New Directions in Modeling and Control of Progressive Processing

Abdel-Illah Mouaddib[*] Shlomo Zilberstein[†] Victor Danilchenko[†]

Abstract

Progressive processing is an approach to resource-bounded execution of a set of tasks under time pressure. It allows a system to limit the computation time allocated to each task by executing a subset of its components and by producing a sub-optimal result. Progressive processing is a useful model for a variety of real-time tasks such as diagnosis, planning, and intelligent information gathering. This paper describes recent results and new directions aimed at generalizing the applicability of progressive processing by addressing the issues of high *duration uncertainty* and *quality uncertainty* associated with each computational unit. We also examine new ways to model inter-task *quality dependency* and a richer topology of task structures.

1 The Progressive Processing Model

Progressive processing is a resource-bounded reasoning technique that allows a system to satisfy a set of requests under time pressure [8; 9]. The technique is based on structuring each problem-solving component as a hierarchy of levels, each of which contributes to the overall quality of the result. The technique is suitable for a wide range of applications such as hierarchical planning [6] and model-based diagnosis [1]. The ability to trade off computational resources against quality of results is shared by other resource-bounded reasoning techniques such as *flexible computation* [4], *anytime algorithms* [13], *imprecise computation* [7; 5] and *design-to-time* [2]. However, the distinctive hierarchical structure of progressive processing facilitates an efficient management of computational resources.

Progressive processing presents some interesting modeling and control problems. The modeling problem involves the characteristics of the environment and the *progressive processing units* (PRUs). For example, the environment may be dynamic with new PRUs being constantly added. Each PRU may have its own deadline and there may be uncertainty regarding the duration and quality of each problem solving component. The meta-level control problem is the problem of deciding how much computational time should be allocated to each PRU. Initially, we assume that each PRU is an independent problem solving task that needs to be executed. Each PRU has a fixed number of computational components, called *levels*, each of which contributes to the overall quality of the result. When the system cannot execute all the levels, it should select those levels that maximize the overall expected quality.

This paper describes recent results and new directions aimed at generalizing the applicability of progressive processing by addressing the issues of high *duration uncertainty* and *quality uncertainty* associated with each computational unit. We also examine new ways to model inter-task *quality dependency* and a richer topology of task structures. The new directions build on previous work by Mouaddib and Zilberstein [10; 11] on control of progressive processing. In [10], an incremental, heuristic scheduler is described that addresses the static version of the problem (a fixed set of PRUs). In [11], an optimal monitoring policy (or schedule) is computed by solving a corresponding Markov Decision Process (MDP) that takes into account duration uncertainty. A similar approach has been developed by Hansen and Zilberstein [3] for control of interruptible anytime algorithms. The extensions we propose here are based on the MDP controller in [11]. These extensions address quality uncertainty and dependency and a more complex structure of tasks.

Our target application for the extended MDP scheduler is in the domain of intelligent information retrieval. Over the past few years there has been a substantial growth in the number of real-time information servers (databanks) over the internet providing a wide range of scientific, economic, and social services. The response to an information request involves a local search process to find relevant information, filtering the results to adapt them to the user needs, and preparing the final response. In an attempt to provide high-quality information, the information providers may need to allocate a considerable amount of computational resources to each request. The vast majority of today's information providers are using a static strategy in order to prepare the response so that the user receives the same data regardless of the load on the system and the cost of satisfying the request. As a result, some requests must be rejected or ignored when the server is faced with a high load.

[*]CRIL/Université d'Artois, Rue de l'université, S.P. 16, 62307 Lens Cedex France, `mouaddib@cril.univ-artois.fr`

[†]Department of Computer Science, University of Massachusetts, Amherst, MA 01003, U.S.A., `shlomo@cs.umass.edu`

Therefore, the domain of real-time intelligent information retrieval can benefit from a progressive processing approach that addresses duration and quality uncertainty and dynamic environments. Each type of information request can be mapped into a progressive processing unit in such a way that the lowest level of the PRU generates a response of minimal quality and each additional level improves the quality of the result. For example, a request for *publications* in a certain area defined by keywords (e.g., anytime and progressive processing) can be satisfied by a PRU with two levels: the first level (that is mandatory) will search for information that includes only title, authors' names, and link to content, while the second level will retrieve the abstract of each article and the publication in which it appeared. Additional levels can perform intelligent filtering to improve the precision of the result.

The progressive processing approach offers several obvious advantages since it allows the system to trade off computational resources against the quality of the response. When operating under high load, the system can exhibit *robustness* and *fairness*, producing a response to every request with a minimal quality. The system can also maximize the return to the server if quality attached to each level of processing represents monetary rewards. The rest of this section defines the progressive processing problem representation more formally.

The (dynamic) progressive processing task consists of a set $P = \{P_1, \ldots, P_n\}$ of individual problems (information requests) such that:

- P is constructed dynamically: an old problem is removed from the set when a response is sent, and a new problem is added to the set when a new request arrives,
- each problem P_i has a deadline D_i to respect,
- each problem P_i could be solved at varying levels through a progressive processing unit u based on a hierarchy of processing levels $\{l_u^1, l_u^2, l_u^3, \ldots, l_u^k\}$, and
- each processing level L is characterized by the tuple $(C(L),\ q(L))$. $C(L)$ is the model of duration uncertainty. This model, in the MDP controller, consists of a discrete distribution of the duration of processing [11]. This distribution is represented by a set of tuples $\{(\Delta_L^1, p_1), (\Delta_L^2, p_2), \ldots, (\Delta_L^i, p_i)\}$, where $(\Delta_L^k,\ p_k)$ means the level L takes Δ_L^k units time with the probability p_k. While this model represents the most likely duration c^L and the standard deviation v^L in the incremental scheduling [10]. $q(L)$ represents the quality improvement of the overall response when the level is executed.

The rest of this paper is organized as follows. Sections 2 and 3 describe the two existing approaches to modeling and control of progressive processing. Section 4 describes the set of extensions that are the focus of current research efforts. We conclude with a description of the anticipated benefits of the extended MDP controller.

2 Heuristic Scheduling

This section presents a heuristic approach to scheduling a given set of progressive processing units. Consider a set P of n problems. Let $\mathcal{U}$ be a set of n PRUs $\{u_1,\ u_2, \ldots,\ u_n\}$ that solve the n problems. The PRUs are sorted by the deadlines (*earliest-deadline-first*) of the corresponding problems. The scheduling problem is to find for each PRU the optimal subset of its levels such that all deadlines are respected. This decision is made under uncertainty regarding the execution time of each processing level and regarding future changes of the set $\mathcal{U}$. Each processing level l_i^j is assigned its most likely duration c_i^j computed from gathered data of previous executions and a utility U_i^j that is defined as follows: $U_i^j = V(q_i^j) - Cost(c_i^j)$, where q_i^j is the quality of the solution at the level l_i^j.

2.1 Framework

We represent the ordered set $\mathcal{U}$ by a graph $\mathcal{G}$ (Figure 1) where each node is a level l_j^i. The successors of each level l_j^i are l_j^{i+1} and l_{j+1}^1 when they exist.

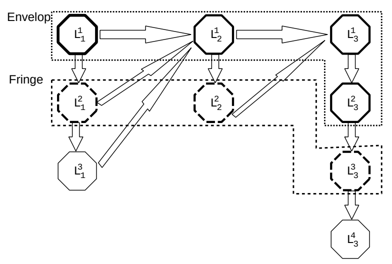

Figure 1: Example showing the framework

This structure allows to perform utility-based scheduling by incrementally inserting nodes (processing levels) into the current schedule. We now define several concepts necessary to describe the scheduling process.

• The *envelop*, $\mathcal{E}$, is a subgraph of $\mathcal{G}$ containing all the levels scheduled for execution (nodes drawn with bold lines in Figure 1). This is similar to the notion of a *closed list* in search algorithms. The envelop is a dynamic structure that can be revised during execution.

• The *fringe*, $\mathcal{F}$, is the set of direct successors to the nodes in $\mathcal{E}$ (nodes drawn with dotted lines in Figure 1). This is similar to the notion of an *open list* in search that contains nodes on the frontier of the search tree that are candidates for expansion.

• The *frozen space*, $\mathcal{Z}$, is a set of branches pruned during the scheduling cycle. It contains the levels that violate some deadlines during the scheduling phase. We save these levels because during the revision of the schedule, these levels may be added to the search space and eventually they may be added to the schedule.

2.2 Incremental scheduling

The main design goal of the incremental scheduler is the ability to return an approximate schedule for all the

PRUs at any time and to revise this schedule when a deviation from the predetermined schedule is detected at run-time. These two properties allow the system to deal with domains characterized with rapid change and a high level of uncertainty.

Conceptual description

With the PRU structure and formal framework described above, scheduling could be seen as the problem of finding an optimal path that visits the maximum number of levels in the graph $\mathcal{G}$ without violating any deadline. There are different possible paths with different qualities. Our strategy consists of finding a minimal schedule that includes all the PRUs and refining it progressively. The scheduler starts its processing by building the schedule with the lowest quality (the minimal envelop) and refining it by inserting additional nodes into the graph as long as all the deadlines are respected. The incremental processing of the scheduler is guided by the progressive structure of the PRUs and by the (easy to construct) *fringe*.

Utility-based scheduling algorithm

The construction of the schedule is based on a series of cycles of expansion of the current envelop. This expansion consists of inserting levels of the fringe into the envelop. This process is repeated until a maximal envelop is reached (i.e., any further expansion leads to a violation of a deadline) or until an external even causes the interruption of scheduling. At each cycle, a schedule is available and its quality is improved from one cycle to another. The algorithm consists of the following steps:

• **Initialization step**:

$$\mathcal{E} = \mathcal{Z} = \emptyset \; and \; \mathcal{F} = \{L_u^1 \in \mathcal{G} \mid \forall\, u \in \mathcal{U}\} \qquad (1)$$

• **Expansion step**:

This step consists of extending $\mathcal{E}$ to all the levels in $\mathcal{F}$:

$$\mathcal{E} = \mathcal{E} \biguplus \mathcal{F} \; and \; \mathcal{F} = \emptyset \qquad (2)$$

The operator $\uplus$ allows to insert levels while respecting the structure of the graph $\mathcal{G}$ and thus at each cycle, $\mathcal{E}$ is a subgraph of $\mathcal{G}$ (as shown in Figure 1).

• **Test of feasibility step**:

The schedule fails when one deadline is violated:

$$\exists\, \gamma \in \mathcal{U}: \sum_{\{\delta \in \mathcal{U},\, D_\delta \leq D_\gamma\}} \sum_{i=1}^{i=d_\delta} c_\delta^i > D_\gamma, \qquad (3)$$

$$\text{where } \{L_\delta^1, \ldots, L_\delta^{d_\delta}\} \subset \mathcal{E}$$

If the schedule fails, go to the *approximation step*, otherwise go to the *new cycle step.*

• **Approximation step**:

The level with the lowest utility, noted L_{min}, when is discarded when the schedule fails. The level L_{min} is selected among the levels L_γ^k inserted by the last expansion cycle.

$$L_{min} = arg(MIN_{L_\gamma^k}(U_{L_\gamma^k})) \qquad (4)$$

The branch containing the successors nodes is pruned from $\mathcal{G}$ and is placed in $\mathcal{Z}$.

Afterwards, go to the *test of feasibility step.*

• **New fringe step**:

For each processing level L_α^i inserted in $\mathcal{E}$ by the last expansion cycle, we insert into the fringe its successor (if it exists). Formally:

$$\mathcal{F} = \{L_\alpha^{i+1} \; if\ exists \mid L_\alpha^i \in \mathcal{E}\} \qquad (5)$$

If the fringe is not empty go to the *expansion step*, otherwise stop the algorithm and return the envelop $\mathcal{E}$.

2.3 Performance and limitations

We have implemented and tested the incremental scheduler [10]. This approach seems to be more suitable than design-to-time [2] and imprecise computation [7] in applications characterized by duration uncertainty. However, this approach has two important limitations. First, the utility-directed greedy scheduling algorithm is a local optimization approach to a scheduling problem that is hard to optimize globally. In addition, this approach is suitable for applications characterized by duration uncertainty but its effectiveness diminishes under a high level of uncertainty.

3 Optimal Monitoring Policies

The standard problem of scheduling and monitoring progressive processing can be viewed as a control problem of a Markov Decision Process (MDP) [11]. The states of the MDP represent the current state of the computation in terms of the unit/level being executed and the time. The rewards associate with a state are simply the rewards for executing each level or a unit. The two possible actions are to execute the next level of the current unit or to move to the next processing unit. The transition model is defined by the duration uncertainty associated with the level selected for execution. This section gives a formal definition of the resulting MDP, describes an algorithm for constructing an optimal policy for action selection, and summaries the performance and limitations of this approach.

3.1 State representation

Let $\mathcal{U}$ be a set of PRUs $\{u_1, u_2, \ldots u_n\}$ and l_i^j is the j-th processing level of unit u_i. Each unit u_i in the set $\mathcal{U}$ has a deadline D_i for finishing its processing. The units in $\mathcal{U}$ are sorted by their deadlines. Δ_i^j is a random variable representing the duration of processing level l_i^j. We model the execution of the entire set of units as a stochastic automaton with a finite set of world states $\mathcal{S} = \{[l_i^j, t] | u_i \in \mathcal{U}\}$ where $0 \leq j \leq MaxLevel(u_i)$ and

$t \geq 0$ represents the remaining time to the deadline of u_i. When the system is in state $[l_i^j, t]$, the j-th level of unit u_i has been executed (since the first level is 1, $j = 0$ is used to indicate the fact that no level has been executed).

3.2 Transition model

The initial state of the MDP is $[l_1^0, D_1 - T]$ where T is the current time. This state indicates that the system is ready to start executing the first level of the first unit. The terminal states are all the states of the form: $[l_n^j, 0]$ or $[l_n^m, t]$ where m is the last level of the last unit n. The former set includes states that reach the deadline of the last unit and the latter set includes states that complete the execution of the last unit (possibly before the deadline).

In every nonterminal states there are two possible actions: **E** (execute) and **M** (move). The **E** action continues the execution of the next level of the current PRU and the **M** action moves to the initial state of the next PRU. Note that by limiting the actions to this set we exclude the possibility of executing levels of previous PRUs, even if the deadlines allow such actions. In other words, we make the *monotonicity* assumption that execution is performed PRU by PRU in the order of their deadlines. This assumption is reasonable for applications characterized by high time pressure and rapid change such as the information retrieval problem. In such applications, it is desirable to report the best result generated for a particular request as soon as the system completes its work on the request.

The transition model is a function that maps each element of $\mathcal{S} \times \{\mathbf{E}, \mathbf{M}\}$ into a discrete probability distribution over $\mathcal{S}$. Equations 6-8 define the transition probabilities for a given nonterminal state $[l_i^j, t]$:

The **M** action is deterministic. It moves the MDP to the next processing unit and updates the remaining time to the deadline of the new unit.

$$Pr([l_{i+1}^0, D_{i+1} - D_i + t] \mid [l_i^j, t], \mathbf{M}) = 1 \tag{6}$$

The **E** action is probabilistic. Duration uncertainty defines the new state. Equation 2 determines the transitions following successful execution and Equation 3 determines the transition to the next PRU when the deadline of the current PRU is reached.

$$Pr([l_i^{j+1}, t - \delta] \mid [l_i^j, t], \mathbf{E}) = Pr(\Delta_i^{j+1} = \delta) \quad when\ \delta \leq t \tag{7}$$

$$Pr([l_{i+1}^0, D_{i+1} - D_i] \mid [l_i^j, t], \mathbf{E}) = Pr(\Delta_i^{j+1} > t) \tag{8}$$

3.3 Rewards and the value function

Rewards are associated with each state based on the quality gain by executing the most recent level. Recall that each level of a unit has a predetermined quality. Therefore,

$$R([l_i^0, t]) = 0 \tag{9}$$

$$R([l_i^j, t]) = q(l_i^j) \tag{10}$$

Now, we can define the value function (expected reward-to-go) for nonterminal states of the MDP as follows [12]:

$$V(s) = R(s) + \max_a P(s'|s,a)V(s') \tag{11}$$

Using our former notation we get $V([l_i^j, t]) =$

$$R([l_i^j, t]) + \max \begin{cases} V([l_{i+1}^0, D_{i+1} - D_i + t] \\ Pr(\Delta_i^{j+1} > t)V([l_{i+1}^0, D_{i+1} - D_i]) + \\ \sum_{\delta \leq t} Pr(\Delta_i^{j+1} = \delta)V([l_i^{j+1}, t - \delta]) \end{cases} \tag{12}$$

The top expression is the value of a move action and the bottom line is the expected value of an execute action. Note that in states of the form $[l_n^j, t]$ it is not possible to execute a move action to the next unit and hence their value function is simply the result of attempting to execute the next level.

Finally, we need to define the value function for terminal states:

$$V([l_n^m, t]) = R([l_n^m, t]) \tag{13}$$

and

$$V([l_n^j, 0]) = R([l_n^j, 0]) \tag{14}$$

3.4 Optimal schedule

The above MDP is a case of a finite-horizon MDP with no loops. This is due to the fact that every transition moves "forward" in the state space by always incrementing the unit/level number. This class of MDPs can be solved easily for relatively large state spaces because the value function can be calculated in one sweep of the state space (backwards, starting with terminal states). In addition, substantial computational savings result from the fact that each processing unit has its own deadline and because many states of the MDP are not reachable by an optimal policy.

Theorem 1 *Given a monotonic progressive processing problem P, the optimal policy for the corresponding MDP is an optimal schedule for P.*

We have implemented a recursive algorithm that computes the value function and the optimal policy. Figure 2 shows a simple example of a set of two PRUs and the resulting policy. The states of the policy are denoted by circles on grids (one grid per PRU) with horizontal axis showing the remaining time to the deadline of the

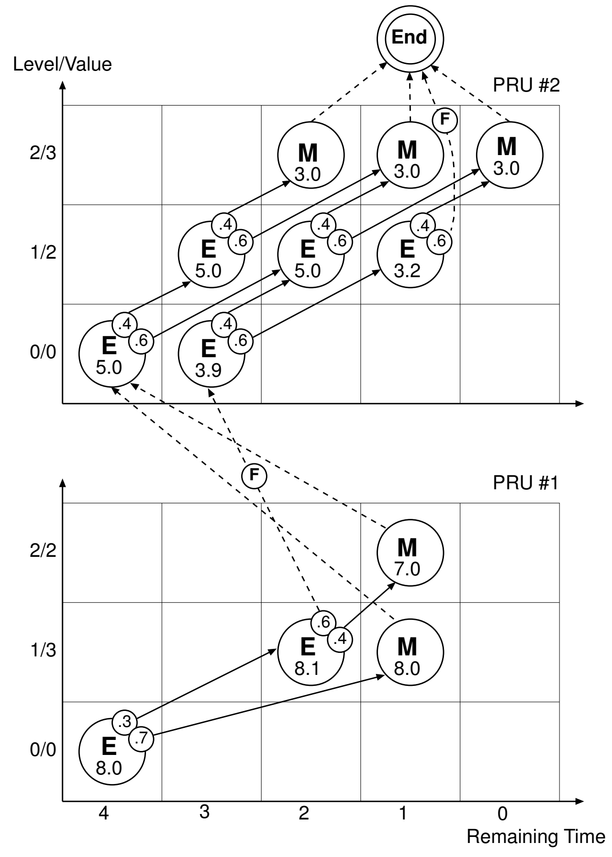

Figure 2: An optimal monitoring policy with 2 PRUs.

PRU and vertical axis showing the level and its value. Each state includes the best action (**E** or **M**) and the expected utility (reward-to-go). Outgoing arrows show the transitions with small circles showing the probability of each transition. The duration is implicit in the graph (by counting the number of time steps for each transition). The dashed lines indicate termination of execution of a PRU. Transitions marked with an F represent failure of an execute action (e.g. duration exceeds the deadline) in which case execution of the level is aborted and control is moved to the next PRU. The initial state is the bottom left state with an expected utility of 8.0. Note that the utility of each state is calculated using Equation 12.

3.5 Performance and limitations

We have implemented and tested the MDP approach to constructing optimal monitoring policies for progressive processing tasks [11]. Not surprisingly, this approach offers a major performance improvement over the heuristic scheduling approach. We also found that constructing the schedule is feasible for large task structures. As a result, we adopt this framework as a foundation for the future directions outlined in the next section. However, the approach suffers from several disadvantages in its current form: (1) The output quality of each processing unit is static and is defined as part of the task structure; (2) The task structure itself is linear; and (3) there is no dependency of output quality of a level of a PRU on previous levels of the same PRU or on other PRUs. The next section describes extensions of the progressive processing model that overcome these limitations and how an optimal monitoring policy can be constructed for the new model.

4 New Directions

We are currently extending the MDP approach to address a more general type of progressive processing task structures. This section describes the key issues addressed by the new model and our approach to handling them.

4.1 Handling quality uncertainty

The standard model of progressive processing assumes that the output quality of each processing unit is static and is defined as part of the task structure. This enabled us to have a simple reward structure by which executing a processing unit results in a fixed, predefined increase in the overall utility. A more flexible model must address *quality uncertainty* and allow us to represent a possible distribution of output quality for each processing unit.

This extension can be handled by replacing the current set of state transitions that include only duration uncertainty with a set of transitions that include both duration and quality uncertainty. In order to be able to associate a fixed reward with each MDP state, the state must also include the *actual* quality produced by the *most recent* level. The new state representation is: $s = (l_j^i, q_j^i, t)$, where q_j^i is the output quality of the most recent level. This is a relatively simple modification which will increase the size of the MDP by a constant factor.

4.2 Handling quality dependency

The standard model assumes that the output quality of each level is independent of previously executed levels. There are two useful ways to relax this assumption. One generalization allows the output quality of each level to depend on the output quality of the previous level of the same PRU. If we adopt the enhanced state representation mentioned above, we will have the output quality of the most recently executed level readily available. All we need to define the new transitions is the conditional probability distribution of output quality. This information can be part of the new task structure definition. While this extension models only a simple form of quality dependency, it will allow us to represent useful dependencies that arise in the information retrieval domain. For example, it will allow us to model the fact that when an initial search for relevant publications produces lower quality (less relevant publications) it is more likely that the outcome of a complex filtering applied to the results will be of lower quality.

Another, more general, form of quality dependency would allow the output quality of each level to depend on the output qualities of a set of other levels (considered its parents) that may belong to the same PRU or different PRUs. If we limit inter-task dependency to the *first* level of each PRU being dependent on the *last* level of the previous PRU, then the solution proposed above is sufficient. However, in general, this extension is substantially more complicated to implement both in terms of the specifications of the task structure and in terms of the size and complexity of the resulting MDP. It also implies a deviation from the linear hierarchical structure of the standard model. Additional task structure modifications are discussed below.

4.3 Enhanced task structures

The standard model requires for all tasks to be sequential and all PRUs to have a linear hierarchy of levels. This simple task structure can be enhanced in several useful ways. First, each PRU can be generalized to include a set of levels and precedence constraints imposed on the levels. Each level can increase the overall quality as long as it is executed in an order that satisfies those constraints. A similar assumption can be made about the overall set of tasks to be executed. We are currently studying a variety of task structures that are both useful (in terms of modeling practical applications) and efficient (in terms of our ability to compute the optimal monitoring policy).

Another aspect of the task structure that can be generalized is the form of time-dependent utility function that we use. The standard model impose strict deadlines on each task and assumes that the comprehensive utility is the sum of qualities produced by all executed levels. We are currently examining more general time-dependent utility functions that would allow us to model situation in which no strict deadlines are imposed on each task and the overall utility is time-dependent.

5 Conclusion

We have examined existing work and future directions in modeling and control of progressive processing. Recent results show that heuristic scheduling can be performed efficiently, but is produces sub-optimal schedules and cannot handle well high levels of duration uncertainty. An alternative MDP controller provides an optimal schedule at a reasonable computational cost. The latter technique offers a major improvement under duration uncertainty and we adopt it as the foundation of current work. To generalize the applicability of progressive processing, we extend the current MDP controller to deal with, in addition to duration uncertainty, quality uncertainty and inter-task quality dependency. These enhancements of the basic model appear to require modest modification to the problem representation that will only increase its complexity by a constant factor. Finally, we address the problem of extending the topology of tasks beyond the current linear hierarchy. The latter extension is a more radical change that remains the focus of future work.

Acknowledgments

Support for this work was provided in part by the National Science Foundation under grants IRI-9634938, IRI-9624992, and INT-9612092, and by the GanymedeII Project of Plan Etat/Nord-Pas-De-Calais, and by IUT de Lens.

References

[1] D. Ash, G. Gold, A. Siever, and B. Hayes-Roth. Guaranteeing real-time response with limited-resources. *Artificial Intelligence in Medicine*, 5(1):49–66, 1993.

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

[3] Hansen and Zilberstein. Monitoring the progress of anytime problem-solving. *AAAI-96*, pages 1229–1234, 1996.

[4] E. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Workshop UAI-87*, pages 429–444, 1987.

[5] D. Hull, W. Feng, and J. Liu. Operating system support for imprecise computation. In *AAAI Fall Symposium on Flexible Computation*, pages 96–99, 1996.

[6] C. Knoblock. Autmatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

[7] J. Liu, K. Lin, W. Shih, A. Yu, J. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, May 1991.

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

[9] A.-I. Mouaddib and S. Zilberstein. Knowledge-based anytime computation. In *IJCAI-95*, pages 775–781, 1995.

[10] A.-I. Mouaddib and S. Zilberstein. Handling duration uncertainty in meta-level control of progressive processing. In *IJCAI-97*, pages 1201–1206, 1997.

[11] A.-I. Mouaddib and S. Zilberstein. Optimal scheduling for dynamic progressive processing. In *ECCAI-98*, pages 499–503, 1998.

[12] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press/Bradford Books, Cambridge, MA, 1998.

[13] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.