Optimal Scheduling of Dynamic Progressive Processing

Abdel-Illah Mouaddib[1] and **Shlomo Zilberstein**[2]

Abstract. Progressive processing allows a system to satisfy a set of requests under time pressure by limiting the amount of processing allocated to each task based on a predefined hierarchical task structure. It is a useful model for a variety of real-time AI tasks such as diagnosis and planning in which it is necessary to trade-off computational resources for quality of results. This paper addresses progressive processing of information retrieval requests that are characterized by high *duration uncertainty* associated with each computational unit and *dynamic operation* allowing new requests to be added at run-time. We introduce a new approach to scheduling the processing units by constructing and solving a particular Markov decision problem. The resulting policy is an optimal schedule for the progressive processing problem. Finally, we evaluate the technique and show that it offers a significant improvement over existing heuristic scheduling techniques.

1 Introduction

Progressive processing is a resource-bounded reasoning technique that allows a system to satisfy a set of requests under time pressure [9, 10]. The technique is based on structuring each problem-solving component as a hierarchy of levels, each of which contributes the the overall quality of the result. The technique is suitable for a wide range of applications such as hierarchical planning [7] 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* [5], *anytime algorithms* [13], *imprecise computation* [8, 6] and *design-to-time* [3]. However, the distinctive hierarchical structure of progressive processing facilitates an efficient management of computational resources.

This paper presents a novel approach to meta-level control of progressive processing. The meta-level control problem is the problem of deciding how much computational time should be allocated to each *progressive processing unit* (PRU). 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. The complexity of the scheduling problem is related to three major aspects of the problem: (1) the environment is dynamic with new PRUs being constantly added, (2) each PRU has its own deadline, and (3) there is uncertainty regarding the duration of each problem solving component (level). When the system cannot provide optimal responses to all the PRUs, the task of the scheduler is to maximize the overall quality of responses under time constraints.

In previous work, Mouaddib and Zilberstein [11] presented an incremental scheduler that addresses the static version of the problem (a fixed set of PRUs). In this paper we present a solution that can handle a dynamic environment and provides better solutions (for both the static and dynamic cases). The solution is based on formulating the scheduling problem as a Markov Decision Process (MDP) and finding an optimal policy (or schedule). A similar approach has been developed by Hansen and Zilberstein [4] for control of interruptible anytime algorithms.

We demonstrate the applicability of the MDP scheduler and evaluate its characteristics 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. A progressive processing approach to the problem leads to substantial performance benefits.

The rest of this paper describes our solution in detail and evaluates the implementation of the scheduler. Section 2 describes the application and how a problem instance is mapped into a progressive processing unit. Section 3 shows how a progressive processing problem can be mapped into a corresponding Markov decision problem and solved using an efficient policy construction algorithm. Section 4 describes the model of execution we used and extends the MDP scheduling approach to the case of a dynamic environment. Section 5 illustrates and evaluates the implementation of the scheduler. We conclude with a summary of the benefits of this approach and further work.

2 Progressive processing of information requests

Many real-time AI tasks can benefit from a progressive processing approach that addresses duration uncertainty and dynamic environments. Specifically, in the domain of real-time intelligent information retrieval, 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 sec-

[1] CRIL/Université d'Artois, Rue de l'université, S.P. 16, 62307 Lens Cedex France, mouaddib@cril.univ-artois.fr
[2] Department of Computer Science, University of Massachusetts, Amherst, MA 01003, U.S.A., shlomo@cs.umass.edu

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

ond level will retrieve the abstract of each article and the publication in which it appeared and perform more intelligent filtering.

Quality improvement may be along several different dimensions: the degree of relevance (filtering information that is not likely to be relevant), the level of detail (adding more information to relevant data already included in the response), and representation of the result (e.g., as a graph rather than a table). Each level of processing can be assigned a quality that is defined either by some subjective estimate of its contribution to the response or by a monetary charge that the user has to pay for quality improvement. In addition, each request will have its own deadline that is defined either by a fixed allowable processing time associated with the request or by the actual deadline imposed by the consumer of the information. The latter case would allow different users to bid for information offering to pay a certain amount of money that depends on both the quality of the result and meeting the deadline.

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 a discrete distribution of the duration of processing. 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 (Δ_L^k, p_k) means the level L takes Δ_L^k units time with the probability p_k. $q(L)$ represents the quality improvement of the overall response when the level is executed.

Given P, the problem is how to construct a schedule of the PRUs that maximizes the comprehensive utility of the system and how to revise that schedule when new problems are added. The following two sections answer these questions.

3 Constructing an optimal schedule

The problem of scheduling and monitoring progressive processing can be viewed as a control problem of a Markov Decision Process (MDP). 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 and describes an algorithm for constructing an optimal policy for action selection.

3.1 State representation

Let $\mathcal{U}$ be a set of units $\{u_1, u_2, \ldots u_n\}$ 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 state 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 mapping from $\mathcal{S} \times \{\mathbf{E}, \mathbf{M}\}$ to a discrete probability distribution over $\mathcal{S}$. Equations 1-3 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.

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

The **E** action is probabilistic. The actual duration 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.

$$Pr([l_i^{j+1}, t - \delta] \mid [l_i^j, t], \mathbf{E}) = Pr(\Delta_i^{j+1} = \delta) \quad \text{if } \delta \leq t \tag{2}$$

$$Pr([l_{i+1}^0, D_{i+1} - D_i] \mid [l_i^j, t], \mathbf{E}) = Pr(\Delta_i^{j+1} > t) \tag{3}$$

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,

$$R([l_i^0, t]) = 0 \tag{4}$$

$$R([l_i^j, t]) = q(l_i^j) \quad (5)$$

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

$$V(s) = R(s) + \max_a P(s'|s,a)V(s') \quad (6)$$

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

$$R([l_i^j, t]) + \max \begin{cases} V([l_{i+1}^0, D_{i+1} - D_i + t]) \\ Pr(\Delta_i^{j+1} > t)V([l_{i+1}^0, D_{i+1} - D_i]) + \\ \sum_{\delta \le t} Pr(\Delta_i^{j+1} = \delta)V([l_i^{j+1}, t - \delta]) \end{cases} \quad (7)$$

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:

$$V([l_n^m, t]) = R([l_n^m, t]) \quad (8)$$

and

$$V([l_n^j, 0]) = R([l_n^j, 0]) \quad (9)$$

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. By solving the MDP we get the following result.

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

Proof: Monotonicity of the progressive reasoning problem (see Section 3.2) limits the space of possible schedules to exactly the space of transitions of the corresponding MDP. The expected value of a given schedule is the same as the sum of the rewards over the states of the MDP. Therefore, the optimal policy represents an optimal schedule for P.

We have implemented a recursive algorithm that computes the value function and the optimal policy. Figure 1 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 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

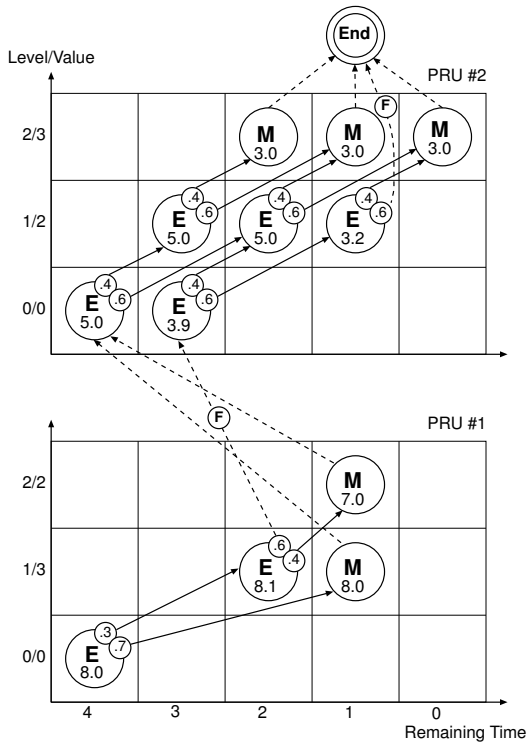

Figure 1. An optimal monitoring policy with 2 PRUs.

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 7.

4 Execution and monitoring of dynamic policies

In the previous section we presented an optimal solution to the control problem of progressive processing. However, our solution did not address two fundamental issues. First, our system must operate in a dynamic environment with new requests for information constantly arriving. As a result, the existing policy needs to be revised to incorporate the new requests. Second, the time needed to construct the policy is short but not negligible. To resolve the latter problem, we assume that policy construction is done in parallel to policy execution using a dedicated processor. The rest of this section presents our solution to the former problem of continuous operation in a dynamic environment.

4.1 Policy revision requests

In order to handle a dynamic environment we modified the policy construction algorithm so that it can handle revision requests. Each revision request includes:

- T_0 the earliest start time of the revised policy.
- T_1 the latest start time of the policy.
- A list of new PRUs to be added to the policy.

T_0 and T_1 reflect the uncertainty regarding the time at which the controller will start using the new policy. In general, the difference between T_0 (earliest start time) and the deadline of the last PRU (latest completion time) is limited by a system parameter which is the

largest allowable scheduling horizon. The time and space complexity of the policy revision algorithm grows linearly with this constant.

4.2 Generating a new policy

Once a revision request is generated, the policy revision algorithm sorts the new and existing PRUs by deadline and recomputes the policy (backwards). If there are n new PRUs and the n-th PRU (with the latest deadline) is inserted at position i, then it is necessary to recompute the policy for PRUs $1 \ldots i$ only. This observation can yield substantial computational savings over a complete reconstruction of the policy for the current set of PRUs. However, our first implementation simply reconstructs the policy after each revision request, since the overall computation time of a new policy is sufficiently small.

4.3 Execution model

The execution model defines the interaction between the two parallel processes of policy construction and control of the progressive processing units. In particular, the execution model determines the answers to the following questions:

- At what point along the execution of the current policy a request for a revised policy will be issued?
- What will be the earliest start time and the latest start time of the request?
- How will execution be controlled during the construction of the new policy?
- How will execution be altered once the new policy is available?

Progressive processing treats each level of a PRU as an atomic unit of execution. Therefore, in our model of execution the above decisions are made only between executing individual levels. If new requests for information arrive, a request to revise the policy is issued as soon as the current level terminates. We assume that the revised policy will be ready after the execution of the next level based on the current policy. At that time, the monitor will simply continue to select levels based on the new policy. To guarantee consistency (i.e., that the monitor will be able to find the continuation state in the new policy), we include the currently executing PRU in the revised policy. All the terminated PRUs are deleted and the new requests are added.

Based on the above execution model, the earliest start time and the latest start time of the request are simply the actual start time of the current PRU (which is known). Once a revision request is issued, the monitor makes one last decision based on the existing policy. When the execution of the selected level terminates, the monitor continues to make decisions based on the revised policy (and possibly issues a new revision request). Note that it is possible for the monitor to observe a state (l_i^j, t) that is reachable by the old policy but is not reachable by the new policy. In such case, the monitor selects the move action and continues with the next PRU following the new policy.

5 Experimental evaluation

We have implemented the execution model described above and the policy construction and revision algorithms. This section illustrates the operation of the resulting system and examines two fundamental questions. The first goal is to compare the performance of our approach to a baseline approach similar to the scheduling of imprecise computation [8]. This approach is based on a strategy that assigns each PRU a new deadline that allows the first level of all the remaining PRUs (the PRUs with the greater deadlines) to be executed based on the worst-case duration. This strategy allows to insert the new PRUs with all their levels and it discards levels with lowest values when the schedule becomes infeasible. The baseline approach constructs a pessimistic but safe schedule using the worst-case duration. The resulting schedule is not optimal.

The second goal of the experimental evaluation is to assess the benefit of our approach in domain characterized by a high-level of duration uncertainty and rapid change such as intelligent information retrieval.

5.1 Experimental design

The information retrieval requests are specified in a rich PRU *language* allowing the system to create the necessary processing units for this application. For example, when a request for *publications* is received, a PRU, named *Publication-PRU*, is created and instantiated by the data of the request (e.g. area). We have collected experimental data on the performance of both our approach and the baseline approach. The quality of result for each problem instance is the *sum* of the qualities of all the levels that were executed.

We collected data by generating random problem instances while varying two important parameters. The first parameter is the *number of the inserted new PRUs* over a short time segment. This number reflects the degree to which an approach is suitable for handling dynamic PRUs. The second parameter is the *degree of duration uncertainty* measured by the *standard deviation*. This parameter allows us to assess the relevance of our approach to applications characterized by a high level of uncertainty such as information retrieval.

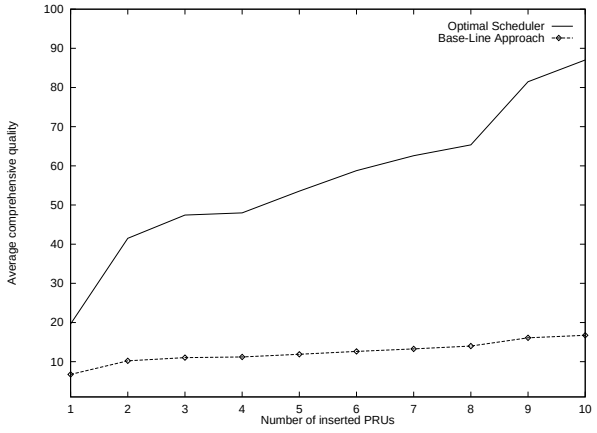

Figure 2. Comprehensive quality with dynamic PRUs

5.2 Handling dynamic PRUs

This experiment compares the comprehensive value of our approach and the baseline approach. We measure the value as a function of the number of inserted new PRUs. Problem instances (10 instances) were generated with *one* PRU in the current policy. For each problem instance, we developed 10 cases (modifying the probability distribution of durations) and we measured the average over these 10 cases. Figure 2 shows the difference between the values of our approach and the baseline approach over the number of inserted PRUs.

The figure confirms the fact that our approach leads to a substantial quality gain over the baseline heuristic approach. The main reason for this is that the policy that we construct covers all possible runtime execution paths including unlikely situations (short durations) that allow the system to execute additional processing levels. This strategy leads to a substantial quality gain not only over the baseline approach but also over all similar scheduling approaches that use a *single* duration such as the average duration [3], the most likely duration [11] or the worst-case duration [2]. Furthermore, the baseline approach is based on a pessimistic strategy that discards all the levels that *may* violate the deadline while our approach takes "risks" when they are justified in terms of expected quality. This explains the substantially slower growth of the comprehensive quality of the baseline approach.

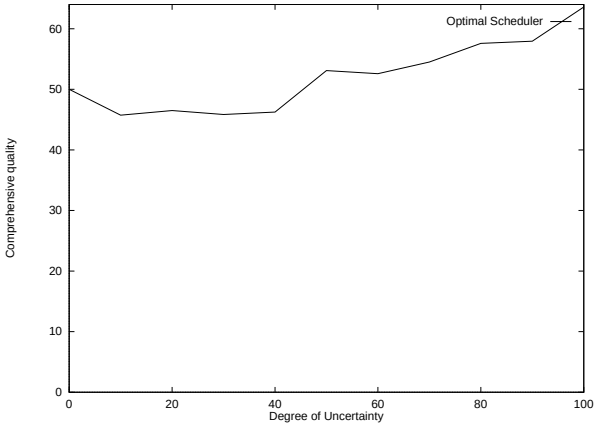

Figure 3. Comprehensive quality for different degrees of uncertainty

5.3 Handling duration uncertainty

This experiment shows the value returned by our approach for different degrees of uncertainty. The experiment assesses the suitability of our approach to environments with high level of duration uncertainty. Problem instances were generated with 10 PRUs and a variation of duration uncertainty from 0% to 100%. Figure 3 shows that the comprehensive value (quality) is stable. Interestingly, after a short decline of expected quality for low variance, quality continues to grow with duration uncertainty. The intuitive explanation is that with high uncertainty there is a chance for substantial time savings and our reactive approach takes advantage of that. Of course, there is also a chance that levels will take more time, but then our approach will skip the *least* valuable units and therefore the net effect on expected value is positive. This observation makes our approach particularly advantageous in situation with high duration uncertainty.

6 Conclusion

This paper presents a new approach to scheduling progressive processing units in domains characterized by real-time, dynamic operation and by duration uncertainty such as intelligent information retrieval. Our approach is based on formulating the scheduling problem as a Markov decision problem and finding an optimal policy. This approach is a major improvement over the incremental scheduler (a local optimization approach) presented in [11]. The policy revision algorithm allows us to apply this approach to dynamic environments that require response to new as well as existing problem-solving requests. Experimental evaluation shows that for the progressive processing units that we considered, the optimal scheduling approach has significant advantages over a heuristic baseline approach. Future directions of this work focus on a richer transition model that captures quality dependency (on the outcome of previous levels) and quality uncertainty in addition to duration uncertainty. Another goal of future work is to apply the technique to control a more complex information retrieval system.

ACKNOWLEDGEMENTS

We thank Victor Danilchenko for implementing the policy construction algorithm. 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] W. Feng and J.W.S Liu, 'An extended imprecise computation model for time-constrained speech processing and generation', in *IEEE Workshop on Real-Time Applications*, pp. 76–80, (1993).

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

[4] Hansen and Zilberstein, 'Monitoring the progress of anytime problem-solving', *AAAI-96*, 1229–1234, (1996).

[5] E.J. Horvitz, 'Reasoning about beliefs and actions under computational resource constraints', in *Workshop UAI-87*, (1987).

[6] D. Hull, W.C. Feng, and J.W.S Liu, 'Operating system support for imprecise computation', in *AAAI Fall Symposium on Flexible Computation*, pp. 96–99, (1996).

[7] C.A. Knoblock, 'Autmatically generating abstractions for planning', *Artificial Intelligence*, **68**, 243–302, (1994).

[8] 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).

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

[10] A.-I Mouaddib and S. Zilberstein, 'Knowledge-based anytime computation', in *IJCAI-95*, pp. 775–781, (1995).

[11] A.-I Mouaddib and S. Zilberstein, 'Handling duration uncertainty in meta-level control of progressive reasoning', in *IJCAI-97*, pp. 1201–1206, (1997).

[12] R.S. Sutton and A.G. 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).