating computational resources in response to input.

Consider designing an on-line system for controlling the avionics hardware of an aircraft that must dynamically adjust to changing weather conditions. This system may consist of control circuits designed to handle specific subsets of weather conditions and a strategy circuit to determine which circuit controls the hardware at any given time.

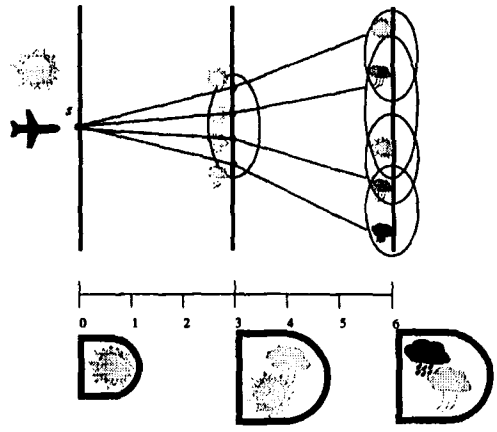

Figure 1: Controlling aircraft avionics hardware with weather-specific control circuits.

In Figure 1, three different weather-specific control circuits are depicted, corresponding to sunny conditions, combinations of sun and wind, and combinations of wind and rain. As the aircraft travels, a strategy circuit monitors current weather conditions and reasons about future weather conditions. The strategy circuit uses this information and a model of circuit delay to determine which circuit should be controlling the avionics hardware at any given time.

In general, an on-line system may be expressed as the the interaction of multiple, possibly concurrent, circuits. Combinations of control and strategy circuits can represent a rich class of planning and control systems.

A Circuit-Paging Architecture

Architectural constraints refer to the constraints governing the design of an on-line system. These constraints limit the computational alternatives available to the agent, and provide a tradeoff between expressiveness and design complexity. We specify a set of constraints based on an analogy to hierarchical memory models. These constraints allow us to express an interesting class of planning and control systems while admitting a careful analysis of the resulting design problems.

In hierarchical memory models, fast low-capacity levels of memory, such as processor cache, combine with slow high-capacity levels, such as main memory. Since the capacity of fast cache is limited, data is distributed and transferred across levels of the hierarchy in an attempt to achieve overall behavior comparable

to the speed of the fastest level with the capacity of the slowest level.

Similarly, in on-line decision making, processing is distributed between fast reactive computations and slower deliberative computations, including design-time computations. The results of processing are stored and transferred across the hierarchy in order to achieve the responsiveness of the fastest reactive computations with bounded on-line resources. Figure 2 depicts a two-level *circuit-paging* architecture to conceptually model these tradeoffs.

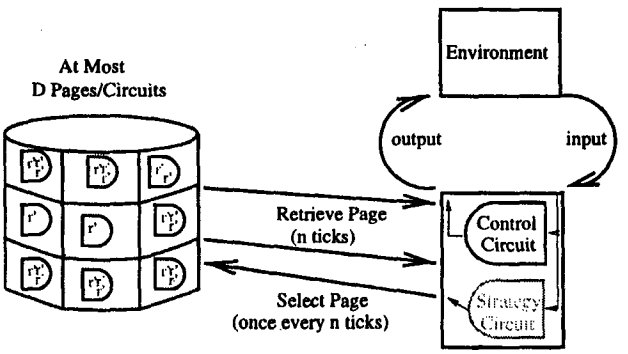

Figure 2: An architecture for paging control circuits.

The circuit-paging architecture studied in this paper consists of a secondary store capable of storing at most D pages/circuits and a fast cache of M bits. An initial control circuit π_0 and a fixed paging strategy circuit γ are loaded into fast cache prior to decision making. Every n ticks, the strategy circuit observes the input and chooses a control circuit to activate n ticks into the future. During each n-tick interval, the active control circuit observes the input at each tick and generates an output by the next tick. Concurrently, the control circuit selected for the next interval is paged into fast cache. An example execution in which $n = 3$ is depicted in Figure 3.

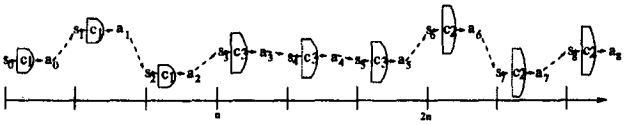

Figure 3: Active control circuit changes every 3 ticks.

To simplify the presentation, we assume a *fully observable* dynamical system, *i.e.* situations are directly observable as inputs. The general framework and specific analytical techniques of this paper do not require this assumption.

A control circuit is defined as a partial function from situations to outputs such that the *domain* V_S of circuit c_S is the set of situations for which it can generate an output in one time tick. A circuit has no defined response to situations outside its domain. If the currently active control circuit observes a situation outside its domain, we say that a *page fault* has occurred.

A circuit-paging architecture may conceptually model many forms of bounded-resource on-line decision making. Tradeoffs involving the capabilities of control circuits and their resource requirements model the effects of bounded resources on decision making.

Paging time models the on-line time delay for deliberation. Given that each control circuit models a particular form of deliberation, the paging time for that circuit models the corresponding deliberation delay. Over all forms of deliberation composing an on-line system, the fixed retrieval interval models a worst-case on-line deliberation delay of at most n ticks.

Understanding Execution-Time Behavior at Design Time

We look at the problem of designing a system that meets a set of behavioral requirements, subject to a set of architectural constraints, and given a predictive model.

- *trajectory* refers to a sequence of input/output pairs,
- *behavior* refers to the set of all trajectories that may result from a given system interacting with a given environment,
- *behavioral requirements* refer to a set of requirements with respect to a behavior, and
- *predictive model* refers to the knowledge available to the designer for anticipating the behavior of the system.

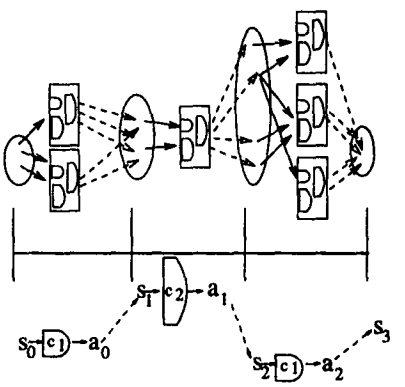

Figure 4: An example behavior.

In the general case, the predictive model is nondeterministic. Often the best the designer can do is determine sets of possible situations. Given a particular situation at time t, the set of possible situations at time $t + \Delta$ depends on how the system responds in the interval $[t, t + \Delta]$ which depends on what control circuits are active during that interval. The upper graphic in Figure 4 illustrates the sets of situations possible at various times given different possible active

control circuits. The lower graphic in Figure 4 illustrates the special case in which the control circuits are completely determined, the initial state is known with certainty, and the environment is deterministic.

The behavior of an on-line system must be modeled accounting for any limitations involving computational resources. Consider an avionics system that must choose between a long but safe course and a short course that might possibly run into a dangerous thunderstorm. A behavioral requirement that does not tolerate risk will favor the former, despite the fact that most trajectories involving the latter course are preferable to all trajectories involving the former.

Now, consider a third alternative of setting a course toward the possible thunderstorm with the option of changing course based on observations as the aircraft approaches the possible storm area. In this case, the time required to sense conditions and deliberate about changing course must be accounted for by the predictive model. If the time delay cannot be satisfactorily bounded at design time, the longer course will still be preferable. If the delay is small enough, however, the designer can ensure that the plane will turn aside before entering the storm area.

The following predictive model concisely characterizes the behavior of an on-line system combining circuit-paging architectural constraints and the behavioral requirement that page faults are not to be tolerated (zero page faults). Given that we have only nondeterministic knowledge of the outputs of the on-line system, we model behavior by looking strictly at the possible transitions between situations over time, independent of the outputs affecting these transitions. The components of this model are depicted in Figure 5 and in the following terms.

P_s^k: the set of situations that are reachable k-ticks into the future, starting from situation $s \in \mathcal{S}$;[1]

$R_s^n = \cup_{\{n \le k < 2n\}} P_s^k$: the set of situations that may occur between n and $2n$ ticks after having started in s (lower view of Figure 5); and,

$F_s^n = \cup_{\{j \ge 0\}} P_s^{nj}$: the set of situations that may occur in exactly nj ticks, for all integers $j \ge 0$, after having started in s (upper view of Figure 5 depicts $P_s^{nj}, j \in \{1,2\}$); for any initial set of situations $P^0 \in 2^{\mathcal{S}}$, $F^n = \cup_{\{s \in P^0\}} F_s^n$.

Analytical Techniques

Given any initial set of possible situations $P^0 \in 2^{\mathcal{S}}$, F^n represents all situations that might occur during execution when some paging decision is made. In making a paging decision in response to input s, R_s^n represents the situations that may occur while the paged control circuit is active. Thus, for each $s \in F^n$, a zero page fault paging strategy must specify a control circuit to cover R_s^n. Figure 6 illustrates this relationship by graphically by composing the two views of Figure 5.

[1] This model technically requires that control circuits are both *Markovian* (output depends strictly on current situation) and *stationary* (output independent of time), so that the resulting system implements a *nondeterministic Markov stationary policy, i.e.* a mapping from situations to sets of possible outputs.

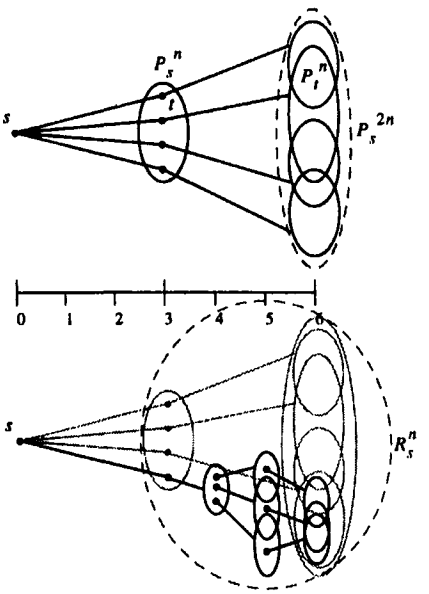

Figure 5: Transition structure for concisely characterizing behavior.

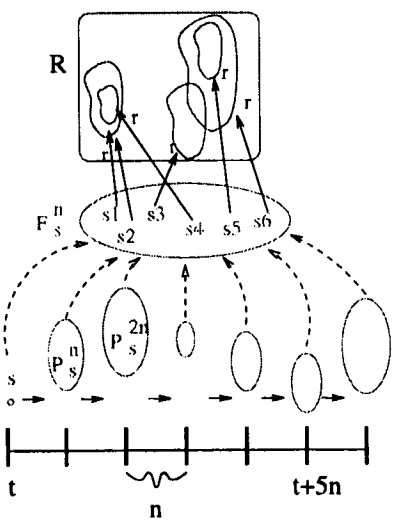

Figure 6: Predictive model reveals covering problem.

The following theorem specifies the necessary and sufficient conditions for determining whether or not a paging strategy exists that guarantees zero page faults, and such that all control circuits available for paging fit into secondary storage.

Theorem 1 *Given P_s^k, $P^0 \in 2^{\mathcal{S}}$, a collection of con-*

trol circuits C, each $c_S \in C$ covering domain $V_S \in 2^S$, and a secondary storage of size D, a zero page fault paging strategy *exists if and only if there exists a subset $C' \subseteq C$ of size $|C'| \leq D$ such that, for each $s \in F^n$, $R_s^n \subseteq V_S$ for some circuit $c_S \in C'$.*

Unfortunately, the decision problem of Theorem 1 is NP-complete by reduction from minimum set cover (Greenwald 1997). Additionally, approximating minimum set cover to any constant is also NP-complete. Nevertheless, approximation algorithms for minimum set cover exist (Johnson 1974).

A subset $Y' \subseteq Y$ is a *minimum set cover* for the finite set X if and only if Y' is the smallest cardinality subset of Y such that every element of X belongs to at least one member of Y'. We obtain the following theorem by transforming the decision problem of Theorem 1 to a minimum set cover problem.

Theorem 2 *Given set $R = \{R_s^n | s \in F^n\}$ and collection C, if $C' \subseteq C$ is a set cover for R, then there exists a zero page fault paging strategy for any secondary storage capable of storing at least $|C'|$ circuits.*

Greenwald (Greenwald 1997) provides an algorithm for finding a subcollection C' satisfying Theorem 2, based on a greedy minimum set cover approximation algorithm. The resulting algorithm is polynomial in the number of situations $|\mathcal{S}|$ and circuits $|C|$. By Theorem 1, this algorithm proves the existence of a zero page fault paging strategy when $|C'| \leq D$. Additionally, it is shown that such a strategy does not exist if $|C'| > D\mathcal{H}(|R|)$.[2]

The difficulty of the design problem provides motivation for looking more carefully at possible tradeoffs. The framework of this paper allows these tradeoffs to be explicitly expressed and evaluated at design time.

Theorem 1 provides insight into the combinatorics of the design-time task. We now introduce a more detailed model of the architectural constraints. We do so by relating the bound on secondary storage D to other resource measures.

Given a single-tick bound on control circuit time delay, a circuit's complexity may be measured solely with respect to its size. In this model the size of a circuit, $B_\delta(c_S)$, is a function of the situations in its domain. This defines a tradeoff between the functionality of a circuit, in terms of variety of situation-specific outputs, and the space required to store the circuit.

The following functions measure worst-case time (space) resource requirements with respect to a subcollection of control circuits $C' \subseteq C$.

$T_\delta^*(C')$ $(B_\delta^*(C'))$: ticks (bits) to execute (store) any circuit in C';

$T_\gamma(d,k)$ $(B_\gamma(d,k))$: ticks (bits) to execute (store) a strategy circuit with situation domain of size k and circuit range of size d; and,

[2] $\mathcal{H}(d)$ is the d^{th} harmonic number.

T_ρ ticks to retrieve one bit of data.

We assume a simple buffering scheme such that the active control circuit and subsequently paged control circuit may be stored in fast cache simultaneously. Given that (1) a control circuit must execute in at most one tick, (2) two control circuits plus the fixed strategy circuit must be stored simultaneously in M bits of on-line storage, and (3) the combined process of executing the strategy circuit and retrieving the page must execute in at most n ticks, the following *executability* equations relate the time-space tradeoffs of a circuit-paging system:

$$T_\delta^*(C') \leq 1 \quad (1)$$

$$2 * B_\delta^*(C') + B_\gamma(d, |F^n|) \leq M \quad (2)$$

$$T_\gamma(d, |F^n|) + T_\rho * B_\delta^*(C') \leq n \quad (3)$$

Equations 1, 2, 3 capture tradeoffs between the number of control circuits that are available for paging and the resource requirements of these circuits. These equations may also be viewed as modeling tradeoffs between deliberation variability and deliberation delay.

Under the assumption that the physical limitations of secondary storage are not binding, the design-time task is summarized by the following theorem.

Theorem 3 *Given C' satisfying Theorem 1, a zero page fault paging strategy for C' is* executable *if and only if $|C'| \leq D$, where $D = \max d$ satisfying Equations 1, 2, 3.*

Theorems 1 and 3 are interdependent. In other words, we require not only that a zero page fault strategy exists, but that such a strategy can be executed within bounded on-line resources.

It is interesting to note that we can eliminate the combinatorics of the design-time task in special cases. In particular, for *any* subcollection C' covering the set $\{R_s^n | s \in F^n\}$, a zero page fault strategy can be made executable by adding on-line resources until $|C'| \leq D$, for $D = \max d$ satisfying Equations 1, 2, 3.

To demonstrate, assume that the worst-case time to execute a strategy circuit is independent of the number of pages in secondary storage, *i.e.* $T_\gamma(d,k) = T_\gamma(1,k), \forall d$. For example, we might assume a strategy circuit constructed as a parallel combination of d 1-page circuits, with negligible additional delay. Furthermore, assume that $T_\gamma(1, |F^n|) + T_\rho * B_\delta^*(C') \leq n$. Equation 2 is then binding and for any set covering C' the amount of on-line cache required to guarantee an executable zero page fault strategy is

$$M \geq 2 * B_\delta^*(C') + B_\gamma(|C'|, |F^n|).$$

In general, we might apply standard search schemes to find an executable zero page fault strategy, if one exists. One method (Greenwald 1997) is to construct a mixed integer linear program (MILP) that combines set covering and executability constraints, and apply standard branch and bound enumeration algorithms to the MILP.

Related Work

Theoretical work on resource-bounded problem solving include development of models of rationality that take deliberation costs into account (Good 1971), flexible algorithms that adjust to varying resource limitations (Dean & Boddy 1988; Horvitz 1987; Shih, Liu, & Chung 1989), and techniques that provide design-time optimization of execution-time behavior under these algorithms (Dean & Boddy 1988; Horvitz 1987; Russell & Wefald 1991; Russell & Subramanian 1995; Zilberstein 1993).

Systems have been designed that rely on fast on-demand reaction (Agre & Chapman 1987), design-time pre-compilation of fast reaction circuits for all possible scenarios (Schoppers 1987; Nilsson 1994), or design-time pre-compilation of circuits with behavior-driven timing guarantees (Kaelbling 1988). Some systems provide on-line deliberation by relying on environment locality assumptions (Dean *et al.* 1993) or the ability to guarantee control over uncertainty (Musliner 1993). Many systems have been designed that provide heuristic hybrid combinations of fast reaction and slow deliberation (Gat 1991; Simmons 1991).

Discussion

This paper presents a general framework for analyzing the problem of designing on-line systems that meet behavioral requirements, subject to architectural constraints, and given a predictive model. We develop a particular set of architectural constraints based on an analogy to hierarchical memory and analyze the resulting design problems. By studying the corresponding design problem, we are able to provide insight into the tradeoffs in designing on-line systems.

Our framework is unique in explicitly considering sequences of actions that are interdependent, in the face of exogenous uncertainty. Prior approaches do not guarantee that any progress will be made across deliberation intervals. We provide specific behavior guarantees, that are suitable for hard real-time applications. The framework and techniques of this paper can be applied to understanding the behavior of systems that combine deliberation and reaction; systems that previously defied analysis.

The circuit-paging architecture provides an explicit model of tradeoffs that are implicitly embedded in many on-line systems. In this architecture, control circuits are paged by a closed-loop paging circuit fixed at design time, paging intervals are of fixed duration, and active control circuits are independent across intervals. Extensions to this architecture include paging multiple strategy circuits, paging at variable or indeterminate duration intervals, and incorporating persistence and dependence across intervals into the circuit models.

References

Agre, P. E., and Chapman, D. 1987. Pengi: An implementation of a theory of activity. In *Proceedings AAAI-87*, 268–272. AAAI.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings AAAI-88*, 49–54. AAAI.

Dean, T., and Wellman, M. 1991. *Planning and Control.* San Mateo, California: Morgan Kaufmann.

Dean, T.; Kaelbling, L.; Kirman, J.; and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *Proceedings AAAI-93*, 574–579. AAAI.

Gat, E. 1991. Integrating reaction and planning in a heterogeneous asynchronous architecture for mobile robot navigation. *SIGART Bulletin* 2(4).

Good, I. J. 1971. Twenty-seven principles of rationality. In Godambe, V. P., and Sprott, D. A., eds., *Foundations of Statistical Inference.* Holt, Rinehart, Winston. 108–141.

Greenwald, L. G. 1997. *Analysis and Design of On-line Decision-Making Solutions for Time-Critical Planning and Scheduling Under Uncertainty.* Ph.D. Dissertation, Brown University, Providence, RI.

Horvitz, E. J. 1987. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence.*

Johnson, D. 1974. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences* 9:256–278.

Kaelbling, L. P. 1988. Goals as parallel program specifications. In *Proceedings AAAI-88*, 60–65. AAAI.

Musliner, D. J. 1993. *CIRCA: The Cooperative Intelligent Real-Time Control Architecture.* Ph.D. Dissertation, The University of Michigan, Ann Arbor, MI.

Nilsson, N. 1994. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research* 1:139–158.

Russell, S. J., and Subramanian, D. 1995. Provably bounded-optimal agents. *Artificial Intelligence Research* 2:575–609.

Russell, S. J., and Wefald, E. H. 1991. *Do the Right Thing: Studies in Limited Rationality.* Cambridge, MA: MIT Press.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings IJCAI 10*, 1039–1046. IJCAII.

Shih, W.-K.; Liu, J. W. S.; and Chung, J.-Y. 1989. Fast algorithms for scheduling imprecise computations. In *Proceedings of the Real-Time Systems Symposium*, 12–19. IEEE.

Simmons, R. 1991. Coordinating planning, perception, and action for mobile robots. *SIGART Bulletin* 2(4).

Zilberstein, S. 1993. *Operational Rationality through Compilation of Anytime Algorithms.* Ph.D. Dissertation, University of California at Berkeley.