Adaptive Control of Acyclic Progressive Processing Task Structures

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

Shlomo Zilberstein
Computer Science Dept
University of Massachusetts
Amherst, MA 01003 USA
zilberstein@cs.umass.edu

Richard Washington
NASA Ames Research Ctr
MS 269-2 Moffet Field
CA 94035, USA
rwashington@arc.nasa.gov

Abstract

The progressive processing model allows a system to trade off resource consumption against the quality of the outcome by mapping each activity to a graph of potential solution methods. In the past, only semi-linear graphs have been used. We examine the application of the model to control the operation of an autonomous rover which operates under tight resource constraints. The task structure is generalized to directed acyclic graphs for which the optimal schedule can be computed by solving a corresponding Markov decision problem. We evaluate the complexity of the solution analytically and experimentally and show that it provides a practical approach to building an adaptive controller for this application.

1 Introduction

Planetary rovers are unmanned vehicles equipped with cameras and a variety of scientific sensors. They have proved to be a cost-effective mechanism in space exploration and will continue to play a major role in future NASA missions [Washington *et al.*, 1999]. Despite dramatic improvements in capabilities and computational power, space systems continue to operate with scarce resources such as power, computer storage, and communication bandwidth. They also have a limited life span. As a result, it is necessary to maximize operation time during periods of no communication with the control center. It is also necessary to manage resources carefully when deciding what activity should be performed and how to best accomplish the rover's tasks. The main objective of this work is to develop planning and monitoring algorithms that can optimize scientific return when operating with limited resources.

For this purpose, the progressive processing model provides a suitable framework [Mouaddib and Zilberstein, 1997; 1998]. Progressive processing allows a system to trade off computational resources against the quality of the result similar to other resource-bounded reasoning techniques such as *"flexible computation"* [Horvitz, 1987],*"anytime algorithms"* [Dean and Boddy, 1988; Zilberstein, 1996], *"imprecise computation"* [Hull *et al.*, 1996; Liu *et al.*, 1991] and *"design-to-time"* [Garvey and Lesser, 1993]. The specific characteristic of progressive processing is the mapping of the computational process (or a plan) into a hierarchical graph of alternative methods. Each method has an associated probabilistic description of the resource/quality tradeoff it offers.

In previous work, Mouaddib and Zilbertstein [Mouaddib and Zilberstein, 1997; 1998] presented a solution to the control problem of progressive processing. This problem is to decide at run-time which subset of methods should be executed so as to maximize the overall utility of a system. The reactive controller is adaptive and takes into account the remaining resources and possible changes in the environment. This is achieved by reformulating the control problem as a Markov decision process (MDP). An efficient implementation that can handle run-time modifications of the overall plan has been implemented [Mouaddib and Zilberstein, 1998]. In the past, only semi-linear task structures have been used. That is, each step of a task could be implemented by one of several alternatives, after which the next step is considered.

The rover application, however, requires several extensions of the previously developed controller. Figure 1 shows a sample plan that illustrates the problem. In the plan, the rover locates an object and aims the camera, possibly after moving closer to the object. It can take a picture at one of three different resolutions, and can compress it using either high or low compression methods. In between, depending on the analysis of the image, the rover may decide to investigate the chemical components of the object. This plan presents several new requirements that have not been previously addressed:

Task inter-dependency: Task execution may depend on the outcome of previous actions. For example, the ability to investigate the chemical components depends on the results of image analysis.

Non linearity: The overall task structure becomes a directed acyclic graph. Execution follows a linear path through the graph.

Multiple resources: The controller must optimize its operation with respect to multiple resources.

The contribution of this paper is three fold. First, we generalize progressive processing to non-linear task structures with one resource and construct an optimal controller for such plans. Second, we analyze the complexity of the solution both analytically and empirically and show that it is an effective approach. Third, we examine the implications of increasing

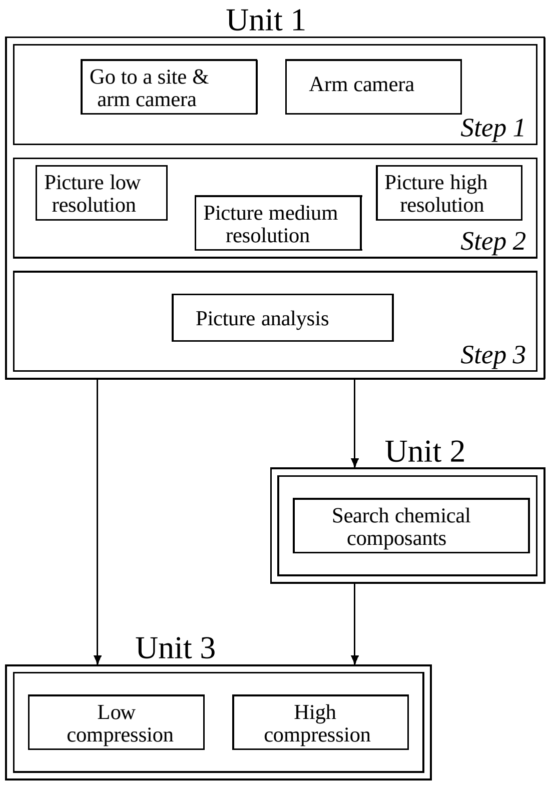

Figure 1: Planetary rover's plan

the number of resources and size of the plan on the complexity. We address this problem by introducing the notion of opportunity cost that can be estimated quickly for large, non-linear plans.

2 Progressive processing control problem

The work presented in this paper builds on the concepts of progressive processing units and plans. In this section, we define these notions and define our problem.

2.1 Preliminary definitions

Progressive processing is a reasoning technique that creates an incremental solution of a problem by using a hierarchy of steps, where each step contributes to the solution. This hierarchy can be viewed as the reasoning scheme which leads us to the solution. We denote by *PRU* (progressive processing unit) this hierarchy. Notice that we may associate a utility to the execution of a PRU's step. This utility is an abstract measure of the step's quality in the solution.

With this intuitive definition, a rover's plan may be seen as an ordered set of PRUs. More generally, the plan may be non-linear, i.e., a directed acyclic graph of PRUs. One unit has many successors. In this graph, a step will be considered for execution when the succession of steps executed so far has sufficient quality to enable this step.

For the example of planetary rovers, we have three PRUs. The first corresponds to going to a site, taking a picture and analyzing this image in order to find rocks. The second depends on the analysis results. If the site contains rocks, the rover may activate its APXS (alpha proton X-ray spectrometer) in order to determine the chemical components of the rocks. The last PRU corresponds to compressing the scientific data (images and chemical components).

We may transform this plan to a linear plan but, in that case, we lose generality and the inter-unit dependency information. In fact, the non-linearity of a plan and the notion of dependence allow for more realistic applications.

The planetary rover's problem (or progressive processing problem control) is to choose the set of executing tasks that maximize the global utility while respecting resource constraints.

2.2 Formal definitions

We formalize these definitions, focusing on one resource: operation time.

Definition 1 *A **plan** $\mathcal{P}$ is a directed acyclic graph of progressive processing units, $P_1, P_2, ..., P_{\mathcal{N}}$.*

Example: One plan is figure 1, where P_1 is unit 1, P_2 is unit 2 and P_3 is unit 3.

Definition 2 *A **progressive processing unit** (PRU), P_i, consists of a set of steps, $n_i = \{N_{i,1}, N_{i,2}, ..., N_{i,\#n_i}\}$, a deadline d_i and a set of successors, $s_i = \{succ(j_1), ..., succ(j_{\#s_i})\}$, where $succ(j_l) = (I^Q_{j_l}, P_{j_l})$ and $I^Q_{j_l}$ is the utility interval $[MinQ_{j_l}, MaxQ_{j_l}]$ corresponding to minimal and maximal utilities required to move to the next unit P_{j_l}.*

Example: PRU P_1 has three steps: $N_{1,1}$ is step 1, $N_{1,2}$ is step 2 and $N_{1,3}$ is step 3. It has two successors: P_2 and P_3.

Definition 3 *A **step** j of a unit i, $N_{i,j}$, consists of a set of tasks, $t_{i,j} = \{T_{i,j,1}, T_{i,j,2}, ..., T_{i,j,\#t_{i,j}}\}$ and a user-specified attribute, skippable, which indicates if this step may be skipped or not. We denote by $s_{i,j}$ the skippable attribute, and $s_{i,j} = 1$ when $N_{i,j}$ is skippable and 0 otherwise. We define $N_{i,0}$ to be the special step (with no tasks) that is used to show the beginning of executing a unit.*

Example: Step 2 of unit 1 has three tasks. The first $T_{1,2,1}$ is "Picture low resolution", the second $T_{1,2,2}$ is "Picture medium resolution" and the third $T_{1,2,3}$ is "Picture high resolution". This step can be skipped if we don't want to take a picture at a particular site.

Definition 4 *A **task** k of a step j of a unit i, $T_{i,j,k}$, consists of an executable module, a utility (or quality) and a discrete probability distribution $rc_{i,j,k}$ of resources consumed when the module is executed. The number of elements of $rc_{i,j,k}$ is $\#rc_{i,j,k}$.*

The probabilities can be determined from ground experimental rovers and simulations.

Example: Task 2 of step 2 of unit 1, $T_{1,2,2}$ consists of taking a medium resolution picture of the site. The distribution of resource consumption for this task, as used in our experiments, is $rc_{1,2,2} = \{(28, 0.2), (29, 0.6), (30, 0.2)\}$.

Definition 5 *The **progressive processing control problem** is to select at run-time the set of tasks that maximizes the global utility.*

2.3 Selecting tasks

When the rover has completed step $N_{i,j}$ (meaning that a task $T_{i,j,k}$ of that step has been executed or that the step has been skipped), it has three possible decisions:

- it can execute one task $T_{i,j+1,k'}$ of the next step $N_{i,j+1}$ if this step exists,
- it can skip the next step $N_{i,j+1}$, if it exists and it is skippable,
- it can move to a accessible successor unit $P_{succ(i)}$.

The optimal decision is the one that maximizes the global utility. The sequence of decisions will determine the set of tasks executed. The global utility is the cumulative reward of the executed tasks (reward is measured by the qualities associated to the tasks).

In this context, the progressive processing control problem is a state space where a transition from one state to an another is the decision made by the rover. The state represents the last executed step and the remaining resources. In fact, the choice of the next action (when the rover is in a particular state) depends only on the current state. This is the **Markov property**. So, the progressive processing problem control may be seen as a problem of controlling a Markov decision process.

2.4 Markov Decision Process Controller

We now define the corresponding Markov decision process.

States

A Markov decision process is a graph of states such that:

Definition 6 *A **state**, $[N_{i,j}; RR]$, consists of the last executed step $N_{i,j}$ and the remaining resources.*

Definition 7 *The **accumulated quality** is the sum of qualities of all executed task.*

Definition 8 ***Quality dependency of units**: A state $[N_{i',0}; RR']$ is the successor of $[N_{i,j}; RR]$ when the accumulated quality of $[N_{i,j}; RR']$ belongs to the utility interval $I^Q_{i'}$ and $(I^Q_{i'}, P_{i'}) \in succ_i$. We say in this case that $[N_{i',0}; RR']$ is an accessible state from $[N_{i,j}; RR']$.*

With the quality dependency, we express the dependence between the problems, including the precedence constraint (quality not null or belonging to a certain interval).

Example: To go to site 2, the rover should have executed $T_{1,3,1}$ (analysis of taken picture). So, we give a high reward to $T_{1,3,1}$ and $MinQ_2 = q_{1,3,1}$, where $(I^Q_2, P_2) \in succ_1$.

Definition 9 *A state is **terminal** when all the resources have been fully elapsed, or when there is no accessible successor.*

Transitions

We have three possible transition types: E, S, and D.

Definition 10 *A **transition** $\mathbf{E_{i,j,k}}$ (type E) is probabilistic. It consists of executing a task $T_{i,j,k}$ of the next step $N_{i,j}$. We denote it $[N_{i,j-1}; RR] \xrightarrow{E_{i,j,k}} [N_{i,j}; RR - \Delta R]$, where ΔR represents the resources consumed for the execution of $T_{i,j,k}$.*

Definition 11 *A **transition** $\mathbf{S_{i,j}}$ (type S) is deterministic. It consists of skipping the next step $N_{i,j}$. We denote it $[N_{i,j-1}; RR] \xrightarrow{S_{i,j}} [N_{i,j}; RR]$.*

Definition 12 *A **transition** $\mathbf{D_i}$ (type D) is deterministic. It consists of passing to an another unit. We denote it $[N_{i',j}; RR] \xrightarrow{D_i} [N_{i,0}; RR]$, where P_i is an accessible successor of $P_{i'}$.*

Transition rules

Now, let us define transition rules.

Pr(state f | state e, transition t) is the probability of moving from e to f when taking transition t.

Let $e = [N_{i,j-1}; RR]$ be the current state.

1. Transitions $E_{i,j,k}$
   Let ΔR be the resources consumed by task $T_{i,j,k}$.
   (a) $\Delta R \leq RR$, $Pr([N_{i,j}; RR - \Delta R] | e, E_{i,j,k}) = Pr(rc_{i,j,k} = \Delta R)$,
   (b) $\Delta R > RR$, $Pr([N_{i',0}; d_{i'} - d_i] | e, E_{i,j,k}) = Pr(rc_{i,j,k} > RR)$, where $[N_{i',0}; d_{i'} - d_i]$ is accessible from e.
2. Transitions $S_{i,j}$ (if $s_{i,j} = 1$, i.e., the step is skippable)
   $Pr([N_{i,j}; RR] | e, S_{i,j}) = 1$.
3. Transitions D_i
   $Pr([N_{i',0}; d_{i'} - d_i + RR] | e, D_i) = 1$, where $[N_{i',0}; d_{i'} - d_i + RR]$ is accessible from e.

2.5 State's value

The value associated to each state $V([N_{i,j}; RR])$ is a measure of utility of this state in the MDP. This value is calculated recursively by using the value of successor states.

For this, we denote $e = [N_{i,j-1}; RR]$, $\mathcal{E_A} = \{$*states accessible from* $e\}$ and $ea_{i'} \in \mathcal{E_A}$ an accessible state for which the associated task is in unit $P_{i'}$.

- Transitions $E_{i,j,k}$

$$\begin{aligned} v_E &= \sum_{\Delta R \leq RR} [Pr(rc_{i,j,k} = \Delta R) \\ &\qquad \cdot V([N_{i,j}; RR - \Delta R])] \\ &\quad + q_{i,j,k} \\ v_F &= \max_{\forall i', ea_{i'} \in \mathcal{E_A}} [Pr(rc_{i,j,k} > RR) \\ &\qquad \cdot V([N_{i',0}; d_{i'} - d_i])] \end{aligned}$$

- Transitions $S_{i,j}$

$$v_S = s_{i,j} \cdot V([N_{i,j}; RR])$$

- Transitions D_i

$$v_D = \max_{\forall i', ea_{i'} \in \mathcal{E_A}} V([N_{i',0}; d_{i'} - d_i + RR])$$

So, the value of one state is:

$$V(e) = \max_{\forall E_{i,j,k}, S_{i,j}, D_i} (v_E + v_F, v_S, v_D) \qquad (1)$$

3 Performance of the MDP controller

We have implemented the execution model described above, the policy-construction algorithm, and a simulator that allows plans to be executed and visualized in the rover domain. The performance of the resulting system is evaluated using the simulator. This section illustrates the results obtained using the system.

3.1 Analysis

Because of the one-to-one correspondence between the states of MDP and the states of the control, the optimal solution of the MDP is the optimal control.

Claim 1 (MDP size) *The number of states in the MDP is bounded by:*

$$\mathcal{N} \cdot \max_{\forall P_i} d_i \cdot \max_{\forall P_i} \#n_i \qquad (2)$$

Proof: Note that two states with same remaining resources and associated step generate the same successors. So, to avoid an exponential complexity, we must notice already created states.

Since consumable resources are bounded, as well as the number of steps, we obtain a bounded number of states. Moreover, two states are different if and only if the associated steps are different or the remaining resources are different. So, we have the bound: $\mathcal{N} \cdot \max_{\forall P_i} d_i \cdot \max_{\forall P_i} \#n_i$. $\square$

Claim 2 (solution time) *The time elapsed to calculate the optimal policy for the MDP is bounded by:*

$$\mathcal{N} \cdot (\max_{\forall P_i} d_i)^2 \cdot \max_{\forall P_i} \#n_i \cdot \left[\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#rc_{i,j,k} + 2 \right] \qquad (3)$$

Proof: The time consumed to solve the MDP is bounded by the number of transitions multiplied by the cost of searching already created states (if we don't want an exponential complexity).

The number of transitions from a state $[N_{i,j-1}; RR]$ is bounded by the number of transitions of each type: $\#rc_{i,j,k}$ of type E, and one each of type S and D. So, for that one state, the number of transitions generated is bounded by:

$$\left[\sum_{k=1}^{\#t_{i,j}} \#rc_{i,j,k} \right] + 2 \leq \max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#rc_{i,j,k} + 2.$$

Therefore, the bound on the number of transitions in the entire MDP is (using equation 2):

$$\mathcal{N} \cdot \max_{\forall P_i} d_i \cdot \max_{\forall P_i} \#n_i \cdot \left[\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#rc_{i,j,k} + 2 \right]$$

Finally, the search of already created states involves d_i states in the worst case: since we store states by unit and by step, only those states associated with the specified unit and step need to be checked.

So, we obtain the following bound:

$$\mathcal{N} \cdot (\max_{\forall P_i} d_i)^2 \cdot \max_{\forall P_i} \#n_i \cdot \left[\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#rc_{i,j,k} + 2 \right]$$

$\square$

In conclusion, our controller has a complexity of $\mathcal{O}(\mathcal{N} \cdot \max_{\forall P_i} d_i)$ in space and $\mathcal{O}(\mathcal{N} \cdot (\max_{\forall P_i} d_i)^2)$ in policy-construction time. We also notice that if the deadline is not a function of N, we obtain a linear complexity.

3.2 Experimental results

We take a non-linear plan composed of $\mathcal{N}$ units and of depth $\log \mathcal{N}$. Each unit is composed of four steps. The first step consists of going to a site, the second of arming the camera, the third of taking the photo (low, medium or high resolution) and the last of compressing the obtained scientific data.

For our experiments, we set the average time to execute this unit to be 32 seconds. So, the deadline of each unit in plan is 32 seconds multiplied by the depth of this unit in the plan. If we assume that the number of nodes at each level of the plan increases by at least a constant factor $b > 1$, then the depth of the plan is $\mathcal{O}(\log \mathcal{N})$, and we obtain the complexity $\mathcal{O}(\mathcal{N} \cdot \log \mathcal{N})$ in space and $\mathcal{O}(\mathcal{N} \cdot (\log \mathcal{N})^2)$ in time.

However, we notice that we use exactly the resources needed to execute a task. What happens if we consume resources in fixed-size packets instead of exactly the needed resources?

$\mathcal{N}$	Value (size=1)	Value (size=2)	Error
20	94.38	90.80	3.79%
40	133.76	129.88	2.90%
60	166.76	160.71	3.63%
80	197.37	187.95	4.77%
100	219.37	209.51	4.49%

Table 1: Value computed as a function of $\mathcal{N}$, using packets of resources.

Table 1 shows state values computed for packet sizes of 1 and 2; little error is incurred by increasing the packet size. Figure 2 shows policy-creation time for plans of realistic size. Using a packet size of 2, the time used to solve the MDP is divided by 4 and the number of states created is divided by 2.

In fact, we can generalize this result:

Claim 3 ***If*** *EC is the number of states created and TC is the time consumed to create these states (for packet size equal to 1),* ***then*** *$\frac{EC}{u}$ is the maximal number of created states and $\frac{TC}{u^2}$ the maximal time elapsed (for packet size equal to u).*

$\frac{EC}{u}$ and $\frac{TC}{u^2}$ are bounds since equations 2 and 3 are bounds for EC and TC.

Proof: We suppose that EC and TC are number of created states and time consumed for packet size equal to 1. We now take packets of u resources.

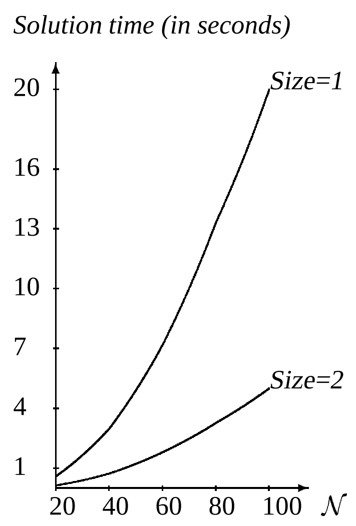

Figure 2: Solution time for packet size equal to 1 and 2

Notice that size of packet has no influence on the size of the problem $\mathcal{N}$. Moreover, it leaves the number of steps in unit and number of tasks in step invariant.

So, the packet size proportionally reduces the deadline of each unit ($d_i \to \frac{d_i}{u}$). Therefore, equation 2 implies that $\frac{EC}{u}$ is the maximal number of created states and equation 3 implies that $\frac{TC}{u^2}$ is the maximal time consumed to create these states.

The last term of equation 3 can be changed by packets, but it is just a constant. □

4 Dynamic operation and scalability

In the previous section, we presented an optimal solution to the control problem of acyclic progressive processing task structures. In this section, we extend the applicability of the solution to situations in which the global policy cannot be computed at run-time either by the control center or the rover itself. There are two primary motivations to avoid run-time construction of the policy.

1. **Dynamic operation:** We want to be able to modify the plan at run-time (e.g., insert or delete a PRU) without necessarily recomputing the control policy.
2. **Scalability:** We want to be able to track more resources, and the complexity of global policy construction grows exponentially with the number of resources.

While we focus in this paper on the dynamic operation, scalability is also enhanced by the proposed solution. Our approach is to exploit the fact that the units of the plan are largely independent. We try to capture the dependency of the execution of each unit on the remaining plan using a notion similar to opportunity cost [Zilberstein and Mouaddib, 1999].

4.1 Dynamic control with one resource

Definition 13 *Let $v_L([N_{i,j}; RR]$ represent the optimal local value of the state after step $N_{i,j}$ with remaining resources RR.*

Note that v_L is the same as V for a modified plan in which unit i has no successors. v_L is simply the value of the best policy for PRU i ignoring the remaining plan, hence it is called local value.

Claim 4 *The global value function can be reformulated as follows.*

$$V([N_{i,j}; RR]) = \max_{\Delta R}\{v_L([N_{i,j}; RR - \Delta R] + v_D([N_{i,j}; \Delta R])\} \quad (4)$$

Proof: This is an immediate result of the definition of v_L, V and v_D, and the additivity of the utility. □

For each PRU, we can easily compute the local value function v_L and construct a local policy off-line. The global value function (and policy) can be constructed at run-time, if we can obtain a fast estimate of v_D.

4.2 Estimating v_D

One way to think about v_D is as the value of terminating the work on a certain PRU with some level of resources, then using the resources for the remaining plan. This is exactly what we need to know. Estimating v_D is equivalent to estimating the global value of the remaining plan. We have examined several approximation schemes for v_D. Due to space limitations, we only sketch one simple approach.

- Construct an optimal local value function and local policy for each PRU (done once, off-line).
- For each level of RR, compute the expected value and expected resource consumption by the optimal local policy (done once, off-line).
- Go over the plan graph "backwards" from the leaves and compute an expected value for each node, for each level of RR at that node. The backup is relatively simple: maximizing for each level of RR at that node the combined parent/child value. In this process, it is necessary to take into account the difference between the deadlines of each parent/child pair. The quality dependency constraints are enforced with respect to the *expected* quality of the parent.

This algorithm is linear in the size of the graph. The computational savings and sub-optimality of the outcome are the result of using precalculated expected resource consumption and value for each PRU.

5 Conclusion

We have presented a solution to the problem of adaptive control of acyclic progressive processing tasks. This approach, which relies on solving a corresponding MDP, generalizes earlier work on MDPC [Mouaddib and Zilberstein, 1998]; it permits for the first time the treatment of non-linear progressive processing task structures. Moreover, our approach addresses effectively the high degree of uncertainty regarding resource consumption. In that sense, it improves on existing models for resource-bounded reasoning such as *"imprecise computation"* [Hull *et al.*, 1996; Liu *et al.*, 1991] and *"design-to-time"* [Garvey and Lesser, 1993]. Finally, the model captures inter-task quality dependency similar to the enable and disable relationships in the *"design-to-time"* framework [Garvey *et al.*, 1993].

We also address in the paper a potential deficiency of the approach which may require a large amount of memory and time to create and solve the MDP and to store the resulting policy. Using packets of consumable resources, we managed to reduce the size of the MDP with a small loss of quality (4.5% for packet size equal to 2). Finally, we address the issue of dynamic operation and scalability by approximating the solution of the MDP.

The solution presented in this paper includes time as the only resource. Future work will allow us to represent a vector of resources [Mouaddib, 2000] such as storage capacity and power. An additional challenge is to develop more precise estimates of the value function v_D and adapt the approach to the case of multiple resources. A related project is using reinforcement learning to estimate v_D [Bernstein and Zilberstein, 2001].

More accurate models of resource usage can be obtained by using results from actual rover testing or high-fidelity simulations. We plan to make use of ongoing development of a high-fidelity rover simulation at NASA Ames Research Center to refine our models to make them more accurate and applicable to realistic rover problems, with the goal of testing our approach on actual rover hardware.

The PRU used in our experiments (Cf. 3.2) can be extended to solve a satellite scheduling problem. With this approach, more units can be scheduled than by traditional methods. Indeed, at this moment, a satellite in orbit around the world schedules the equivalent of only about 20 units. With our approach, it would be able to schedule 100, thus increasing its productivity 5-fold.

Acknowledgements

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

References

[Bernstein and Zilberstein, 2001] Daniel S. Bernstein and Shlomo Zilberstein. Reinforcement learning for weakly-coupled mdps and an application to planetary rover control. In *UAI (submitted for publication)*, 2001.

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

[Garvey and Lesser, 1993] A. Garvey and V. Lesser. Desing-to-time real-time scheduling. *IEEE Transaction on systems, Man, and Cybernetics*, 23:1491–1502, 1993.

[Garvey *et al.*, 1993] A. Garvey, M. Humphrey, and V. Lesser. Task interdependencies in desing-to-time real-time scheduling. In *AAAI*, pages 580–585, 1993.

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

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

[Liu *et al.*, 1991] J. Liu, K. Lin, W. Shih, J. Chung A. Yu, and W. Zhao. Algorithms sor scheduling imprecise computations. *IEEE Computer*, 24:58–68, 1991.

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

[Mouaddib and Zilberstein, 1998] A-I. Mouaddib and S. Zilberstein. Optimal scheduling of dynamic progressive processing. In *ECAI*, pages 449–503, 1998.

[Mouaddib, 2000] A. I. Mouaddib. Optimizing multi-criteria decision quality in a progressive processing system. In *AAAI Symposium on realtime autonomous agent*, pages 56–61, 2000.

[Washington *et al.*, 1999] R. Washington, K. Golden, J. Bresina, D. E. Smith, C. Anderson, and T. Smith. Autonomous rovers for mars exploration. In *Proceedings of The 1999 IEEE Aerospace Conference*, 1999.

[Zilberstein and Mouaddib, 1999] S. Zilberstein and A-I. Mouaddib. Reactive control of dynamic progressive processing. In *IJCAI*, pages 1268–1273, 1999.

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