Optimizing Resource Utilization in Planetary Rovers

Shlomo Zilberstein
Computer Science Department
University of Massachusetts
Amherst, MA 01003 U.S.A.
zilberstein@cs.umass.edu

Abdel-Illah Mouaddib
CRIL-IUT de Lens-Université d'Artois
Rue de l'université, S. P. 16
62307 Lens Cedex France
mouaddib@cril.univ-artois.fr

Abstract

Autonomous planetary rovers must operate under tight constraints over such resources as operation time, power, storage capacity, and communication bandwidth. To maximize scientific return, the rover is given multiple methods in which to accomplish each step of a plan. The different alternatives offer a tradeoff between resource consumption and the quality of the outcome. We show how to choose the best way to execute a task based on the availability of resources, the progress made with the task so far, and the remaining workload. Each task is controlled by a precompiled policy that factors the effect of the remaining plan using the notion of an opportunity cost.

Introduction

This paper is concerned with the design of a reactive meta-level controller that can optimize the operation of autonomous planetary rovers. Such rovers operate under tight resource constraints such as power, storage capacity, and communication bandwidth (Bresina *et al.* 1999; Washington *et al.* 1999). The time available to carry out experiments is limited as is the overall lifespan of the rover. The amount of power that is available (between recharges) is limited and must be carefully managed. Storage capacity to be used for raw data and processed data before transmission to a control center is also limited. Some of the resources are renewable: batteries could be recharged and storage space could be freed once the data is transmitted. Our focus in this paper is on optimal management of multiple resources between periods in which they can be renewed. The solution takes into account the high level of uncertainty regarding the consumption of resources by the rover's activities. For example, there is uncertainty about the amount of power and time required to bring the rover to a certain location, and there is uncertainty about the amount of storage that will be needed for a sequence of compressed images.

The combination of scarce resources and a high level of uncertainty present a complex meta-level control problem. The question is how to decide quickly during execution time which tasks should be executed and how to revise these decisions based on the actual progress being made, the availability of resources, and the remaining workload. Our approach to this problem is based on mapping each primitive activity (such as navigation, taking pictures, conducting experiments, or on-board data analysis) into a progressive processing task structure that specifies alternative ways to accomplish each aspect of the activity. For example, Figure 1 illustrates a possible task structure for taking a picture of a certain object. First the object must be located. Then, the rover may approach the object (either getting within an acceptable distance or an optimal distance for picture taking) and aim the camera. The picture can be taken at any one of several resolutions and then compressed at different levels of compression. The choices made along the execution of the task will affect the level of resource consumption as well as the quality of the outcome.

Progressive processing task structures make it possible for a system to trade off between resource consumption and quality of result (Mouaddib 1993; Mouaddib and Zilberstein 1997; 1998). The framework described in this paper is based on a similar approach we have developed for dynamic composition of information retrieval techniques (Zilberstein and Mouaddib 1999). The control of autonomous planetary rovers, however, has several different characteristics. First, we have multiple resources to monitor rather than one (time is the only resource monitored in previous work). Second, the reward structure depends on the ability to maximize scientific return with limited resources, but minimizing resource consumption in itself is not an explicit goal (minimizing response time is an explicit goal in the information retrieval application). Finally, unlike the information retrieval application, the set of tasks to be performed over a given time period is relatively stable (in previous work we considered a dynamic set of tasks with frequent updates). The specific characteristics of the rover control problem raise several fundamental challenges.

1. Handling *multiple* resources rather than execution time only.
2. Handling the *dependency* of quality and resource consumption on the intermediate quality or state.
3. Handling a *flexible task structure* in which some levels include several alternatives or optional steps.
4. Selecting the "best" set of methods in a dynamic en-

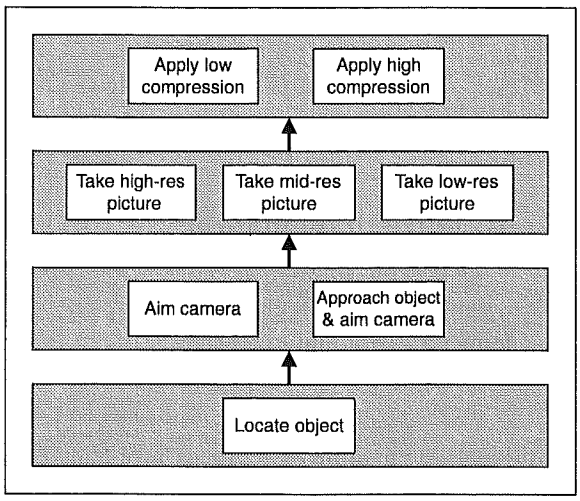

Figure 1: Illustration of a progressive processing task for taking a picture of an object

vironment taking into account the progress already made, the availability of resources, and the entire plan.

There are a number of complementary research efforts designed to develop planning and execution architectures for spacecraft systems (Blackmon *et al.* 1999; Bresina *et al.* 1999; Estlin *et al.* 1999; Muscettola *et al.* 1998) and, more generally, for real-time autonomous systems (Bonasso *et al.* 1996; Musliner *et al.* 1993). The work described in this paper is designed to complement these efforts by developing a reactive approach to managing multiple resources under a high level of uncertainty.

The ability to dynamically adjust computational effort based on the availability of computational resources has been studied extensively by the AI community since the mid 1980's. These efforts have led to the development of a variety of techniques such as *anytime algorithms* (Dean and Boddy 1988; Zilberstein and Russell 1996), *design-to-time* (Gravey and Lesser 1993), *flexible computation* (Horvitz 1988), *imprecise computation* (Liu *et al.* 1991), and *progressive reasoning* (Mouaddib 1993; Mouaddib and Zilberstein 1997). This work extends this flexibility to task structures that include both computational actions and physical actions. Section 2 gives a formal definition of the problem. We then solve the problem in two steps. In Section 3, we develop an optimal solution for a single task, ignoring the fact that additional tasks are waiting for execution. Section 4 shows how to handle multiple tasks by factoring the effect of the remaining plan using the notion of an opportunity cost. In section 5 we address some open problems and current work that examines the effectiveness this approach. We conclude with a summary and brief discussion of related work.

The control problem

This section defines an enhanced form of progressive processing task structures and the corresponding meta-level control problem. Each plan assigned to a rover is mapped into a set of task structures defined as follows.

Definition 1 *A* **plan**, $\mathcal{P}$, *is composed of a set of* activities, *a set of ordering constraints among activities, and an initial resource allocation* r_0.

The overall plan is generated off-line at the control center by a mixed-initiative planning process that is beyond the scope of this paper. We focus on the reactive, on-board scheduling process only. Resources are represented as vectors of discrete units measuring the availability of each resource. We assume initially that the plan is totally ordered and discuss the generalization of the technique to partially-ordered plans in Section 5.

The rover can perform a certain number of (parameterized) activities each of which has a predefined task structure associated with it. For simplicity of the discussion, we avoid the extra identifier indicating the type of activity when we consider the control of a single activity. Each activity is associated with a progressive processing unit.

Definition 2 *A* **progressive processing unit** *(PRU) is composed of a sequence of processing levels,* $(l_1, l_2, \ldots, l_L)$.

Definition 3 *Each processing level,* l_i, *is composed of a set of* p_i **alternative modules**, $\{m_i^1, m_i^2, \ldots, m_i^{p_i}\}$.

Each module can perform the logical function of level l_i, but it has different computational characteristics defined below.

Definition 4 *The* **module descriptor**, $P_i^j((q', \Delta r)|q)$, *of module* m_i^j *is the probability distribution of output quality and resource consumption for a* given *input quality.*

The module descriptor specifies the probability that module m_i^j consumes Δr resources (a discrete vector specifying the number of units of each resource used by the module) and produces a result of quality q' when the quality of the previously executed module is q. Module descriptors are similar to *conditional performance profiles* of anytime algorithms (Zilberstein and Russell 1996).

When the system completes a task, it receives a reward that depends on the quality of the output.

Definition 5 *Each PRU has a* **reward function**, $U(q)$, *that measures the immediate reward for performing the activity with overall quality* q.

Rewards are cumulative over different activities.

Given a plan $\mathcal{P}$, a library of task structures that specify a PRU for each activity in the plan, the module descriptors of all the components of these PRUs, and corresponding reward functions for each activity, we define the following control problem.

Definition 6 *The* **reactive control problem** *is the problem of selecting a set of alternative modules so as to maximize the expected utility over a complete plan.*

The meta-level control is "reactive" in the sense that we assume that the module selection mechanism is very fast, largely based on off-line analysis of the problem.

Optimal control of a single activity

We begin with the problem of meta-level control of a single progressive processing unit corresponding to a single activity. This problem can be formulated as a Markov decision process (MDP) with states representing the current state of the activity. The state includes the current level of the PRU, the quality produced so far, and the remaining resources. The rewards are defined by the utility of the solution. The possible actions are to *execute* one of the modules of the next processing level. The transition model is defined by the descriptor of the module selected for execution.

Note that in certain situation it might be beneficial to skip the execution of a particular level or the complete activity. To allow that, we introduce a *dummy* module in each level that consumes no resources and produces zero quality. This guarantees that at least one module is executable in each level regardless of the availability of resources.

The rest of this section gives a formal definition of the MDP and the reactive controller produced by solving it.

State transition model

The execution of a single progressive processing unit, u, can be seen as an MDP with a finite set of states $\mathcal{S} = \{[l_i, q, r] | l_i \in u\}$ where $0 \leq i \leq L$ indicates the last executed level, $0 \leq q \leq 1$ is the quality produced by the last executed module, and r is the remaining resources.

When the system is in state $[l_i, q, r]$, one module of the i-th level has been executed. (The first level is $i = 1$; $i = 0$ is used to indicate the fact that no level has been executed.) The states $[l_L, 0, r]$ represent termination with no useful result and remaining resources r.

The initial state of the MDP is $[l_0, 0, r]$, where r is the available resources. The initial state indicates that the system is ready to start executing a module of the first level of the PRU. The terminal states are all the states of the form $[l_L, q, r]$. In every nonterminal state, the possible actions are $\mathbf{E}_{i+1}^j$ (execute the j-th module of the next level). To complete the transition model, the probabilistic outcome of these actions are defined as follows.

To simplify the presentation, we assume that a module is executable only when there are enough resources to cover the worst-case possibility. This can be relaxed if we add a mechanism to abort an action once it requires more resources than available.

The outcome of each action, $\mathbf{E}_{i+1}^j$, is probabilistic. Resource consumption and quality uncertainties define the new state.

$$Pr([l_{i+1}, q', r - \Delta r] \mid [l_i, q, r], \mathbf{E}_{i+1}^j) = P_{i+1}^j((q', \Delta r)|q) \quad (1)$$

Rewards and the value function

Rewards are determined by the given reward function applied to the final outcome. Note that no rewards are associated with intermediate results, although this could be easily incorporated into this model.

We now define a value function (expected reward-to-go) over all states. The value of terminal states is defined as follows.

$$V([l_L, q, r]) = U(q) \quad (2)$$

The value of nonterminal states of the MDP is defined as follows.

$V([l_i, q, r]) =$

$$\max_j \sum_{q', \Delta r} P_{i+1}^j((q', \Delta r)|q)\ V([l_{i+1}, q', r - \Delta r]) \quad (3)$$

This defines a finite-horizon MDP, or equivalently, a state-space search problem that can be represented by a decision tree or AND/OR graph. It can be solved using standard dynamic programming or using a search algorithm such as AO*

Theorem 1 *Given a progressive processing unit u, an initial resource allocation r_0, and a reward function $U(q)$, the optimal policy for the corresponding MDP provides an optimal strategy to control u.*

Proof: Because there is a one-to-one correspondence between the reactive control problem and the MDP (including the fact that the PRU transition model satisfies the Markov assumption), and because of the optimality of the resulting policy, we conclude that it provides an optimal reactive strategy to control the execution of the given progressive processing unit. □

We note that the number of states of the MDP is bounded by the product of the number of levels L, the maximum number of alternative modules per level $\max_i p_i$, the number of discrete quality levels, and the number of possible resource vectors. While resource measures could vary over a large range, the size of the control policy can be reduced by using a coarse unit. Therefore, unit choice introduces a tradeoff between the size of the policy and its effectiveness.

We have implemented the policy construction algorithm for a problem that involves only one resource (time). Figure 2 shows the results we got with a task structure composed of 3 levels, with 5 modules per level (all levels included a dummy 6th module that allows the controller to skip that level). The five unit sizes in this case represent multiples of 1, 10, 20, 40, and 80 of the original quality and time units. The dark bars show the time to construct the policy (logarithmic scale measured in milliseconds). The light bars show the relative reduction in the expected value of the initial state with respect to the optimal value. (The reduction in

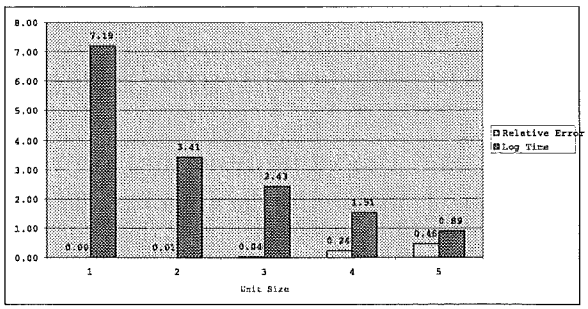

Figure 2: The effect of resource unit size on policy construction time and value

value is due to the fact that the process is modeled using coarse resolution and a compact policy.) It is interesting to note that it takes 15619 seconds to construct a precise policy with no error (left columns) while an approximate policy with a unit size of 20 (middle columns) takes only 0.27 seconds and the error is only 3.6%. These preliminary results are consistent with our intuition that the optimal policy can be approximated with a coarse resource unit and a compact policy.

Optimal Control of multiple PRUs using opportunity cost

Suppose now that we need to schedule the execution of a complete plan that includes $n+1$ activities. One approach is to construct an optimal schedule by generalizing the solution presented in the previous section. That is, one could construct a large MDP for the combined sequential decision problem including the entire set of $n+1$ PRUs. Each state must include an indicator of the activity (or PRU) number, i, leading to a general state represented as $[i, l, q, r]$.

This rather complex MDP is still a finite-horizon MDP with no loops. Moreover, the only possible transitions between different PRUs are from a terminal state of one PRU to an initial state of a succeeding PRU. Therefore, we can solve this MDP by computing an optimal policy for the *last* PRU for any level or resource availability r, then use the value of its initial states to compute an optimal policy for the previous PRU and so on.

Theorem 2 *Given a plan $\mathcal{P}$ represented as a sequence of progressive processing units, and a reward function $U_i(q)$ associated with each PRU, the optimal policy for the corresponding MDP provides an optimal strategy to control $\mathcal{P}$.*

This is an obvious generalization of Theorem 1. The complete proof, by induction on the number of PRUs, is omitted.

We now show how to measure the effect of the remaining n PRUs on the execution of the first one. This effect can be reformulated in a way that preserves optimality while suggesting an efficient approach to meta-level control that does not requires run-time construction of the policy.

Definition 7 *Let $V^*(i,r) = V([i, l_0, 0, r])$ denote the expected value of the optimal policy for the last $n-i+1$ PRUs.*

To compute the optimal policy for the i-th PRU, we can simply use the following modified reward function.

$$U_i'(q,r) = U_i(q) + V^*(i+1,r) \quad (4)$$

In other words, the reward for completing the i-th activity is the sum of the immediate reward and the reward-to-go for the remaining PRUs using the remaining resources. Therefore, the best policy for the first PRU can be calculated if we use the following reward function for final states:

$$U_0'(q,r) = U_0(q) + V^*(1,r) \quad (5)$$

Definition 8 *Let $OC(r, \Delta r) = V^*(1,r) - V^*(1, r-\Delta r)$ be the resource* **opportunity cost function.**

The opportunity cost measures the loss of expected value due to reduction of Δr in resource availability when starting to execute the last n PRUs.

Definition 9 *Let the* **OC-policy** *for the first PRU be the policy computed with the following reward function:*

$$U_0'(q,r) = U_0(q) - OC(r_0, r_0 - r)$$

The OC-policy is the policy computed by deducting from the actual reward for the first task the opportunity cost of the resources it consumed.

Theorem 3 *Controlling the first PRU using the OC-policy is optimal.*

Proof: From the definition of $OC(r, \Delta r)$ we get:

$$V^*(1, r_0 - \Delta r) = V^*(1, r_0) - OC(r_0, \Delta r) \quad (6)$$

To compute the optimal schedule we need to use the reward function defined in Equation 4 that can be rewritten as follows.

$$U_0'(q, r_0 - \Delta r) = U_0(q) + V^*(1, r_0) - OC(r_0, \Delta r) \quad (7)$$

Or, equivalently:

$$U_0'(q,r) = U_0(q) + V^*(1, r_0) - OC(r_0, r_0 - r) \quad (8)$$

But this reward function is the same as the one used to construct the OC-policy, except for the added constant $V^*(1, r_0)$. Because adding a constant to a reward function does not affect the policy, the conditions of Theorem 2 are met and the resulting policy is optimal. □

Theorem 3 suggests an optimal approach to scheduling an arbitrary set of $n+1$ activities by first using an OC-policy for the first PRU that takes into account the resource opportunity cost of the remaining n activities. Then the OC-policy for the second PRU is used taking into account the opportunity cost of the remaining $n-1$ activities and so on. To be able to implement this approach we need to have the control policies readily available. This issue is addressed in the following section.

Current work and open problems

In the previous section, we presented a solution to the control problem of multiple progressive processing units without accounting for its computational complexity. In particular, the opportunity cost must be computed and sometimes revised when the plan is modified during execution. Once the opportunity cost is revised, a new policy for the current PRU must be constructed. In principle, finding the exact opportunity cost requires the construction of an optimal policy for the entire plan. In this section we discuss current work aimed at reducing the complexity and enhancing the applicability of this framework.

Using precompiled control policies In order to implement an effective reactive controller for progressive processing, one should avoid reconstruction of control policies on-board or at the control center. Instead, we propose to:

1. use a fast approximation scheme to estimate the opportunity cost; and
2. use pre-compiled policies for different levels of opportunity cost.

We have examined several approaches to estimating the opportunity cost. Function approximation techniques seem to be suitable for learning the opportunity cost from samples of examples for which we can compute the exact cost off-line. In order to avoid computing a new policy (for a single PRU) each time the opportunity cost is revised, we can divide the space of opportunity cost into a small set of regions representing typical situations. For each region, an optimal policy would be computed off-line and stored in a library. At run-time, the system will first estimate the opportunity cost and then use the appropriate pre-compiled policy from the library. These policies remain valid as long as the overall task structure and the utility function are fixed.

Handling multiple resources As the number of resources grow, the size of each module descriptor and the overall policy also grow. This growth is exponential in the number of resources. Because we anticipate the number of resources in this application to remain small (two or three), the effect on computational complexity of policy construction is limited. Another source of complexity, however, is the approximation of the opportunity cost of multiple resources. In general, the OC function is not additive over resources. However, independence relationships between certain cost functions could simplify the approximation by allowing us to approximate separately each cost function.

More complex task structures The progressive processing task structures we have studied so far are rather limited. They are composed of a sequence of levels each with a set of alternative methods. We are examining several extensions including trees and directed acyclic graphs. Cycles in the task structure could be handled as well (making the control problem an infinite horizon MDP). Cycles could be used to represent multiple attempts to execute actions that fail.

Partially-ordered plans Another generalization of this work is to allow the components of a PRU or the overall plan to be partially ordered. The effect of this generalization on a single PRU is that the state of the MDP must include the "frontier" of the execution subgraph with several different modules being ready for execution. This is much more complex than the single point in sequence of levels we are currently using. This generalization will require some restrictions on how large the state space may grow (for example, by limiting the non-linearity of the plan to just a few choice points). Handling an overall plan that is partially ordered is not difficult, as long as the different activities remain independent.

Dependency among activities Right now we assume that different activities in a plan are independent, except for the fact that they share resources and contribute to the comprehensive value of the plan. Dependency among activities could be represented using additional state variables that capture the sources of dependency. Each action in this model will have possible stochastic effects on some state variables. Activities (or specific methods in the progressive processing task structure) could be conditioned on these state variables. The size of the state space grows exponentially with the number of additional state variables, making it hard to model highly dependent activities.

Conclusion

We present an approach to meta-level control of the activity of planetary rovers by mapping each activity into a progressive processing unit and formulating the control problem as an MDP. It is shown that an optimal policy for a plan composed of a sequence of activities can be constructed by controlling a single PRU at a time, taking into account the opportunity cost of the remaining tasks. To apply this model to control the operation of an autonomous rover, a fast approximation of the opportunity cost is needed. Finally, a highly reactive controller is described that uses a library of pre-compiled control policies to operate in a dynamic environment.

A less complex model of progressive processing that relies on heuristic scheduling has been studied in (Mouaddib and Zilberstein 1997). The task structure, however, is limited to a linear set of levels with one module per level and no quality uncertainty or quality dependency. The heuristic scheduler is fast, but it is not optimal. Heuristic scheduling of computational tasks has also been studied by Garvey and Lesser (1993) for the *design-to-time* problem-solving framework. The latter framework represents explicitly non-local interactions between sub-tasks.

The progressive processing framework relates to a large body of work within the systems community on *imprecise computation* (Liu *et al.* 1991). Each task

in that model is decomposed into a *mandatory* subtask and an *optional* subtask. A variety of scheduling algorithms have been developed for imprecise computation under different assumptions about the optional part. Our model allows for a richer representation of quality and resource uncertainty and quality dependency. Unlike imprecise computation, the schedule constructed by the MDP scheduler is a *conditional schedule*; the selection of modules is conditioned on the *actual* resource consumption and outcome of previous modules.

The application of dynamic programming to control interruptible anytime algorithms has been studied by Hansen and Zilberstein (1996). Optimal monitoring of progressive processing tasks using a corresponding MDP has been studied by Mouaddib and Zilberstein (1998) with respect to a simpler task structure and without the notion of quality uncertainty and quality dependency. The notion of *opportunity cost* is borrowed from economics. It has been used previously in meta-level reasoning by Russell and Wefald (1991). Horvitz (1997) uses a similar notion to develop a model of *continual computation* in which idle time is used to solve anticipated future problems.

The use of pre-compiled control policies to construct a highly reactive real-time system has been studied by several researchers. For example, Greenwald and Dean (1998) show how a real-time avionics control system can use a library of schedules that cover all possible situations. Each schedule is conditioned on the state of the flight operation.

Acknowledgments

We thank Richard Washington and John Bresina for introducing us to the domain of planetary rover control and for numerous helpful discussions. Andy Arnt implemented the policy construction algorithm. This work was supported in part by the National Science Foundation under grants IRI-9624992 and INT-9612092, by the GanymedeII Project of Plan Etat/Nord-Pas-De-Calais, and by IUT de Lens.

References

R.P Bonasso, D. Kortenkamp, D. Miller, and M. Slack. Experience with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical AI*, 1996.

T.T. Blackmon, L. Nguyen, C.F. Neveu, D. Smith, C. Anderson, and V. Gupta. Command generation for planetary rovers using virtual reality. *Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 1999.

J. Bresina, K. Golden, D. Smith, and R. Washington. Increased flexibility and robustness for Mars rovers. *Fifth International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 1999.

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

T. Estlin, A. Gray, T. Mann, G. Rabideau, R. Castano, S. Chien, and E. Mjolsness. An integrated system for multi-rover scientific exploration. *Sixteenth National Conference on Artificial Intelligence*, 541–548, 1999.

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

L. Greenwald and T. Dean. A conditional scheduling approach to designing real-time systems. *AI Planning Systems*, 1229–1234, 1998.

E.A. Hansen and S. Zilberstein. Monitoring the progress of anytime problem-solving. *Thirteenth National Conference on Artificial Intelligence*, 1229–1234, 1996.

E. Horvitz. Reasoning under varying and uncertain resource constraints. *Seventh National Conference on Artificial Intelligence*, 111–116, 1988.

E. Horvitz. Models of continual computation. *Fourteenth National Conference on Artificial Intelligence*, 286–293, 1997.

J. Liu, K. Lin, W. Shih, A. Yu, J. Chung, and W. Zao. Algorithms for scheduling imprecise computations. *IEEE Transactions on Computers*, 24(5):58–68, 1991.

A.I. Mouaddib. *Contribution au raisonnement progressif et temps réel dans un univers multi-agents.* PhD thesis, University of Nancy I, (in French), 1993.

A.I. Mouaddib and S. Zilberstein. Handling duration uncertainty in meta-level control of progressive reasoning. *Fifteenth International Joint Conference on Artificial Intelligence*, 1201–1206, 1997.

A.I. Mouaddib and S. Zilberstein. Optimal scheduling of dynamic progressive processing. *Thirteenth Biennial European Conference on Artificial Intelligence*, 449–503, 1998.

N. Muscettola, P.P. Nayak, B. Pell, and B.C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2), 1998.

D. Musliner, E. Durfee, and K. Shin. Circa: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), 1993.

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

R. Washington, K. Golden, J. Bresina, D.E. Smith, C. Anderson, and T. Smith. Autonomous rovers for Mars exploration. *IEEE Aerospace Conference*, 1999.

S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence* 82(1-2):181–213, 1996.

S. Zilberstein and A.I. Mouaddib. Reactive control of dynamic progressive processing. *Sixteenth International Joint Conference on Artificial Intelligence*, 1268–1273, 1999.