ADAPTIVE RESOURCES CONSUMPTION IN A DYNAMIC AND UNCERTAIN ENVIRONMENT

An Autonomous Rover Control Technique using Progressive Processing

Simon Le Gloannec, Abdel Illah Mouaddib
GREYC UMR 6072, Université de Caen Basse-Normandie, Campus Côte de Nacre
bd Maréchal Juin, BP5186, 14032 Caen Cedex, France
{slegloan, mouaddib}@info.unicaen.fr

François Charpillet
MAIA team, LORIA, Campus Scientifique, BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France
charp@loria.fr

Keywords: MDP, Resource-bounded Reasoning, Hierarchical control, Autonomous agents, Planning and Scheduling.

Abstract: This paper address the problem of an autonomous rover that have limited consumable resources to accomplish a mission. The robot has to cope with limited resources: it must decide the resource among to spent at each mission step. The resource consumption is also uncertain. Progressive processing is a meta level reasoning model particulary adapted for this kind of mission. Previous works have shown how to obtain an optimal resource consumption policy using a Markov decision process (MDP). Here, we make the assumption that the mission can dynamically change during execution time. Therefore, the agent must adapt to the current situation, in order to save resources for the most interesting future tasks. Because of the dynamic environment, the agent cannot calculate a new optimal policy online. However, it is possible to compute an approximate value function. We will show that the robot will behave as good as if it knew the optimal policy.

1 INTRODUCTION

Resources consumption control is crucial in the autonomous rover context. For example, a rover must know when it has to go back to a docking station before having no energy left. The question for the agent is to know where, when, and the amount of resources it has to spend for a particular task. We want this agent to be adaptive : it has to choose the amount of resources to spend in each task by taking into account the expected value. This is the reason why we use progressive processing to model and to control the mission. This model of reasoning describes tasks that can be performed progressively, with several options at each step. The resources consumption is probabilistic. Progressive processing provides a meta-level resources consumption model. Then, we obtain an optimal policy control thanks to a MDP.

For now, progressive processing does not cope with dynamic environments. In a real robot application, it is not uncommon to see new tasks incoming during the mission. When it happens, the agent has to completely recalculate the resource control policy to behave optimally. Most of the time, it doesn't have sufficient time to do it. For these reasons, we propose here a value function approximation method that allows the agent to compute a local near optimal policy very quickly in order to cope with the dynamic changes. We will bring experimental result to validate this method.

The paper is divided into five main sections. The third section introduces progressive processing and its use for an autonomous rover mission application. In the fourth section, we present the dynamic environment and propose a way to cope with changes during the mission. This theory is experimentally validated in the last section.

2 RELATED WORK

- **Control of Progressive Processing:** Progressive processing has been introduced in (Mouaddib and Zilberstein, 1998), and has been used in a real rover application (Zilberstein et al., 2002) and in a retrieval information engine (Arnt et al., 2004). Concerning the retrieval information engine, users send requests to a server that can respond very precisely and use lots of resources, or approximatively and use less resources. Progressive processing permits to model the server tasks. It also permits to adapt the resources dedicated to each task by taking the number of requests into account. We will briefly present the rover application in this paper.

- **MDP Decomposition:** an MDP is used to formalise the mission resource consumption control. We can either compute an optimal policy or an approximate policy. Our approximation is based on MDP decomposition. This technique has been largely studied in the literature. Dean et al (Dean and Lin, 1995) investigated methods that decompose global planning problems into a number of local problem. In (Parr, 1998), the method is to build a cache of policies for each part of the problem independently, and then to combine the pieces in separate steps. (Meuleau et al., 1998) also proposed a decomposition technique for a problem where tasks have indepedent utilities, and are weakly coupled.
- **Value Function Approximation:** our work also deals with value function approximation. In (Feng et al., 2004), the state space is dynamically partitioned into regions where the value function is the same throughout the region. Authors make piecewise constant and piecewise linear approximations. (Pineau et al., 2003) introduced point base value iteration. It approximates an exact value iteration solution by selecting a small set of representative belief points and by tracking the value and its derivative for those points only.

3 MODEL A MISSION WITH PROGRESSIVE PROCESSING

The problem formalism will be presented hierarchically: the mission is divided into progressive processing units (PRU), which are separated into levels.

3.1 Formalism

A mission is a set of tasks, and each task is an acyclic graph vertex (see Figure 1). To facilitate access to the formalism, we assume that this acyclic graph is an ordered sequence. This involves no loss of generality. You can see this sequence as a particular path in the graph (for example A, B, E, F). There are **P** tasks in the sequence. Each task is modelled by a progressive processing unit (PRU). It is structured hierarchically. Each PRU_p, $p \in \{1,\dots,\mathbf{P}\}$ is a level ordered finite sequence $[\mathsf{L}_{p,1},\dots,\mathsf{L}_{p,L}]$. An agent can process a level only if the preceding level is finished. The process can be interrupted after each level. It means that the agent can stop the PRU execution, but it receives no reward for it. Other situations have been proposed when the agent can have a reward at each level. However, in our case, only a complete task accomplishment provides a reward.

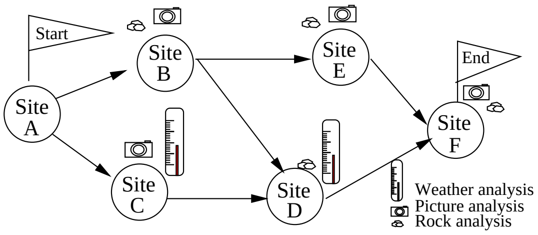

Figure 1: A mission.

Each level $\mathsf{L}_{p,\ell}$ contains one or more modules $[\mathsf{m}_{p,\ell,1},\dots,\mathsf{m}_{p,\ell,M}]$. A module is a specific way to execute a level. The agent can only execute one module per level. The execution of a module produces a quality **Q** and consumes resources. A progressive processing unit definition is illustrated on Figure 2 for a picture task. Firstly, the rover has to aim its camera, then it chooses the picture resolution. At the end, it saves the picture. The execution is performed from bottom to top.

The quality $\mathbf{Q}_{p,\ell,m} \in \mathbb{R}^+$ is a criterion to measure the module execution impact. There is no immediate reward after each level processing. The agent receives the sum of all the $\mathbf{Q}_{p,\ell,m}$ only when the last level is performed.

The resource consumption in a module $\mathsf{m}_{p,\ell,m}$ is probabilistic. We denote as $\mathcal{P}r_{p,\ell,m}$ the probability distribution of resources consumption.

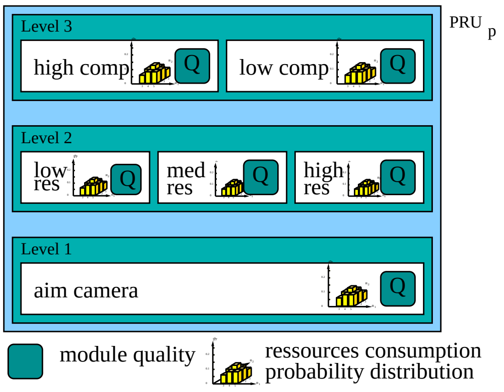

Figure 2: PRU.

3.2 Mission Control

The problem of control we address in this paper consists of a robot exploring an area where it has sites to visit and performs exploration tasks. The problem is that the robot cannot know in advance its resource consumption. We use then a MDP to control the mission. The agent is supposed to be rational. It must maximise the mathematical expected value. It computes a policy that correspond to this criterion before executing the mission. The on-line mission control process consists in following this policy. In the next section, we present the MDP model and the control policy calculation. At each site the agent must take a decision to **stay for continuing** the exploration or to **move to another site**.

3.2.1 Modelling the Mission as an Markov Decision Process

Formally, a MDP is tuple $\{\mathcal{S},\mathcal{A},\mathcal{P}r,\mathcal{R}\}$ where :

- $\mathcal{S}$ is a finite set of states,
- $\mathcal{A}$ is a finite set of actions,
- $\mathcal{P}r$ is a mapping $\mathcal{S}\times\mathcal{A}\times\mathcal{S}\rightarrow[0,1]$,
- $\mathcal{R}:\mathcal{S}\rightarrow\mathbb{R}$ is a reward function.

Given a particular rational criterion, algorithms for solving MDPs can return a policy π, that maps from $\mathcal{S}$ to $\mathcal{A}$, and a real-valued function $V:\mathcal{S}\rightarrow\mathbb{R}$. Here, the criterion is to maximise the expected reward sum.

3.2.2 States

In the progressive processing model, a state is a tuple $\langle \mathtt{r},\mathbf{Q},\mathtt{p},\ell\rangle$. $\mathtt{r}$ is a reel number that indicates the amount of remaining resources. $\mathbf{Q}$ is the cumulated quality since the beginning of the current PRU. $\mathtt{p}$ and ℓ indicates the last executed level $\mathsf{L}_{\mathtt{p},\ell}$ done. The failure state, denoted as $\mathbf{s}_{\mathtt{failure}}$ is reached when the resource quantity is negative. The reward is assigned to the agent only when the last level of the PRU has been successfully executed. Then, it is necessary to store the cumulated quality $\mathbf{Q}$ in the state description. We have introduced level 0 in order to represent the situation where the agent begins the PRU execution.

3.2.3 Actions

There are two kinds of actions in the progressive processing model. An agent can execute only one module in the next level or move to the next PRU. These two actions are depicted in Figure 3. Formally, $\mathcal{A}=\{\mathbf{E}_{\mathtt{m}},\mathbf{M}\}$. When the agent reaches the last level in a PRU$_{\mathtt{p}}$, it directly moves to the next PRU$_{\mathtt{p}+1}$.

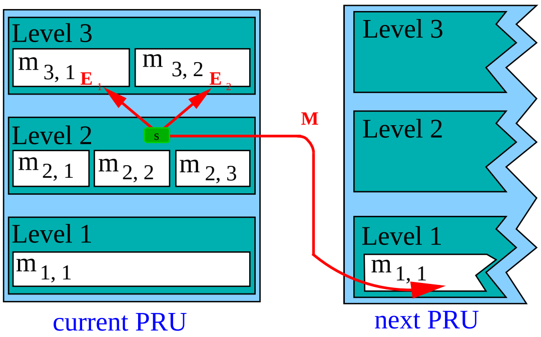

Figure 3: Actions.

Actually, $\mathbf{E}_{\mathtt{m}}$ is an improvement of the the current PRU whereas $\mathbf{M}$ is an interruption. The agent can progressively improve the PRU execution, it can also stop it at any moment.

3.2.4 Transitions

$\mathbf{M}$ is a deterministic action whereas $\mathbf{E}_{\mathtt{m}}$ is not. Indeed, the module execution consumes resources and this consumption is probabilistic. After executing a given module, the agent always cumulates a fixed quality. The uncertainty is related to the resources consumption probability distribution. Thus,

$$\begin{aligned}
\mathcal{P}r(\langle \mathtt{r},\mathbf{Q},\mathtt{p},\ell\rangle,\mathbf{M},\langle \mathtt{r},0,\mathtt{p}+1,0\rangle) &= 1\\
\mathcal{P}r(\langle \mathtt{r},\mathtt{p},\mathbf{Q},\ell\rangle,\mathbf{E}_{\mathtt{m}},\langle \mathtt{r}',\mathbf{Q}',\mathtt{p},\ell+1\rangle) &= \mathcal{P}r(\Delta\mathtt{r}|\mathtt{m}_{\mathtt{p},\ell,\mathtt{m}})\\
\text{where}\quad \mathbf{Q}' = \mathbf{Q}+\mathbf{Q}_{\mathtt{p},\ell,\mathtt{m}}&\\
\text{and}\quad \mathtt{r}' = \mathtt{r}-\Delta\mathtt{r}&
\end{aligned}\tag{1}$$

3.2.5 Reward

A reward is given to the agent as soon as it finishes a PRU. This reward corresponds to the cumulated quality through the modules path. If the agent leaves a PRU without finishing it, it receives no reward. This makes a sense for exploration task where the robot has no reward if the task is not completely finished.

$$\mathcal{R}(\langle \mathtt{r},\mathbf{Q},\mathtt{p},\mathsf{L}_{\mathtt{p}}\rangle) = \mathbf{Q} \tag{2}$$

$$\mathcal{R}(\langle \mathtt{r},\mathbf{Q},\mathtt{p},\ell<\mathsf{L}_{\mathtt{p}}\rangle) = 0 \tag{3}$$

where $\mathsf{L}_{\mathtt{p}}$ is the number of levels in PRU$_{\mathtt{p}}$.

3.2.6 Value Function

The control policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ depends on a value function that is calculated thanks to the Bellman equation 4. We assume that $V(\mathbf{s}_{\mathtt{failure}}) = 0$.

$$
\begin{aligned}
&V(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \ell \rangle) = \\
&\begin{cases} 0 \quad \text{if } \mathtt{r} < 0 \quad \text{(failure)} \\ \mathcal{R}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \ell \rangle) + \max(V_{\mathbf{M}}, V_{\mathbf{E}}) \quad \text{otherwise} \end{cases} \\
&V_{\mathbf{M}}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \ell \rangle) = \\
&\begin{cases} 0 & \text{if } \mathtt{p} = \mathbf{P} \\ V(\langle \mathtt{r}, \mathbf{0}, \mathtt{p}+1, 0 \rangle) & \text{otherwise} \end{cases} \\
&V_{\mathbf{E}}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{p}, \ell \rangle) = \\
&\begin{cases} 0 \text{ if } \ell = \mathtt{L}_{\mathtt{p}} \\ \max\limits_{\mathbf{E}_{\mathtt{m}}} \sum\limits_{\Delta \mathtt{r}} \mathcal{P}r(\Delta \mathtt{r} | \mathtt{m}_{\mathtt{p},\ell,\mathtt{m}}) . V(\langle \mathtt{r}', \mathbf{Q}', \mathtt{p}, \ell + 1 \rangle) \\ \text{where} \quad \mathbf{Q}' = \mathbf{Q} + \mathbf{Q}_{\mathtt{p},\ell,\mathtt{m}} \\ \text{and} \quad \mathtt{r}' = \mathtt{r} - \Delta \mathtt{r} \end{cases}
\end{aligned}
\tag{4}
$$

Action **E** consumes some resources whereas **M** does not. The agent will execute the module that gives it the best expected value.

3.3 Control Policy

The main objective is to compute a policy that the robot will follow in order to optimise the resource consumption during the mission. It has been proven that we could calculate an optimal policy (Mouaddib and Zilberstein, 1998). When the mission is fixed before execution time, we can get this policy with a backward chaining algorithm. With this policy, the robot can choose at any moment the decision that maximise the global utility function for the mission i.e. the obtained reward sum. But, this policy is fixed for a given mission. If the mission changes during execution time, this policy is no longer up to date.

Even if the policy algorithm is linear in the state space, the time needed to compute the optimal policy is often more than a module execution time. This occurs especially when the mission (and then the state space) is large. This is the reason why we propose an other way to calculate the policy. We will not calculate an optimal policy, but a near-optimal policy which could be used to take good decision when the mission changes.

4 THE DYNAMIC ENVIRONMENT

We suppose in this paper that tasks can come or disappear during execution time. This assumption is realistic for an explorer rover.

Instead of seeing the control as a global policy for the mission, we decompose it in two parts, the current PRU and the rest of the mission. The policy is only calculated for the current PRU. It only dependson the expected value function for the rest of the mission.

We could yet obtain the optimal policy with this method by simply calculating the expected value function for the rest of the mission without any approximation. Thus, we will naturally not save time. Then, we present here a method to calculate the expected value function very quickly.

5 VALUE FUNCTION APPROXIMATION

The quick value function approximation is based on a decomposition technique. Decomposition techniques are often use in large Markov decision processes. Here, we will re-compose an approximate expected value function in two times. Firstly, we calculate an optimal local value function for each PRU in the mission. These local functions are splitted into function pieces. Secondly, we re-compose the value function with all the function pieces as soon as a change occurs at run-time. This section is divided in two parts, the decomposition, and the recomposition. We denote as V^* the optimal value function and $V^\sim$ its approximation. The main objective of the recomposition is to fit V^* as good as possible.

5.1 Decomposition

The decomposition consists in the calculation of the value functions for each $\mathsf{PRU}_{\mathtt{p}}$ in the mission. These functions are indeed performance profiles (denoted as $f_{\mathtt{p}}$): they indicate the expected value for this PRU if $\mathtt{r}$ are allocated. Three performance profiles examples are depicted in Figure 4. The PRU γ can not consume more than 31 resource units, and give a maximum expected value of 25.

The main idea is to use these functions to recompose the global approximate value function. The performance profiles receive a preliminary treatment. The progressive processing provides value functions that are similar to anytime reasoning value functions :

- they increase with the allocated resource amount,

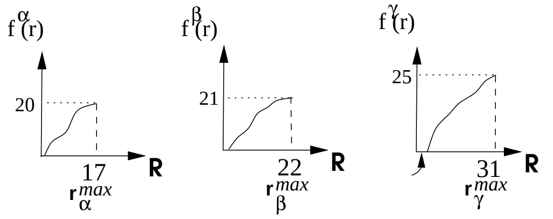

Figure 4: 3 PRU performance profiles.

- the best improvements are made with few resources, the growth of V^* is higher at its beginning than at its end.

For these reasons we split f_p into i_p pieces $g_{\mathrm{p},1},\dots,g_{\mathrm{p},i_\mathrm{p}}$. During the recomposition, we will take the best pieces to re-compose the beginning of $V^\sim$, and the worst for the end. We keep the part of f_p with the best growth, i.e. the piece between $(0,0)$ and $(\mathtt{r}^{max}_{\mathrm{p},1}, f_\mathrm{p}(\mathtt{r}^{max}_{\mathrm{p},1}))$. $\mathtt{r}^{max}_{\mathrm{p},1}$ is the resource amount that provides the best tradeoff between the resources spent and the local expected value.

$$\mathtt{r}^{max}_{\mathrm{p},1} = \operatorname*{argmax}_{\mathrm{p},\mathtt{r}}\left(\frac{f_\mathrm{p}(\mathtt{r})}{\mathtt{r}}\right) \qquad (5)$$

$$\forall i > 1, \mathtt{r}^{max}_{\mathrm{p},i} = \operatorname*{argmax}_{\mathrm{p},\mathtt{r}}\left(\frac{f_{\mathrm{p},i}(\mathtt{r})}{\mathtt{r}}\right) \qquad (6)$$

When the first $\mathtt{r}^{max}_{\mathrm{p},1}$ is found, we save the first piece of f_p as $g_{\mathrm{p},1}$. We split f_p in $g_{\mathrm{p},1}$ and $f_{\mathrm{p},2}$. We continue to search the second $\mathtt{r}^{max}_{\mathrm{p},2}$ in $f_{\mathrm{p},2}$; we save $g_{\mathrm{p},2}$ etc... until $f_{\mathrm{p},i_\mathrm{p}}$ is empty.

$$g_{\mathrm{p},1} : \left| \begin{array}{l} \left[0\dots\mathtt{r}^{max}_{\mathrm{p},1}\right] \rightarrow \mathbb{R} \\ \mathtt{r} \rightarrow f_\mathrm{p}(\mathtt{r}) \end{array} \right.$$

$$f_{\mathrm{p},1} : \left| \begin{array}{l} \left[0\dots\mathtt{r}^{max}_{\mathrm{p}} - \mathtt{r}^{max}_{\mathrm{p},1}\right] \rightarrow \mathbb{R} \\ \mathtt{r} \rightarrow f_\mathrm{p}(\mathtt{r} - \mathtt{r}^{max}_{\mathrm{p},1}) - f_\mathrm{p}(\mathtt{r}^{max}_{\mathrm{p},1}) \end{array} \right.$$

$$\forall i > 1, \quad g_{\mathrm{p},i} : \left| \begin{array}{l} \left[0\dots\mathtt{r}^{max}_{\mathrm{p},i}\right] \rightarrow \mathbb{R} \\ \mathtt{r} \rightarrow f_{\mathrm{p},i}(\mathtt{r}) \end{array} \right.$$

$$\forall i \geq 1, \quad f_{\mathrm{p},i+1} : \left| \begin{array}{l} \left[0\dots\mathtt{r}^{max}_{\mathrm{p}} - \Sigma^{i}_{j=1}\mathtt{r}^{max}_{\mathrm{p},j}\right] \rightarrow \mathbb{R} \\ \mathtt{r} \rightarrow f_{\mathrm{p},i}(\mathtt{r} - \mathtt{r}^{max}_{\mathrm{p},i}) - f_{\mathrm{p},i}(\mathtt{r}^{max}_{\mathrm{p},i}) \end{array} \right. \qquad (7)$$

Figure 5 illustrates the decomposition method for one PRU performance profile f_p.

All PRU are treated in the same way, such that we obtain a list of performance profile pieces $\{g_{\mathrm{p},i}, 1 \leq \mathrm{p} \leq \mathbf{P}, 1 \leq i \leq i_\mathrm{p}\}$. We sort this set with the best tradeoff criterion : $g_{\mathrm{p},i}(\mathtt{r}^{max}_{\mathrm{p},i})/\mathtt{r}^{max}_{\mathrm{p},i}$. When this set is sorted, the agent is ready to re-compose $V^\sim$.

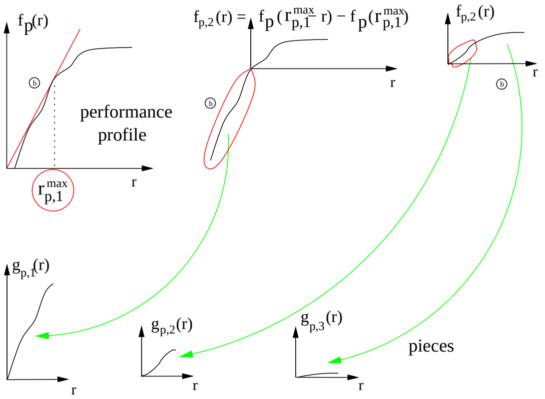

Figure 5: Decomposition.

5.2 Recomposition

When the mission changes during execution time, the agent re-composes $V^\sim$ by assembling all the pieces. The first g element is the one with the best growth, and so on. The method is depicted on Figure 6. $V^\sim$ and V^* have the same support $[0\dots\mathtt{r}^{max}]$. In order to know the expected value for a given amount $\mathtt{r}$, the agent check the value $V^\sim(\mathtt{r})$. Then, it will take its local decision according to this approximate value.

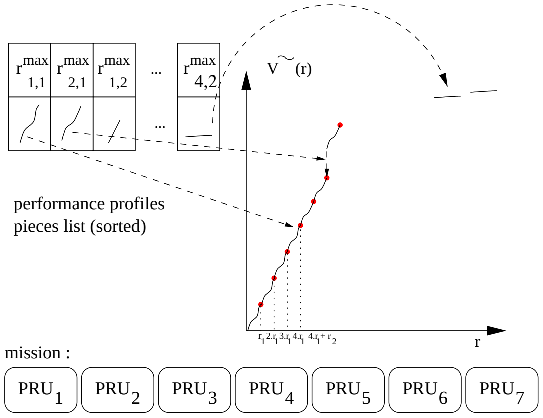

Figure 6: $V^\sim$ recomposition.

6 VALIDATION

In the previous sections, we present how to obtain an approximation of the value function. The rover will use this function in order to take decision if the mission change during execution time. Now, we must prove that :

- the time needed to obtain $V^\sim$ is short enough,

- the rover can take good decisions by using $V^{\sim}$.

Concerning the time needed to calculate $V^{\sim}$, we obtain very good results : it takes just few milliseconds. Tabular 1 give computation time for different mission sizes.

Table 1.

# PRUs	20	40	60	80	100	120	140
V^* (in s)	3	7	16	35	63	112	187
$V^{\sim}$ (in ms)	0,4	0,9	1,2	1,7	2,2	2,6	3,5

To validate our approach, we will show that the agent can take good decisions by using $V^{\sim}$ as an expected value. The decision quality evaluation is made throw the Q-value function comparison for each pair $(state, action)$ between the policies locally obtained with $V^{\sim}$ and V^*.

The validation consists in four steps :

1. we calculate both $V^{\sim}$ and V^*,
2. we calculate both policies (π_0^* and $\pi_0^{\sim}$) for the current PRU$_0$,
3. we calculate Q_0^*, the local optimal Q-value function (for PRU$_0$),
4. compare the loss of value while the agent is using $\pi_0^{\sim}$ instead of π_0^*.

Note that point 2 and 3 are done together at the same time. As the MDP for PRU$_0$ has few states, the calculation of π_0^*, Q_0^* and $\pi_0^{\sim}$ is quick. We did some experiments with mission composed of different PRU kinds and number. Figure 7 is an example of $V^{\sim}$ and V^* for 20 PRU with 4 PRU kinds.

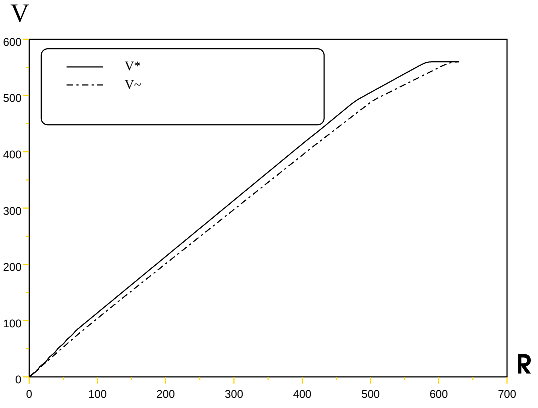

Figure 7: $V^{\sim}$ and V^*.

On this Figure 7 we see that $V^{\sim}$ is a good approximation of V^*. We have done several experiments and have obtain some good results. Q_0^* is given by the following equation.

$$Q_0^*(\langle \mathtt{r}, \mathbf{Q}, \mathtt{0}, \ell \rangle, \mathbf{E}_{\mathtt{m}}) = V_{\mathbf{E}_{\mathtt{m}}}(\langle \mathtt{r}, \mathbf{Q}, \mathtt{0}, \ell \rangle) \quad (8)$$

$$Q_0^*(\langle \mathtt{r}, \mathbf{Q}, \mathtt{0}, \ell \rangle, \mathbf{M}) = V^*(\mathtt{r}) \quad (9)$$

It makes no sense to calculate the Q-value for $\pi_0^{\sim}$. When the agent takes its decisions using $\pi_0^{\sim}$ instead of π_0^*, there may be a loss of value. The error value is given by :

$$e(\mathbf{s}) = Q_0^*(\mathbf{s}, \pi_0^*(\mathbf{s})) - Q_0^*(\mathbf{s}, \pi_0^{\sim}(\mathbf{s})) \quad (10)$$

Indeed, if for a given state $\mathbf{s}$, $\pi_0^*(\mathbf{s}) = \pi_0^{\sim}(\mathbf{s})$, then the error is zero. Otherwise, $e(\mathbf{s})$ measure the loss of value. The objective is to obtain as few error as possible. When an error exists, the better is to have a small value.

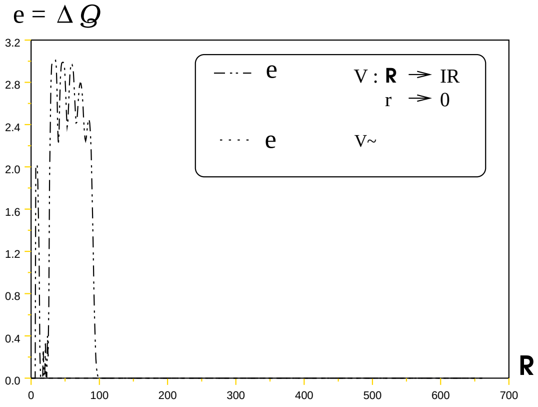

Figure 8: $V^{\sim}$ and V^*.

Figure 8 is an error measure with $V^{\sim}$ and V^* depicted on Figure 7. When we use our approximation method, there is no error. It explains why we do not see any dotted curve on the Figure. The other curve corresponds is the error measure if the agent considers that the expected value is null ($V(\mathtt{r}) = 0$). When the robot has few resources, it does not always take an optimal decision. However, when the agent follows our approximative policy, it never make mistakes.

Of course, this example seems to be chosen because it is the best. But it is not the case. We made some experiments with two different PRU kinds. The first PRU kind is composed of tightened modules : near qualities, near resource consumption. On the contrary, the second PRU kind is composed of distant modules : separate qualities, distant resource consumption.

When the agent calculates $\pi_0^{\sim}$ on a PRU of the first type, there are lots of errors. Indeed, the modules are almost the same, so the Q-value difference between two actions is low. Then, it is very difficult for the agent to take the right decision. However, in this case, the error is also low. When the agent chooses a module that is near of the optimal one, it does not make the right decision, but this decision is good.

When the agent calculates $\pi_0^\sim$ on a PRU of the second type, there are few errors. For a given state, Q-values are separate because the module qualities are separate. Then, most of the time $\pi_0^\sim$ is equal to π_0^*, the robot chooses the right decision.

To conclude, if modules are clearly separate, the robot takes the right decision. If modules are near, the robot takes good decisions, with a low Q-value error level.

6.1 Limits of this Approximation Method

The resource consumption probability distribution follows a normal distribution law in all the modules we use for our experiments. Most of the time, it represents modules that can be found a real application. We have also tried to make some experiments with modules in which the resource consumption probability distribution is not normal. We made a risky module $\mathtt{m}$: the resource consumption is 4 or 16, with $\mathcal{P}r(4|\mathtt{m}) = 0.5$ and $\mathcal{P}r(16|\mathtt{m}) = 0.5$. In this case the module can only consume 4 or 16 units, but not 8. In this kind of particular case V^* is not smooth. As a result, $V^\sim$ is not a good approximation. Then, $\pi_0^\sim$ and π_0^* are different, there is a lot of error. But this case does not represent a realistic resource consumption module.

7 CONCLUSIONS

Resource consumption is crucial for an autonomous rover. Here, this rover has to cope with limited resources to executed a mission composed of hierarchical tasks. These tasks are progressive processing units (PRU). It is possible to compute an optimal resource control for the entire mission by modelling it with an MDP. In the case where the mission changes at execution time, the rover has to recompute online a new global policy. We propose a way to quickly compute an approximate value function that can be used to calculate a local policy on the current PRU. MDP Decomposition and value function approximation techniques are used to calculate $V^\sim$. We have shown in the last section that the agent takes good decisions when it uses $V^\sim$ to compute its local policy $\pi_0^\sim$. In a near future, we intend to complete our demonstration on real robots by considering dynamic situations where missions can change online.

REFERENCES

Arnt, A., Zilberstein, S., Allan, J., and Mouaddib, A. (2004). Dynamic composition of information retrieval techniques. *Journal of Intelligent Information Systems*, 23(1):67–97.

Dean, T. and Lin, S. H. (1995). Decomposition techniques for planning in stochastic domains. In *proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1121–1127, Montreal, Quebec, Canada.

Feng, Z., Dearden, R., Meuleau, N., and Washington, R. (2004). Dynamic programming for structured continuous markov decision problems. In *proceedings of UAI 2004*, pages 154–161.

Meuleau, N., Hauskrecht, M., Kim, K., Peshkin, L., Kealbling, L., Dean, T., and Boutilier, C. (1998). Solving very large weakly coupled markov decision processes. In *proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, Madison, WI*.

Mouaddib, A. I. and Zilberstein, S. (1998). Optimal scheduling for dynamic progressive processing. In *proceedings of ECAI*, pages 499–503.

Parr, R. (1998). Flexible decomposition algorithms for weakly coupled markov decision process. In *proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, Madison, WI*.

Pineau, J., Gordon, G. J., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for pomdps. In *proceedings of the 18th International Joint Conference on Artificial Intelligence IJCAI*, pages 1025–1032.

Zilberstein, S., Washington, R., Berstein, D., and Mouaddib, A. (2002). Decision-theoretic control of planetary rovers. *LNAI*, 2466(1):270–289.