View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Elsevier - Publisher Connector

ELSEVIER International Journal of Approximate Reasoning 25 (2000) 169–186
INTERNATIONAL JOURNAL OF APPROXIMATE REASONING
www.elsevier.com/locate/ijar

Optimal scheduling of progressive processing tasks

Shlomo Zilberstein [a,*], Abdel-Illah Mouaddib [b]

[a] *Department of Computer Science, University of Massachusetts, Amherst, MA 01002, USA*
[b] *CRIL/Université d'Artois, Rue de l'université, S.P. 16, 62307 Lens Cedex, France*

Received 1 October 1999; accepted 1 May 2000

Abstract

Progressive processing is an approximate reasoning model that 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 tasks such as information retrieval, automated diagnosis, or real-time image tracking and speech recognition. In performing these tasks it is often 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. Evaluation of the technique shows that it offers a significant improvement over existing heuristic scheduling techniques. Moreover, the framework presented in this paper can be applied to real-time scheduling of a wide variety of task structures other than progressive processing. © 2000 Elsevier Science Inc. All rights reserved.

Keywords: Progressive processing; Approximate reasoning; Scheduling; Real-time control; Markov decision problems; Resource-bounded reasoning

* Corresponding author. Tel.: +1-413-545-4189; fax: +1-413-545-1249.
E-mail addresses: zilberstein@cs.umass.edu (S. Zilberstein), mouaddib@cril.univ-artois.fr (A.-I. Mouaddib).

0888-613X/00/$ - see front matter © 2000 Elsevier Science Inc. All rights reserved.
PII: S0888-613X(00)00049-9

1. Introduction

Progressive processing is a model of computation for approximate reasoning that allows a system to satisfy a set of requests under time pressure [11,12]. The model is based on structuring each problem-solving component as a hierarchy of levels, each of which contributes to the overall quality of the result. Progressive processing is suitable for a wide range of applications such as hierarchical planning [8], model-based diagnosis [1] and speech recognition [3]. This model complements a large body of work on computational methods that offer a tradeoff between computational resources and quality of results. Closely related techniques include *anytime algorithms* [2,15], *imprecise computation* [3], *flexible computation* [7] and *design-to-time* [5]. However, the hierarchical structure of progressive processing facilitates an efficient management of computational resources and the ability to handle duration uncertainty effectively.

The distinctive characteristic of progressive processing is that each task is composed of a hierarchy of modules, each of which can produce an output of a certain quality. Each task in this framework is handled by a *progressive processing unit* or PRU. Each PRU includes multiple problem-solving modules that are called execution *levels*. The first level is mandatory. It produces an output with minimal acceptable quality. Each succeeding level of a given PRU is capable of improving the output quality (or reducing error). We assume in this paper that different tasks are independent and that each task has a deadline. We also assume that each level of a PRU is a non-preemptable module.

Progressive processing has been proved useful in real-time domains in which it is not feasible (computationally) or desirable (economically) to compute the best output for each task [11,12]. In order to use this approach, however, one needs to address the following real-time scheduling problem (also referred to as meta-level control). Given a set of PRUs, the question is how to decide what quality of output should be produced for each task so as to guarantee minimal quality for each task and maximize the overall performance (measured by the cumulative quality).

Fig. 1 illustrates a progressive processing task structure that includes five PRUs sorted by their deadlines. The number of levels per PRU ranges from two to four in this example. Once a level is executed, the scheduler/controller may decide to execute the next level so as to improve the output quality of the *same* PRU or to start executing a *different* PRU. The framework developed in this paper takes into account the *duration uncertainty* associated with each module. That is, we assume that the distribution of execution time is provided for each component of a PRU. The run-time selection of levels for execution is affected by the *actual* run-time of previous levels. Problem-solving techniques that rely on search have a high duration uncertainty. Therefore, taking into

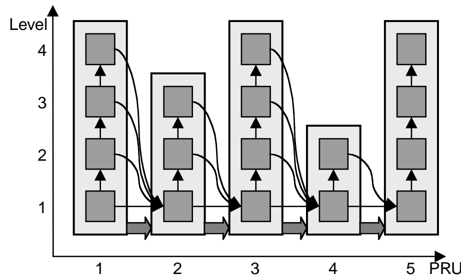

Fig. 1. An illustration of a progressive processing task structure.

account duration uncertainty can improve the scheduling of resource-bounded systems [5,6]. The solution presented in this paper is an improvement over a previous heuristic approach to the problem [13]. Moreover, the approach presented here can better handle dynamic environments under a high level of duration uncertainty.

The solution to the scheduling problem is based on reformulating it as a Markov decision problem (MDP) and finding an optimal policy (or schedule). 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 resource to each request. The vast majority of today's information providers uses 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. Unlike previous approaches to stochastic scheduling (that maximize the likelihood that a schedule will be successfully executed), the approach we develop leads to a *conditional* schedule that determines what components will be executed based on the actual progress of the computation.

The rest of this paper describes our solution in detail and evaluates the implementation of the scheduler. Section 2 describes the application and how each 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. Section 6 describes related work. We conclude with a summary of the benefits of the approach and a description of current efforts to enhance the expressibility of the model and make it suitable for a variety of approximate reasoning tasks.

2. Progressive processing of information queries

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., imprecise computation 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 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 $\{l_u^1, l_u^2, l_u^3, \ldots, l_u^k\}$,
- 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. 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 associated with a state are simply the rewards for executing each level or 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 the jth 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 $\mathcal{S} = \{[l_i^j, t] \mid u_i \in \mathcal{U}\}$ where $0 \leqslant j \leqslant \text{MaxLevel}(u_i)$ and $t \geqslant 0$ represents the remaining time to the deadline of u_i. When the system is in state $[l_i^j, t]$, the jth 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 must be performed PRU by PRU in the order of their deadlines. Enforcing this monotonicity constraint seems 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 $\mathscr{S} \times \{\mathbf{E}, \mathbf{M}\}$ into a discrete probability distribution over $\mathscr{S}$. Eqs. (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. Duration uncertainty defines the new state. Eq. (2) determines the transitions following successful execution and Eq. (3) determines the transition to the next PRU when the deadline of the current PRU is reached.

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

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

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,

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

$$R([l_i^j, t]) = q(l_i^j). \tag{5}$$

Now, we can define the value function (expected reward-to-go) for non-terminal states of the MDP as follows [14]:

$$V(s) = R(s) + \max_a P(s'|s,a)V(s'). \tag{6}$$

Using our former notation we get:

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

The top expression is the value of a move action and the bottom one 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:

$$V([l_n^m, t]) = R([l_n^m, t]) \tag{8}$$

and

$$V([l_n^j, 0]) = R([l_n^j, 0]). \tag{9}$$

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.

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 an algorithm that computes the value function and the optimal policy. Fig. 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 the horizontal axis showing the remaining time to the deadline of the PRU and the 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 Eq. (7).

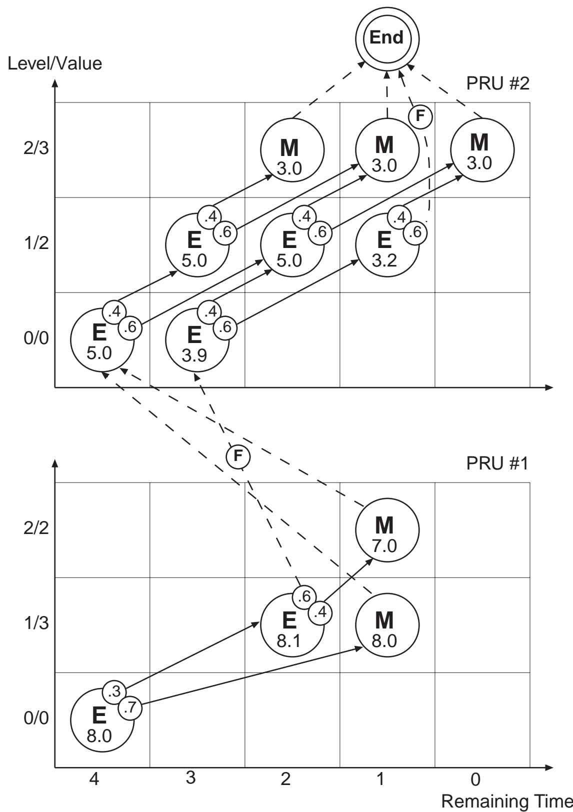

Fig. 2. An optimal monitoring policy with two PRUs.

4. Execution and monitoring of dynamic policies

In the previous section, we presented an optimal solution to the control problem of monotonic 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 (when policy revision takes a non-negligible amount of time), 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 nth 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 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 will a request for a revised policy 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 a 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 [9]. 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 the insertion of the new PRUs with all their levels and discards levels with the 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 domains 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.

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. The average number of levels per PRU is three. Fig. 3 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 run-time 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 [5], the most likely duration [13] or the worst-case duration [3]. 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.

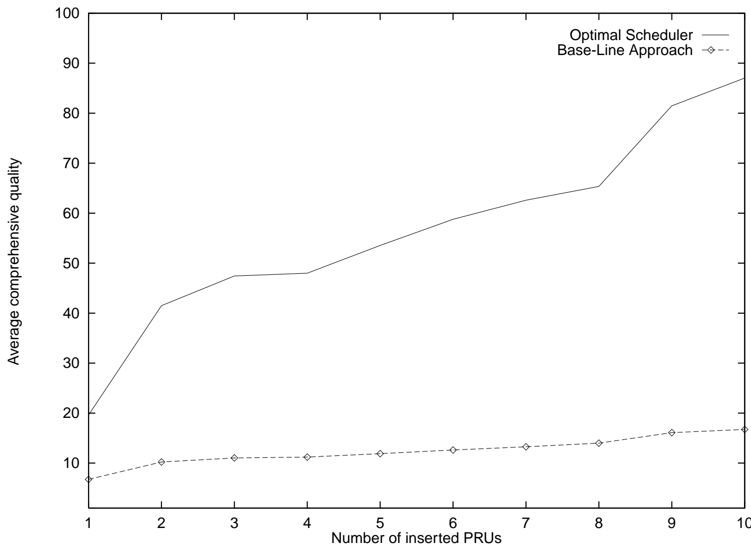

Fig. 3. Comprehensive quality with dynamic PRUs.

5.3. Handling duration uncertainty

This experiment shows the value returned by our approach and a baseline approach, similar to imprecise computation, for different degrees of uncertainty. In order to compare the results with the imprecise computation model, all PRUs have two levels representing the mandatory part and optional part of imprecise computation. The experiment assesses the suitability of our approach to environments with high levels of duration uncertainty. In addition, it shows the advantage over the imprecise computation model as duration uncertainty grows. This is due to the use of the worst-case duration by the latter model. When duration uncertainty is very high, the worst-case duration is significantly larger than the average, and it becomes more beneficial to apply our approach. The baseline approach is pessimistic and it discards the levels that *may* violate the deadline. The number of these levels becomes larger as duration uncertainty increases.

Problem instances were generated with 10 PRUs (with two levels each) and a variation of duration uncertainty from 0% to 100%. Fig. 4 shows that the comprehensive value (quality) is stable for our approach, while it is significantly affected by uncertainty in the baseline approach. Interestingly, after a short decline of expected quality for low variance, in our approach, 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 this. Of course, there is also a chance that

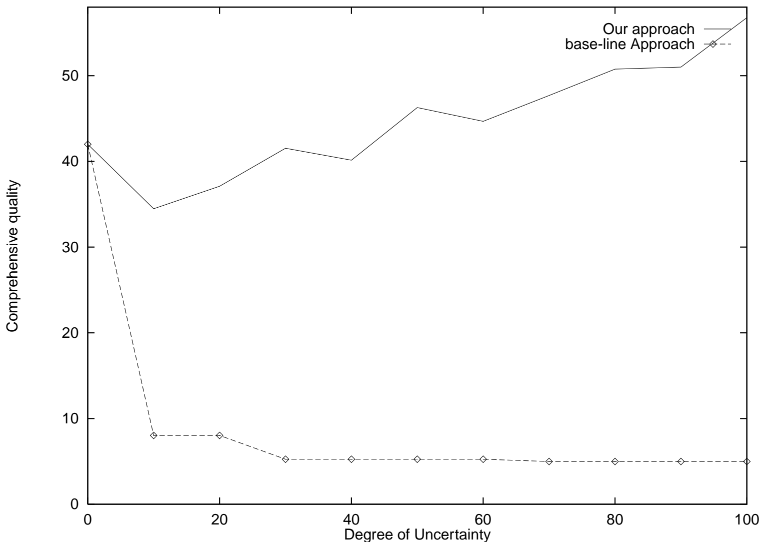

Fig. 4. Comprehensive quality for different degrees of uncertainty.

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 situations with high duration uncertainty.

To summarize, our experimental evaluation examines two fundamental questions. The first question is the the degree of performance gain by our approach with respect to a baseline approach based on a pessimistic, but safe, strategy using the worst-case duration. We show that, in dynamic situations, the performance gain is substantial because a pessimistic strategy discards all levels that *may* violate the deadline. The second question is the robustness of the approach under different levels of duration uncertainty. While the performance of the baseline approach deteriorates under a high level of uncertainty, our approach exhibits stable performance. These results suggest that the MDP scheduler offers an effective way to deal with duration uncertainty in real-time progressive processing.

6. Related work

In previous work, Mouaddib and Zilberstein have constructed an incremental scheduler to address the static version of the scheduling problem (a fixed set of PRUs) [13]. In this heuristic approach, scheduling can be seen as the problem of finding an optimal path that visits the maximum number of levels

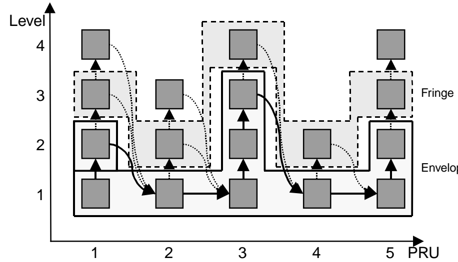

Fig. 5. An illustration of the operation of an incremental scheduler for progressive processing.

in the graph shown in Fig. 5 without violating any deadline. The scheduler starts its processing by building the schedule with the lowest quality (the minimal envelope) and refining it by inserting additional levels 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*.

More formally, the overall task structure is represented as a graph, $\mathscr{G}$, with the following components:

- The *envelope*, $\mathscr{E}$, is a subgraph of $\mathscr{G}$ containing all the levels scheduled for execution (nodes drawn within the lower (bold border) region in Fig. 5). This is similar to the notion of a *closed list* in search algorithms. The envelope is a dynamic structure that can be revised during execution.
- The *fringe*, $\mathscr{F}$, is the set of direct successors to the nodes in $\mathscr{E}$ (nodes drawn within the upper (dashed border) region in Fig. 5). This is similar to the notion of an *open list* in a search that contains nodes on the frontier of the search tree that are candidates for expansion.
- The *frozen space*, $\mathscr{Z}$, is a set of branches pruned during the scheduling cycle. It contains the levels that violate some deadlines during the scheduling phase. These levels are saved because, during the revision of the schedule, they may be added to the search space and eventually added to the schedule.

The construction of the schedule is based on a series of cycles of expansion of the current envelope. This expansion consists of inserting levels of the fringe into the envelope. The process is repeated until a maximal envelope is reached (i.e., any further expansion leads to a violation of a deadline) or until an external event causes the interruption of scheduling. At each cycle, a schedule is available, and its quality is improved from one cycle to another. The progressive processing task structure allows one to perform a utility-based scheduling by incrementally inserting nodes (processing levels) into the current schedule. The inserted node is the one with the highest contribution to the overall quality.

While the above approach is suitable for real-time scheduling (because it is very fast computationally), it has two major limitations. First, the utility-directed greedy scheduling algorithm is a local optimization approach to the scheduling problem. In addition, this approach is suitable for applications with duration uncertainty, but its effectiveness diminishes under a high level of uncertainty. The MDP scheduler presented in this paper addresses these limitations effectively.

The results reported in this paper are closely related to a large body of work on the imprecise computation model [3,4,10]. Each task in this model is decomposed into a *mandatory* subtask and an *optional* subtask. The mandatory subtask must be executed to produce results of some initial value; the optional subtask may be executed to increase the value of the results. With a few exceptions, tasks in this model are assumed to be independent and to have individual deadlines. Several scheduling algorithms have been developed for imprecise computation under certain assumptions about the optional part. For example, precision of the result can be assumed to improve linearly or by steps through the execution of the optional part [9]. In comparison, progressive processing can be viewed as a type of imprecise computation in which the optional part is a sequence of steps (or what we call levels). Unlike classical approaches to scheduling, the schedule constructed by the MDP scheduler is a *conditional schedule*. The selection of computational modules is conditioned on the *actual* execution times of previous modules. As a result, the system can react quickly (without rescheduling) to unlikely events (such as slower-than-expected progress in problem-solving).

7. 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 [13]. 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.

We are currently extending the MDP approach to address a more general type of progressive processing task structures. These extensions focus on the following three aspects of the model.

7.1. 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 includes only duration uncertainty with a set of transitions that includes 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: $s = (l^i_j, q^i_j, t)$, where q^i_j 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.

7.2. 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.

7.3. Enhanced task structures

The standard model requires 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 with 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 imposes 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 situations in which no strict deadlines are imposed on each task and the overall utility is time-dependent.

Our work on the extended task structure indicates that the MDP approach is suitable and relatively easy to apply to a wide variety of real-time computational models. The approach effectively handles the uncertainty about solution quality and duration and allows the system to construct an optimal schedule.

Acknowledgements

Victor Danilchenko helped with the implementation of 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, B. Hayes-Roth, Guaranteeing real-time response with limited-resources, Artificial Intelligence in Medicine 5 (1) (1993) 49–66.

[2] T.L. Dean, M. Boddy, An analysis of time-dependent planning, in: The Seventh National Conference on Artificial Intelligence, 1988, pp. 49–54.

[3] W. Feng, J. Liu, An extended imprecise computation model for time-constrained speech processing and generation, IEEE Workshop on Real-Time Applications, 1993, pp. 76–80.

[4] W. Feng, J. Liu, Algorithms for scheduling tasks with input error and end-to-end deadlines, Technical Report UIUCDCS-R-94-1888, Department of Computer Science, University of Illinois at Urbana-Champaign, 1994.

[5] A. Garvey, V. Lesser, Design-to-time real-time scheduling, IEEE Transactions on Systems, Man, and Cybernetics 23 (6) (1993) 1491–1502.
[6] E.A. Hansen, S. Zilberstein, Monitoring the progress of anytime problem-solving, in: The 13th National Conference on Artificial Intelligence, 1996, pp. 1229–1234.
[7] E. Horvitz, Reasoning about beliefs and actions under computational resource constraints, in: Workshop on Uncertainty in Artificial Intelligence, 1997.
[8] C. Knoblock, Automatically generating abstractions for planning, Artificial Intelligence 68 (1994) 243–302.
[9] J. Liu, K. Lin, W. Shih, A. Yu, J. Chung, W. Zao, Algorithms for scheduling imprecise computations, IEEE Transactions on Computer 24 (5) (1991) 58–68.
[10] V. Lopez-Millan, W. Feng, J.W.-S. Liu, Using the imprecise-computation technique for congestion control on a real-time traffic switching element, in: The International Conference on Parallel and Distributed Systems, 1994.
[11] A.-I. Mouaddib, Contribution au raisonnement progressif et temps réel dans un univers multi-agents, PhD thesis, University of Nancy I, 1993 (in French).
[12] A.I. Mouaddib, S. Zilberstein, Knowledge-based anytime computation, in: The 14th International Joint Conference on Artificial Intelligence, 1995, pp. 775–781.
[13] A.I. Mouaddib, S. Zilberstein, Handling duration uncertainty in meta-level control of progressive reasoning, in: The 15th International Joint Conference on Artificial Intelligence, 1997, pp. 1201–1206.
[14] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, MIT Press/Bradford Books, Cambridge, MA, 1998.
[15] S. Zilberstein, Using anytime algorithms in intelligent systems, AI Magazine 17 (3) (1996) 73–83.