Meta-Level Control Under Uncertainty for Handling Multiple Consumable Resources of Robots

Simon Le Gloannec, Abdel-Illah Mouaddib, François Charpillet

Meta-Level Control Under Uncertainty for Handling Multiple Consumable Resources of Robots

Simon Le Gloannec and Abdel-Illah Mouaddib

GREYC-Université de Caen
Campus II
BP 5186 F-14032 Caen Cedex
{*slegloan, mouaddib*}*@info.unicaen.fr*

François Charpillet

LORIA
BP 239
F-54506 Vandœuvre-lès-Nancy, France
charp@loria.fr

***Index Terms*— Supervisory Control - Agents and Agent Based Systems - Behavioral Robotics - Mobile Robotics**

***Abstract*— Most of works on planning under uncertainty in AI assumes rather simple action models, which do not consider multiple resources. This assumption is not reasonable for many applications such as planetary rovers or robotics which much cope with uncertainty about the duration of tasks, the energy, and the data storage. In this paper, we outline an approach to control the operation of an autonomous rover which operates under multiple resource constraints. We consider a directed acyclic graph of progressive processing tasks with multiple resources, for which an optimal policy is obtained by solving a corresponding Markov Decision Process (MDP). Computing an optimal policy for an MDP with multiple resources makes the search space large. We cannot calculate this optimal policy at run-time. The approach developed in this paper overcomes this difficulty by combining: decomposition of a large MDP into smaller ones, compression of the state space by exploiting characteristics of the multiple resources constraint, construction of local policies for the decomposed MDPs using state space discretization and resource compression, and recomposition of the local policies to obtain a near optimal global policy. Finally, we present first experimental results showing the feasibility and performances of our approach.**

I. Introduction

There has been considerable work in AI on planning under uncertainty. However, this work generally assumes an extremely simple model of action that does not consider continuous time and multiple resources[1]. These assumptions are not reasonable for many application domains such as space mission and planetary rovers which much cope with uncertainty about the duration of tasks, the energy required, the data storage necessary and limited communication capacity. Limited communication capacity combined with multiple resource constraints require that the remote spacecraft or planetary rover operates autonomously. The need of autonomy and robustness in the face of uncertainty will grow as rovers become more capable and as missions explore more distant planets.

Planning systems that have been developed for planetary rovers and similar applications typically use a deterministic model of the environment and action effects. Such a planning system produces a deterministic sequence of actions to achieve a set of tasks under nominal conditions. These current planning systems, which rely on re-planning to handle uncertainty , are myopic and do not model the uncertainty in the planetary applications. As the mission complexity (the set of tasks grows) and communication constraints grow, the weakness of these approaches will become critical.

Decision Theory is a framework for reasoning under uncertainty, rewards, and costs. This framework allows to find a tradeoff between uncertainty on the multiple resources consumption, the value gained when achieving a goal and the cost of consuming resources. This framework combined with the progressive processing that allows a rover to trade off execution resources against the quality of the result similar to resource-bounded reasoning provides a suitable framework. The objective of this paper is to complete previous decision-theoretic approaches on controlling progressive processing to overcome new requirements of the rover applications that we illustrate by an example of plans in the next section. These plans present several new requirements that have not been previously addressed:

• **Task inter-dependency:** Task execution may depend on the outcome of previous actions.

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

The contribution of this paper is twofold. First, we generalize previous decision-theoretic control techniques [2], [3] to handle multiple resources under uncertainty by using a Markov Decision Process using multidimensional utility and value functions. Second, we examine the effect of increasing the size of plans (the acyclic graph) on the MDP. We address the problem of the large size of the MDP by using classical techniques of decomposition of the large MDP into smaller ones that are easy to solve and then recompose the local policies to obtain an optimal or near optimal global policy. The remaining of the paper describes these different steps of our approach that consisting of formalizing the problem of planning under multiple resources constraints with a Markov Decision Process with multidimensional value and utility functions, examining the complexity of solving the obtained MDP to construct an optimal policy and then tackling this difficulty : (1) decomposing a large MDP into smaller ones

[4], (2) compressing the states of multiple variables that are not significantly different, that is, that have the same transition probabilities to other states, and thus the same expected return, under the optimal policy, (3) constructing local policies of small MDPs under the model of discretization of state space of multiple resources using a minimal partition of the state multiple resources using point 2 and (4) re-composing local policies to obtain a near optimal global policy [5]. The rest of the paper describes in detail all these steps.

II. Progressive task planning

A. General definitions

Definition 1: An **exploration graph** $\mathcal{G}$ is a directed acyclic graph of P progressive processing units $PRU_1, PRU_2, \ldots, PRU_P$.
in the rest of the paper, we assign p or PRU_p as the p^{th} PRU in the exploration graph (figure 1)

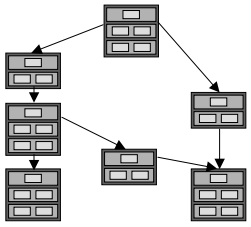

Fig. 1. An exploration graph

Definition 2: A **progressive processing unit** (PRU), PRU_p consists of a set of L_p levels, $\mathcal{L}_p = \{l_{p,1}, l_{p,2}, \ldots, l_{p,L_p}\}$.

Definition 3: A **level** $l_{p,l}$ in PRU_p consists of a set of $M_{p,l}$ modules, $\mathcal{M}_{p,l} = \{m_{p,l,1}, m_{p,l,2}, \ldots, m_{p,l,M_{p,l}}\}$.
A level corresponds to a specific task. The PRU_p in figure 4 is divided into 3 levels ($L_p = 3$), which can be for example $l_{p,1}$: start, $l_{p,2}$: take a picture and $l_{p,3}$: save the picture.

Definition 4: The **module** $m_{p,l,m}$ of the level $l_{p,l}$ in PRU_p consists of a quality $Q_{p,l,m}$ and a probability distribution $\Pi_{p,l,m}$ over the consumed resources when the module is executed
A module is a specific way to execute a level task. For example, $m_{p,2,1}$: take a low resolution picture and $m_{p,2,2}$: take a high resolution picture ($M_{p,2} = 2$) in the level $l_{p,2}$ of PRU_p in figure 4, and which means $Q_{p,2,1} < Q_{p,2,2}$. An example of module is given in figure 3.
Example :
In Figure 2 we give an example of a **progressive processing unit**. The first level consists of starting the task. At the second level, the robot takes a picture. It can choose between a high or a low resolution. Finally, it can compress the picture with two different methods. The quality of a high resolution picture is obviously better than that of a low resolution picture.

B. Extension to multiple resources

The main contribution of our work is the extension of progressive task planning to multiple resources. The problem

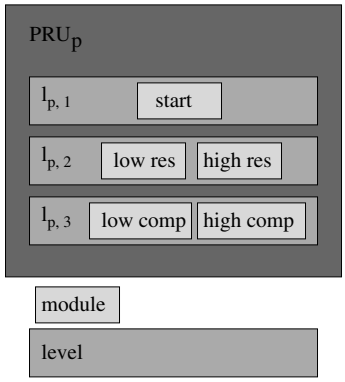

Fig. 2. A Progressive processing unit

is that we do not know exactly how many resources will be consumed by a specific task. We represent this uncertainty with a resource probability distribution. A task can for example consumes between 8 and 12 units of energy.

1) Resource dependency: In this problem, we must take into account all resources. There consumption may be sometimes dependent. We distinguish two kinds of dependency : internal and external dependencies. We can have an internal dependency when the energy consumption depends on the time consumption for a specific task. For example, the more the robot digs a hole, the more it takes time and energy. We can easily express this with a dependency mapping f such that $energy = f(time)$.

An external dependency happens when a resource consumption for a given task affects a later task. For example, if the rover takes a picture during the night, it has to use the flash, which consumes energy, but he can also wait until the day. It has to make a choice between two resources, that can have consequences for the rest of the plan.

The rover will evolve in a real environment, and resource consumption may also depend on some external factors like temperature or sunshine. For the moment, we do not take these factors into account.

2) Resource vector: We have chosen to work with resource vector, for example if we are dealing with energy and time, we denote it as $\overrightarrow{R} = \{energy, time\}$. In general, for r resources, we note $\overrightarrow{R} = \{R_1, R_2, ...R_r\}$, where R_ρ the ρ^{th} resource, $1 \leq \rho \leq r$. We assume that we know in real time the amount of remaining resource and we note it $\overrightarrow{R}_{rem}$.

Definition 5: To model the evolution of the remaining resource vector, we introduce the following notation :

$$\overrightarrow{R}_{rem} \leq \overrightarrow{R}'_{rem} \quad \Leftrightarrow \quad \forall \rho \in [1; r],\ R_\rho \leq R'_\rho \qquad (1)$$

Definition 6: It is not physically possible to have a single resource with negative value. In such situation we denote

$$\overrightarrow{R}_{rem} = \overrightarrow{R}_\emptyset \quad \Leftrightarrow \quad \exists\, \rho \in [1; r],\ R_\rho < 0 \qquad (2)$$

The resources we are dealing with, like time and energy, are continuous variables. The better for us would be to take into account this continuous dimension as long as we can. But realistically we need to use a discretization.

Definition 7: A **discrete probability distribution** of the possible consumption of the resources vector for the module $m_{p,l,m}$ is :
$\Pi_{p,l,m} = \{(P_{p,l,m,1}, \overrightarrow{R}_{p,l,m,1}), \ldots, (P_{p,l,m,X}, \overrightarrow{R}_{p,l,m,X})\}$ where X is the number of possible consumed resource vector, $\overrightarrow{R}_{p,l,m,x}$ is a consumed resources vector, and $\sum_{x=1}^{X} P_{p,l,m,x} = 1$.

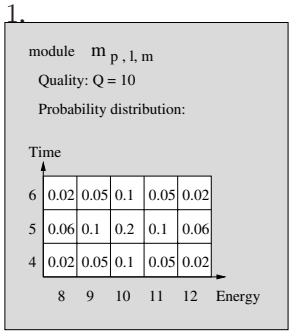

Fig. 3. A module descriptor

The module described in figure 3 has a quality of 10 and consumes two resources, energy and time. The execution of this module can for example consume 9 units of energy and 5 units of time with probability 0.1. We denote it as $(P_{p,l,m,x} = 0.1, \overrightarrow{R}_{p,l,m,x} = \{9, 5\})$. In this example $X = 3 \times 5 = 15$

C. Task selection

Definition 8: The **progressive processing control problem** is to select at run-time the set of tasks that maximizes the global utility.
When the rover has executed a module for a given level $l_{p,l}$, he has to choose between two actions :

- select a module of the next level $l_{p,l+1}$ and execute it,
- skip the remaining levels in the PRU_p to move to another $PRU_{p'}$ accessible from PRU_p in the exploration graph.

The optimal decision is the one that maximizes the global utility. The sequence of decisions will determine the set of modules to be executed. The global utility is the cumulative reward of the executed modules (reward is measured by the qualities associated to modules). Since the rover decision process only depends on the quality of the next modules and the current remaining resources, the problem of module selection respect the Markov property. We can control the rover with a Markov Decision Process (MDP).

III. Markov Decision Process Controller

A. Definitions

An MDP is a quadruplet $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, R\}$ where $\mathcal{S}$ is a set of states $\mathcal{A}$ is a set of actions, $\mathcal{T}$ is a set of transitions, R is a reward function. In the following, we define what $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, R\}$ means in our context.

Definition 9: The **accumulated quality** Q_{acc} is the sum of the previously executed module quality $Q_{p,l,m}$ in the current PRU.

Definition 10: A **state**, $s = [l_{p,l}, Q_{acc}, \overrightarrow{R}_{rem}]$, consists of the last executed step, the accumulated quality and the remaining resources vector $\overrightarrow{R}_{rem}$.

Definition 11: there are two different kinds of terminal states

- A **failure state** $[l_{p,l}, Q_{acc}, \overrightarrow{R}_{\emptyset}] = [failure, \overrightarrow{R}_{\emptyset}]$ is reached when one resource is totally consumed(see definition 6)
- A **final state** is reached when a task of a terminal PRU has been executed (a PRU with no successor in the exploration graph)

Definition 12: There are two possible **actions** (see figure 4): execute $\mathbf{E}^{m}_{p,l+1}$ and move $\mathbf{M}_{p \to p'}$. The action $\mathbf{M}_{p \to p'}$ moves the MDP to the $PRU_{p'}$. The action $\mathbf{E}^{m}_{p,l+1}$ execute the m^{th} module $m_{p,l+1,m}$ of level $l_{p,l+1}$ (if $l < L_p$).

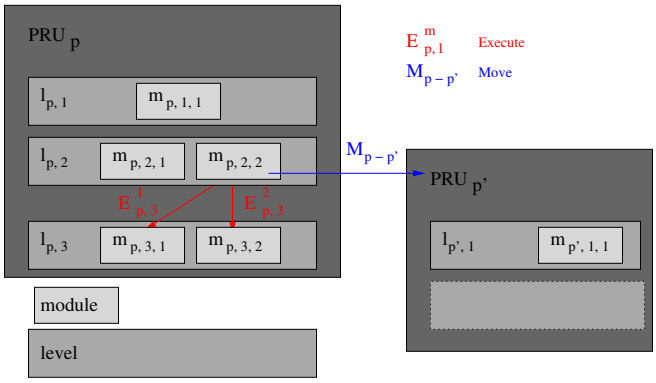

Fig. 4. 2 PRU and the two possible actions **E** and **M**

Definition 13: the **transition model** is a mapping from $\mathcal{S} \times \{\mathbf{E}, \mathbf{M}\}$ to a discrete probability distribution over $\mathcal{S}$. The move action is deterministic :

$$\Pr([l_{p',0}, 0, \overrightarrow{R}] | [l_{p,l}, Q_{acc}, \overrightarrow{R}], \mathbf{M}_{p \to p'}) = 1$$

The execution action is probabilistic, the distribution is given by the module descriptor $\Pi_{p,l,m}$ (Definition 4 and 7),
if $\forall r_\rho \in \overrightarrow{R}', r_\rho \geq 0$

$$\Pr([l_{p,l+1}, Q'_{acc}, \overrightarrow{R}'] | [l_{p,l}, Q_{acc}, \overrightarrow{R}], \mathbf{E}^{m}_{p,l+1}) = P_{l+1,m,l}$$

where $Q'_{acc} = Q_{acc} + Q_{p,l+1,m}$ and $\overrightarrow{R}' = \overrightarrow{R} - \overrightarrow{R}_{l+1,m,l}$,
else $\forall \overrightarrow{R}', \exists r_\rho \in \overrightarrow{R}', r_\rho < 0$:

$$\Pr([failure, \overrightarrow{R}'] | [l_{p,l}, Q_{acc}, \overrightarrow{R}], \mathbf{E}^{m}_{p,l+1}) = \sum_{\overrightarrow{R}' \leq \overrightarrow{R}_{\emptyset}} P_{l+1,m,x}$$

Definition 14: Rewards are associated with each state based on the quality gained by executing the most recent module $M_{p,l,m}$.

$$Rew([l_{p,l}, Q_{acc}, \overrightarrow{R}]) = \begin{cases} 0 & \text{if } l < L \\ Q_{acc} & \text{if } l = L \end{cases} \quad (3)$$

B. State's value

We adapt the Bellmann equation to our problem

$$V(s = [l_{p,l}, Q_{acc}, \overrightarrow{R}]) = Rew(s) + \max_a \sum_{s'} \Pr(s'|s, a).V(s') \quad (4)$$

$$= \max \begin{cases} Rew(s) + \max_m(a = \mathbf{E}^m_{p,l+1}) & \sum_{x=1}^{X} \Pr(s'|s, \mathbf{E}^m_{p,l+1}).V(s') \\ Rew(s) + \max_{p'}(a = \mathbf{M}_{p \to p'}) & 1.V(s'') \end{cases}$$

$$with \begin{cases} s' = [l_{p,l+1}, Q_{acc} + Q_{p,l+1,m}, \overrightarrow{R}_{rem} - \overrightarrow{R}_{p,l+1,m,x}] \\ s'' = [l_{p',1}, 0, \overrightarrow{R}] \end{cases}$$

However s' could be the state $[failure, \overrightarrow{R}]$ that does not appear in the equation because it's value is 0. For terminal states :

$$V([l_{p,L_p}, Q_{acc}, \overrightarrow{R}]) = Rew([l_{p,L_p}, Q_{acc}, \overrightarrow{R}]) \quad (5)$$

$$V([failure, \overrightarrow{R}]) = Rew([failure, \overrightarrow{R}]) = 0 \quad (6)$$

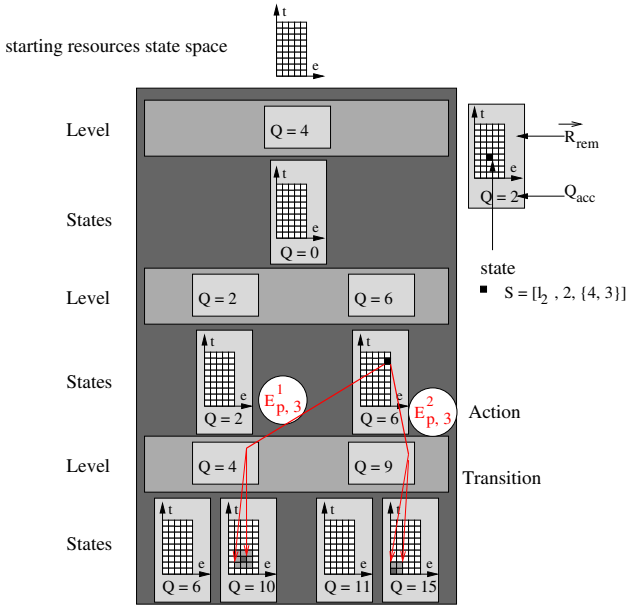

Fig. 5. State representation for a single PRU

C. Optimal policy computation

We are dealing with a finite-horizon MDP with no loops. Transitions with **E** and **M** move forwards in the state space by always incrementing level or unit number. Although there are a lot of states due to the number of resources, this kind of MDP can easily be solved because the value function is calculated in one sweep (backward chaining, beginning with terminal states); But we have two problems:

- The MDP is large, i.e there is a lot of states and transitions. Although we calculate each state value only once, the computation time increases exponentially in the number of resources with the number of PRU. Therefore we cannot calculate the global optimal policy at run-time.
- We cannot add ore remove any PRU to the exploration graph without recalculating the entire new MDP. This global approach is not adapted to our problem.

The rover has to choose his action at run-time. But it can not calculate at run time the entire MDP values : when the number of PRU increases , the number transitions corresponding to a **M**ove action increases too. We decide to approximate the MDP, to allow dynamic task selection. In the next section, we address the problem of solving this large MDP.

IV. Solving the large MDP

A. Principle

To overcome these difficulties we decompose the large MDP into smaller ones, that we later recombine to construct a nearly optimal policy. The goal of the decomposition is to avoid the calculation of the state values that are not directly accessible from the current state. We just want to focus on the states that are in the current PRU. The states in the next PRU are not evaluated at run-time, in order to avoid combinatorial explosion. We evaluate all PRU initial states before run-time. At run-time the agent evaluates the state values in the current PRU, and chooses the best action between **M** and **E**. We explain now the way we used to decompose the global MDP.

Since the transition corresponding to an action **M** is deterministic, a natural way to decompose the MDP is to calculate a local policy for each PRU(figure 6.b). We keep only the action **E** and we obtain as many local policies as there are different PRU in the graph. For one PRU, we do not store the entire state space, but only the starting level state space (see the top of figure 5), because an action **M** can only reach the starting level of the next PRU. Once all the states have been pre-evaluated, we store this starting state space and its values for each PRU(figure 6.c) in a library, before execution time.

At run-time, we dynamically recompose the MDP. The question is to decide if it is better to remain or to leave the current PRU. We examine the local policy of the current PRU, we compare the expected value for the best execution action $V_{\mathbf{E}}$ to the value for the best move action $V_{\mathbf{M}}$. Since it is not feasible to calculate $V_{\mathbf{M}}$ at run-time, we make an approximation that we denote $V_{\mathbf{M}dec}$ (for decomposition)(figure 7.c).

The remaining of this section explain how we managed to calculated $V_{\mathbf{M}dec}$. We also have problem with the number of resources, with which the starting state space we store for each PRU grow. We find a way to improve the state space storage. Finally, in section (V), we compare $V_{\mathbf{M}}$ to $V_{\mathbf{M}dec}$.

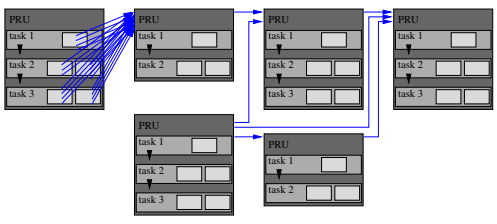

Fig. 6. Decomposition

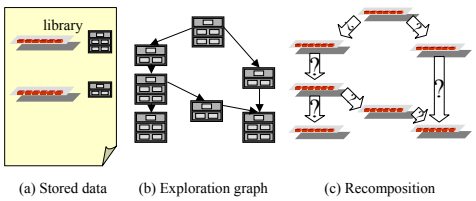

Fig. 7. Recomposition

B. Decomposition

To estimate a PRU we calculate the estimated value for all the states in its starting level (figure 8), $\mathcal{S} = \{[l_{p,0}, Q_{acc} = 0, \overrightarrow{R}_{rem}]\}$(figure 6.b).

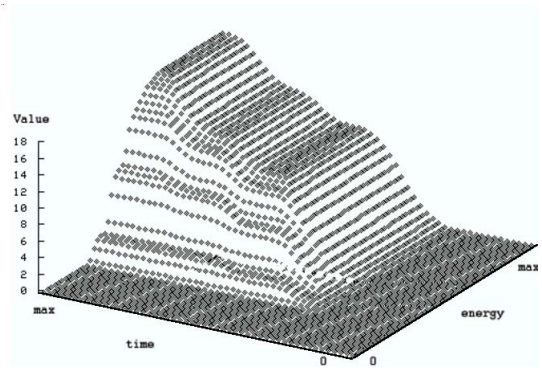

Fig. 8. Starting states values for a PRU with 2 ressources

C. State data storage and space compression

We are dealing with multiple resources and facing a new problem: how can we **store** the starting state space for each PRU (figure 6.c)? We can however quickly **find** the expected value for a given state, which is necessary for the run-time recomposition. This section explain first how the transformation $T : \mathcal{S} \rightarrow \mathcal{S}_{stored}$ works. In a second time, we explain T^{-1} : $\mathcal{S}_{stored} \rightarrow \mathcal{S}$. A starting state correspond to $s = [l_{p,0}, Q_{acc} = 0, \overrightarrow{R}]$, only $\overrightarrow{R}$ varies, so we project the starting state space on $[\overrightarrow{R}]$. This state space correspond to a r-dimensionnal cartesian product with one dimension for each resource that T tranforms into a 1-dimensionnal space (corresponding to values). Since we have a monotonic function $\forall s,\ 0 \leq V(s) \leq V_{max} = V([\overrightarrow{R}_{max}] = \{(r_{max})_1, \ldots, (r_{max})_r\})$, we project the state space on the value space (figure 9 is the projection of figure 8). For each value, we make groups of states with the same value in $\mathcal{S}_v$. And in each group we only keep the states with minimal resources in $(\mathcal{S}_v)_{min}$. Notice that there can be several states with minimal resources in a $\mathcal{S}_v$ set. Formally :

$$\mathcal{S}_v = \{s = [\overrightarrow{R}]; V(s) = v\} \quad (7)$$

$$(\mathcal{S}_v)_{min} = \{s \in \mathcal{S}_v, \nexists\, s' \in \mathcal{S}_v, \forall \rho \in [1; r],\ r'_\rho \leq r_\rho\} \quad (8)$$

and we finally store the set of groups :

$$\mathcal{S}_{stored} = \{s \in (\mathcal{S}_v)_{min},\ 0 \leq v \leq V_{max}\} \quad (9)$$

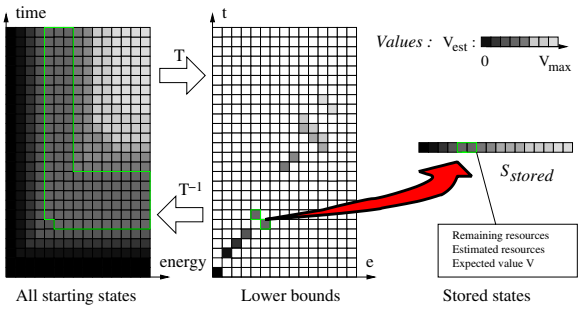

Fig. 9. multi resource data compression

The transformation T^{-1} is very simple. We want to know a state value, given $\overrightarrow{R}$. We just have to pass through the $\mathcal{S}_{stored}$ vector and stop when we reach a state with more resources. The last encountered state give us the value. We managed to transform $\mathcal{S} = \{[\{0, 0\}], ..., [\{29, 29\}]\}$(size = 900) into $\mathcal{S}_{stored}$ with a length of 20.

D. Recomposition

The goal of this paragraph is to recompose dynamically at run-time a policy that approximate the optimal one. To do it, we need to calculate the expected value $V_{\mathbf{M}_{dec}}$ for a $\mathbf{M}$ action from a state, given the exploration graph and the stored data. Therefore we have to recompose the MDP (figure 7). We calculate $V_{\mathbf{M}_{dec}}$ by maximizing the sum of the expected state values in last PRU. We first calculate each PRU depth in the exploration graph, keep for each depth the PRU with the best global expected value in a queue. Then, we sort the PRU queue, we put the best PRU to the top.

V. Experiments

A. Methods comparison

We compare the optimal policy with the policy obtained by decomposition. The advantages of the second method are the short computation time, and the small state space. However, we need to analyze how good the policy obtained by decomposition approaches the optimal policy.

To do so we calculate two error values: the mean error e_{mean}, and the decision error. The mean error indicates if the values obtained by decomposition and recomposition are close to the optimal policy. The decision error counts the times the error exceeds a fixed threshold. The measure our result in term of graph depth, so we experiment only on queues of PRU. We show in the next section that the mean error is small and that the decision error converges toward zero.

B. Error measure

We denote $\overrightarrow{R}_{max}$ as the amount of resources required to execute all tasks for the whole plan. We denote $V_{opt}(\overrightarrow{R})$ as the value obtained with the optimal policy for a state s with $\overrightarrow{R}$ remaining resources, and $V_{dec}(\overrightarrow{R})$ the value obtained with the decomposition method.

$$e(\overrightarrow{R}) = \frac{|V_{opt}(\overrightarrow{R}) - V_{dec}(\overrightarrow{R})|}{V_{opt}(\overrightarrow{R})} \quad (10)$$

The mean and the max error for $\mathcal{R} = \{\{0, \ldots, 0\}, \ldots, \overrightarrow{R}_{max}\}$ are

$$e_{mean} = \frac{\sum_{\overrightarrow{R} \in \mathcal{R}} e(\overrightarrow{R})}{Card(\mathcal{R})} \quad \text{and} \quad e_{max} = \max_{\overrightarrow{R} \in \mathcal{R}} e(\overrightarrow{R})$$

For the decision error, we take a threshold of 20%, considering that the rover could make a bad decision if the error exceeds this value.

C. Results

We first experiment on a queue with identical $PRUs$.

Graph depth	1	2	4	5	10	20	40
Mean error (%)	0	7.4	6.7	7.1	8.0	8.0	8.4
Max error (%)	0	28	28	28	28	28	28
Decision error (%)	0	1.6	3.3	2.6	1.3	0.6	0.3

We also used queues with different $PRUs$.

Graph depth	1	2	4	5	10	20	40
Mean error (%)	0	2.6	2.7	2.8	2.7	3.9	2.5
Decision error (%)	0	5.0	4.4	5.5	2.7	1.3	0.6

D. Discussion

Errors are made when the PRU queue is short, but the goal of our algorithm is to deal with very large PRU queues. For short PRU queues, we can calculate the optimal policy without any approximation.

In each case the mean error stays constant. The more resources and PRU are left, the better is our approximation. The decision error decreases toward zero. The max error is high, and it stays constant. This is the main problem of our algorithm. Errors are locally high, so we intend to search for better approximation methods to reduce them.

VI. Related work

Our work combines two aspects: Progressive tasks, and decomposition of large MDPs. For the progressive tasks, we use the work of [2] as a starting point. A first approach for an application to mobile robotics has been done in [3]. Lots of work has been developped for decomposing a large MDPs into independent subproblems [6], [7]. There are three classes of decomposition of large MDPs: firstly the *State decomposition* : the state space of the MDP is divided into subspaces to form subMDPs, so that the MDP is the union of the subMDPs [4], [7]; Secondly, *Policy decomposition*: problem solved by merging a subpolicy with others where actions, transitions, probabilities and rewards all may change when the others policies change. This makes for a more complicated composition; Thirdly, *Action decomposition*: the subMDPs are treated as "communicating" concurrent and parallel processes [5] and particular actions in one subMDP are not affected by policies used to solve other subMDPs.

VII. Conclusion and perspectives

We have presented a solution to the problem of planning under uncertainty with multiple resources. This approach relies on solving a corresponding MDP, generalizes earlier work on MDP controller [2], [3]; it permits for the first time to deal with multiple resources. Moreover, our approach addresses effectively the problem of limited amount of memory required for creating and solving the large obtained MDP and storing the resulting policy. Using decomposition of the large MDP into smaller ones, we managed to reduce the size of MDPs to create and to solve, using a compression data technique, we managed to store the resulting policies and using re-composition approach based on an approximation technique of value functions, we managed the construction of a global policy with a small loss decision quality. With such a decomposition method, we can add new PRU to the exploration graph. This approach allow an effective method to increase autonomy and robustness of robots.

References

[1] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington, "Planning under continuous time and resource uncertainty : A challenge for ai," in *UAI*, 2002.

[2] A.-I. Mouaddib and S. Zilberstein, "Optimal scheduling for dynamic progressive processing," in *ECAI-98*, 1998, pp. 499–503.

[3] S. Cardon, A. Mouaddib, S. Zilberstein, and R. Washington, "Adaptive control of acyclic progressive processing task structures," in *IJCAI*, 2001, pp. 701–706.

[4] R. Parr, "Flexible decomposition algorithms for weakly coupled markov decision process," in *UAI-00*, 2000.

[5] N. Meuleau, M. Hauskrecht, K. Kim, L. Peshkin, L. Kealbling, T. Dean, and C. Boutilier, "Solving very large weakly coupled markov decision processes," in *UAI-98*, 1998.

[6] C. Boutilier, R. Brafman, and C. Geib, "Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning," in *IJCAI 97*, 1997.

[7] T. Dean and S.Lin, "Decomposition techniques for planning in stochastic domains," in *IJCAI-95*, 1995, pp. 1121–1127.