HAL
archives-ouvertes.fr

A Decision-Theoretic Scheduling of Resource-Bounded Agents in Dynamic Environments

Simon Le Gloannec, Abdel-Illah Mouaddib, François Charpillet

▶ To cite this version:

Simon Le Gloannec, Abdel-Illah Mouaddib, François Charpillet. A Decision-Theoretic Scheduling of Resource-Bounded Agents in Dynamic Environments. International Conference on Automated Planning and Scheduling - ICAPS 2005, Jul 2005, Monterey, California/USA, United States. inria-00000414

HAL Id: inria-00000414
https://hal.inria.fr/inria-00000414

Submitted on 10 Oct 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Decision-Theoretic Scheduling of Resource-Bounded Agents in Dynamic Environments

Simon Le Gloannec, Abdel-Illah Mouaddib
GREYC-CNRS
Bd Marechal Juin, Campus II, BP 5186
14032 Caen cedex, France
{slegloan, mouaddib}@info.unicaen.fr

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

Introduction

Markov Decision Processes (MDPs) provide a good, robust formal framework for modeling a large variety of stochastic planning and decision problems, but they are unsuitable to realistic problems, in particular, to solve problems in rapidly changing environments. The existing approaches use Markov decision processes to produce a policy of execution for a static set of tasks; in a changing environment (i.e. with an evolving set of tasks), a complete computation of the optimal policy is necessary. We consider a queue of tasks that can change on-line. Potential application of this approach would be an adaptive retrieval information engine (Arnt *et al.* 2004).Our main claim is that it is possible to dynamically compute good decisions without completely calculating the optimal policy. Similar approaches have been developed to deal with MDPs with large state spaces using different decomposition techniques (Boutilier, Brafman, & Geib 1997; Parr 2000). A similar dynamic resource allocation problem has been developed in (Meuleau *et al.* 1998). A non dynamic approach has been developed for robots in (Schwarzfischer 2003).

It has been shown in (Mouaddib & Zilberstein 1998) that it is possible to find an optimal policy for this kind of problem. This optimal solution suffers from a lack of flexibilty to cope with changes in a dynamic environment. We develop an approach which provides more flexibility for MDPs to deal with dynamic environments. This approach consists of two steps. The first step consists of an off-line preprocessing of tasks and the compilation of policies for all possible available resources. The second step is a quick online approximation of the policy of executing the current task given the current state of the queue.

Problem Statement

Description

We consider an autonomous agent which has the capability of performing different kinds of stochastic tasks $T_1, T_2, \ldots, T_t$ progressively. A **progressive task** T is executed stage by stage, and it can be stopped after any stage. A quality is associated with each stage of task T. The duration Δr of execution of each stage is uncertain. The utility of a task is the sum of the quality of the executed stages. This formalism is described (Mouaddib & Zilberstein 1998) in details. For each type of task T_n we compute a **task performance profile** $f_n(r)$ that corresponds to its local expected value(see Figure 4). It represents what the agent should expect to gain if a certain task is being accomplished with a quantity r of resources.

Given an ordered list composed by several instances of these progressive tasks (for example $L=\{T_3, T_1, T_3, T_1, T_1, T_2, T_1, T_3, T_2, T_3, T_2\}$), the agent must perform some of them before a fixed deadline D to maximize the global utility.The queue of tasks evolves **dynamically**, during on-line phase. Tasks can appear or disappear everywhere (see Figure 1). Our goal then is to

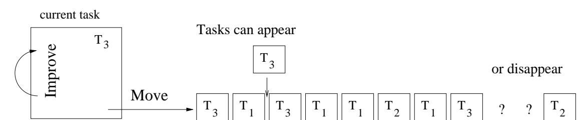

Figure 1: Dynamic changes in the task list

find an effective way to decide quickly if it is desirable to continue with the current task or to change to another one. This local decision depends on the prediction of the **expected value** of the remaining tasks in the list.

The decision problem We are facing to a resource allocation problem: we consider time as a **resource** r, which is **limited** by D. The duration of execution of each task is **uncertain**, resource consumption is **uncertain**. The agent has to allocate some resources for the current task, and some for the remaining tasks. Differently speaking it must **decide** if it is preferable to continue its work on the current task, or to give it up and switch to another task, by taking into account the state of current task, the list of remaining tasks, and the avalaible resources. Then, our resources allocation problem becomes a decision problem.

We present the formalism we use to solve our problem in a non-dynamic environment in the next paragraph, and in the next section we will explain how to cope with changes in the task list.

An MDP controller

At each step, the agent has to make a decision about continuing the improvement (**I**mprove) of the current task or

Copyright © 2005, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

abandoning it in favor of the next task (**M**ove). This decision depends on the current state of the decision process. In other words, it depends on the current state and the available resources. Consequently, the decision process respects the Markov property. We could deploy an MDP for the whole list of tasks, but we do not. The basic idea is to compute the policy of executing the current task. This policy $\Pi_{T,L}$ depends on L, the list of remaining tasks, and T the current task. As long as L remains unchanged, the agent follows $\Pi_{T,L}$. If L changes, it updates the current policy of the current task to $\Pi_{T,L_{new}}$. When the agent moves to a new task T_{new}, it assesses the current list of remaining tasks L_{new} and then it derives a new policy $\Pi_{T_{new},L_{new}}$.

Formal Framework An MDP is a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, Rew\}$ where $\mathcal{S}$ is a set of states representing the amount of remaining resources r, Initially, $s_0 = [D]$ where D is the deadline. Terminal states represent all the situations where r has been fully elapsed $s = [r < 0]$. $\mathcal{A}$ is a set of actions $\{\mathbf{M}ove, \mathbf{I}mprove\}$ (see Figure 3). The **M**ove action is deterministic: the agent moves from the current task and examines the next one in the dynamic list. The second action is stochastic, **I**mprove consists in spending a certain quantity of resources Δr in the next stage. $\mathcal{T}$ is a set of transitions, Rew is a reward function. In the following, we define what $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, Rew\}$ means in our context. The states represent resources remaining, the actions are **I**mprove and **M**ove, and the reward is the accumulated quality for all improvements previously achieved in the task. The transitions are given in the description of each progressive task type.

$$\Pr(s' = [r - \Delta r] | s = [r], \mathbf{I}) = P_{Stage}(\Delta r) \quad (1)$$

$$\Pr(s' = [r < 0] | s = [r], \mathbf{I}) = \sum_{\Delta r > r} P_{Stage}(\Delta r) \quad (2)$$

Policy When the agent starts to execute a task T, it computes a local policy $\Pi_{T,L}$ where L represents the list of remaining tasks. Since the list L remains unchanged, it follows its policy $\Pi_{T,L}$ (see Figure 2). As soon as L changes to L_{new}, the agent has to change the local policy $\Pi_{T,L}$ to $\Pi'_{T,L_{new}}$. To do so, the computation of $\Pi_{T,L_{new}}$ is based

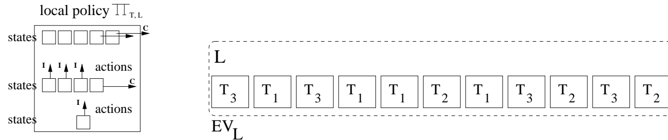

Figure 2: A local policy

on a quick computation of the new expected values of actions **M** and **I**. We describe below how those functions are computed.

Value function In order to compute the policy, we use a value function $V(s)$ based on the Bellman equation:

$$\begin{cases} V(s = [r \geq 0]) = Rew(s) + \max_{\mathbf{A}} \sum_{s'} \Pr(s'|s, \mathbf{A}).V(s') \\ V(s = [r < 0]) = 0. \end{cases} \quad (3)$$

Policy $\Pi_{T,L}$ is local, therefore we estimate the value of the states after a **M**ove action with a second value function that we denote as EV. EV_L is the value the agent can expect to gain if it accomplishes tasks in L with r resources. $V_{T,L}$ is the expected value of achieving task T taking into account tasks in L. In our context, the Bellman equation becomes: $V_{T,L}(s = [r]) = Rew(s) +$

$$\max_{\mathbf{A}} \begin{cases} EV_L(s = [r]) & \text{if } \mathbf{A} = \mathbf{M} \\ \sum_{s' = [r \geq 0]} \Pr(s'|s, \mathbf{I}).(V_{T,L}(s')) & \text{if } \mathbf{A} = \mathbf{I} \end{cases} \quad (4)$$

where $s' = [r - \Delta r]$ represents a possible state after the improvement. Equation 4 needs to be solved as soon as the queue L changes to L_{new}. This leads to a computation of $EV_{L_{new}}$ and $V_{T,L_{new}}$. We compute first $EV_{L_{new}}$. Then, we obtain the new local policy $\Pi'_{T,L_{new}}$ by computing the $V_{T,L_{new}}$ function using a backward-chaining algorithm on all the states in T. Figure 3 represents the mechanism of action selection. In the current state the agent has 176 resources left. The dotted rectangles represent the remaining tasks in the queue L. Now, the problem is to compute the

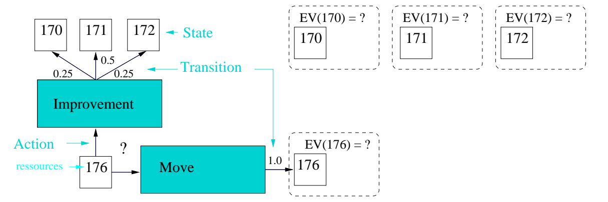

Figure 3: Action selection

Expected Value EV_L. It is possible to compute the exact EV_L of an optimal policy $\Pi^*_{T,L}$. We just have to use the Bellman equation on all the possible future states in L with a backward chaining algorithm. But with this method, the generated MDP is very large. But if we add a task in the middle of this linear graph, we must re-deploy the Bellman equation. This solution is not convenient if tasks are often inserted in the list i.e. if the environment changes. We can still keep the MDP model for the local task, but we must find another way to compute EV_L. We must evaluate the Expected Value for the rest of the plan quickly. Therefore we have to sacrifice optimality.

Rather than computing EV_L using a global MDP, we divide this process into two phases. An off-line phase, where we compile performances profiles for each type of task (see Figure 4), and an on-line phase, where we recompose dynamically the EV_L function (see Figure 5) when it is necessary.

In fact, we are faced with an MDP decomposition problem. We create a local MDP for each type of tasks, and we compute policies for local $MDPs$ $\Pi_{T,\emptyset}$ (policy of executing T assuming $L = \emptyset$). The local policies are represented by performance profiles. We recombine them on-line to obtain an approximate $EV_L \simeq recomposition_{T' \in L}(V_{\Pi_{T',L}})$.

Task performance profile construction The first phase consists of the computation and the storage of the performance profile function for each task type. The performance profile function $f_n(r)$ corresponds to the exact expected value if we have r resources to spent in the task of type

T_n.We consider that this task is independent from the others, and we compute a policy for a local MDP, without the **M**ove action. This means that we assume $L = \emptyset$. Differently speaking, $f_n(r) = V_{\Pi_{T',\emptyset}}([r])$. This computation is quick. The local MDP has few states, and each state is evaluated once. It only depends on the number of stages in the task, and also on the maximum amount of resources that the agent can spent in the tasks. Now, we have to combine all these functions f_n to find an EV_L function for a list composed of tasks of type $\{T_1, T_2, \ldots, T_n\}$. This function must be a good approximation of the exact EV_L.

Dynamic operation

Principle

This section is divided into two phases: the off-line phase, where we compute the performance profile $f_n(r) = V_{\Pi_{T',L}}([r])$ for each task, and the on-line phase, where the agent recomposes the expected value function EV_L.

Off-line: The task pre-processing The performance profile functions $f_n(r)$ of each task are increasing functions in r. Note that the more time the agent spends on a task, the higher the expected reward will be. These curves are also bounded by a maximum, which corresponds to the reward if the task is completely achieved. The curves give us several information. On the one hand we know the expected value for this task, while on the other, we know the quality cost ratio, i.e. $f_n(r)/r$. The first phase of the pre-processing consists of finding the best quality cost ratio among all tasks.

$$max_{n,r} f_n(r)/r \tag{5}$$

Then we isolate the part of the curve which precedes this point, as figure 4 shows. We start again to seek the second best quality cost ratio, until all the curves are processed. We finally store the points and slices of curves $c_{i,j}$ in a table R_T (cf algorithm 1). $c_{i,j}$ is the jth slice of curve f_i.

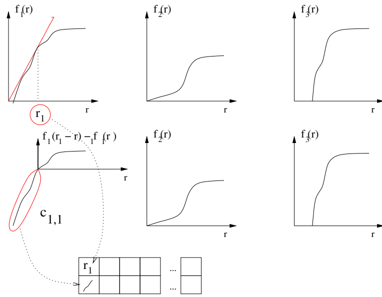

Figure 4: Finding the best ratio quality/cost

On-line: Expected Value function reconstitution It is the second phase. All the next computations are done at run time. We recompose the EV_L function with all the information that we stored during the off-line phase $\{f_1, \ldots, f_t\}$, R_T, and with the list. The method is simple (see Algorithm 2). We have a list of tasks = $\{T_3, T_1, T_3, T_1, T_1, T_2, T_1, T_3, T_2, T_3, T_2\}$, and we recompute the expected value for all possible values of r, which represents the remaining time at a given moment. Algorithm 2 is illustrated in figure 5. Basically, we want that

Algorithm 1 Task pre-processing

Require: $\{f_1; \ldots; f_t\}, R_T = \emptyset$
1: **while** $\forall i \leq t, \exists r, f_{T_i}(r) > 0$ **do**
2: $r'_{t'}, f_{t'} = max_{i,r} f_i(r)/r$
3: $R_T \leftarrow R_T \cup \{[r'_{t'}, f_{t'}(r \leq r'_{t'}]\}$
4: $f_{t'}(r) = f_{t'}(r - r'_{t'}) - f_{t'}(r'_{t'})$
5: **end while**
Ensure: R_T

the agent maximizes its future rewards, i.e. its EV_L. Thus, the EV_L curve is recomposed incrementally by adding each slice of curve in R_T until all available resources has been fully elapsed. This problem is similar to the knapsack problem. We start to add the slice of curve that maximize the ratio quality/cost. We add as many slices as the number of tasks of that type in L allowed by the remaining resources. Then, we add the slice of curve that corresponds to the second ratio quality/cost in R_T. This processing continues until all slices of curve $c_{i,j}$ in R_T have been added or the resources have been fully elapsed. In the example L contains four instances of the task T_1. In the previous section, i.e during the off-line phase, we found that the first part of the task of type T_1 gives a maximum quality cost ratio for r_1 resource. Consequently, if we have only r_1 resources left, it will be better to spent them on a task T_1. Thus, we start by adding one time the corresponding slice of the curve $f_1(0 \leq r \leq r_1)$. We are not sure that the exact EV_L corresponds to the slice of curve we add, it is just an approximation. But we are sure that we can not expect[1] less than this slice of curve, i.e. the approximation we make is a lower bound for the exact expected value function. If it has r resources $r_1 \leq r \leq 2 \times r_1$, it can do the first part of a task T_1, and starts the second task T_1. Therefore, we add the same slice of curve on the top of the first one (see Figure 5). In our example, we have four tasks of type T_1 in the list, so if we

Algorithm 2 Reconstitution of the Expected Value function (approximation) EV_L

Require: R_T, L
1: $threshhold = 0$
2: $r = 0$
3: **while** $R_T \neq \emptyset$ **do**
4: $(r', f_{t'}) \leftarrow first(R_T)$
5: remove $first(R_T)$ from R_T
6: $n =$ number of task $T_t \in L$
7: **for** $i = r$, $i \leq n \times r', i^{++}$ **do**
8: $m = i \; mod \; r'$
9: $q = i \; div \; r'$
10: $EV_L(i) = m \times f_{t'}(r') + f_{t'}(q) + threshhold$
11: **end for**
12: $r = r + n \times r'$
13: $threshhold = EV_L(r)$
14: **end while**
Ensure: EV_L

[1]Note that it is an expectation. During the execution of this task we can have less reward than what we expected.

have more than $r > 4 \times r_1$ resources, the agent can expect to start four times T_1. With the rest $(r - 4 * r_1)$ it can expect to start another task. In R_T, T_2 is the best task to start after T_1. We add the corresponding slice of curve to EV_L between $4 \times r_1$ and $4 \times r_1 + r_2$. We continue adding slices of curve while R_T is not empty. If the agent has more than R_{max} resources then it is certain to achieve all the tasks, with all their improvements. The approximation of EV_L is finished.

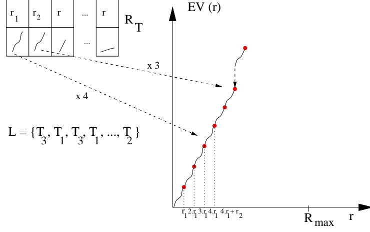

Figure 5: Reconstition of $EV_L(r)$

How do we use this function in practice

The dynamic on-line approximation In the preceding section, we explained how to reconstitute the expected value. However in practice we do not need to know all the values of $EV_L(r)$. Let D -r_{now} be the resources remaining to complete the mission. Moreover, we can use only a maximum amount of resources r_{max} for the current task. In fact, we just compute EV_L for all r in $[D$ -r_{now} -r_{max}, D -$r_{now}]$. We made this reconstitution in order to compare $EV_L(r)$ obtained by our approach with that obtained while solving the whole MDP for all the remaining tasks in the list. The complexity of computation is proportional to the number of elements in R_T. Thus, the computation of $EV_L(r)$ is quick and is more suitable to the dynamicity.

Analysis

In this section, we make a comparison between EV_{exact}, EV_{dyna}, and $EV_{pw-linear}$. This comparison concerns the time needed to compute each of them and the error made by our approach.

Complexity Comparison To compute EV_{exact} we develop a set of states $\mathcal{S} = \{s = [r, imp, T]\}$, r are the remaining resources, imp the quality of the last improvement made, and T is the task in the queue. r is considered discreet. We use a backward chaining algorithm using the Bellman equation, therefore each state is evaluated only once. The complexity of EV_{exact} is linear in the number of states $\#\mathcal{S}$. But the state space is huge. $m(EV_{exact}) = \#\mathcal{S} = R_{max} \times \prod_{T_i \in L} \#imp_{T_i}$. m is a mesure of complexity. $m(EV_{dyna}) = \sum_{R_T} \#c_{i,j} \times \#T_i$ where $c_{i,j}$ is a slice of the T_i performance profile curve, and $\#T_i$ the number of times where T_i appears in L. $m(EV_{dyna})$ is linear in the size of R_T and L. There are much fewer slices in R_T than there are states in $\mathcal{S}$. Thus, $m(EV_{dyna}) \leq m(EV_{exact})$. For a queue of 100 tasks, EV_{exact} takes several minutes, and EV_{dyna} takes less than one second. More results can be found on *http://users.info.unicaen.fr/~slegloan/thesis/*.

Value Comparison Our approximation EV_{dyna} is just a lower bound of the EV_{exact} function. If all the tasks executions where deterministic, EV_{dyna} and EV_{exact} would be equal on some specific points, at the "end" of each slice (fig 6).

$$\forall r, EV_{dyna}(r) \leq EV_{exact}(r). \tag{6}$$

Unfortunatelly, our approximation is just a lower bound

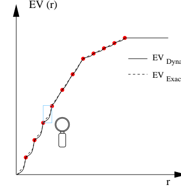

Figure 6: Deterministic Tasks

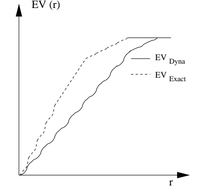

Figure 7: With Uncertainty

of what the agent should expect to gain in the worst case. We intend to mesure the difference between EV_{dyna} and EV_{exact} in the future, in order to fill the gap betwenn the two curves.

Conclusion and future works

We have presented a robust solution that copes with uncertainty due to dynamicity of environment in stochastic planning problems. The Markovian approach allows us to cope with uncertainty. The progressive approach allows us to adapt the decision in a dynamic environment. Our work combines two approaches: Progressive tasks and decomposition of large MDPs. For the future, we intend to add multiple limited resources, like energy. We would also like to extend this approach to include more specific spatial and temporal constraints. This model can be adapted to a large set of problems in a dynamic and uncertain environment like robotic applications, and information retrieval agent.

References

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

Boutilier, C.; Brafman, R.; and Geib, C. 1997. Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *IJCAI 97*, 1156–1163.

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 *UAI-98*.

Mouaddib, A., and Zilberstein, S. 1998. Optimal scheduling for dynamic progressive processing. In *ECAI-98*.

Parr, R. 2000. Flexible decomposition algorithms for weakly coupled markov decision process. In *UAI-00*.

Schwarzfischer, T. 2003. Quality and utility - towards a generalization of deadline and anytime scheduling. In *ICAPS 2003*, 277–286.