An Optimal Execution of Autonomous Agents with Plans Under Time and Resource Uncertainty

Aurélie Beynier, Abdel-Illah Mouaddib
Bd Marechal Juin, Campus II
BP5186
14032 Caen Cedex, France
{abeynier, mouaddib}@info.unicaen.fr

Rich Washington[*]
NASA Ames Research Center
Moffett Field, 94035-1000
USA
richw@email.arc.nasa.gov

Abstract

In this paper we develop an approach for optimal execution of plans under time and resource uncertainty. This line of research has been described as a challenge of AI in [5]. The authors claim that existing methods fail because they suffer from many limitations where some of them consist of: (1) they only handle simple time constraints, (2) they assume a simple model of uncertainty concerning action durations and resource consumption. In many domains, such as space applications (rovers, satellites), these assumptions are not valid. We present an approach that relaxes those assumptions by considering the following: (1) a mission is an acyclic graph of tasks that leads to complex dependencies between tasks, (2) a task has a temporal window during which it can be executed, (3) there is uncertainty about the durations and resource consumption of tasks. This class of problems is found in some scenarios of rover domains where the objective of the rover is to maximize the overall value of the mission. For that, we present a Markov Decision Process Agent taking into consideration uncertainty on temporal intervals execution and resource consumption. We describe some experimental results and the scalability of the approach.

1. Introduction

In this paper we develop an approach of planning under time and resource uncertainty along the research lines described in a challenge paper of [5]. In that challenge paper, authors claim that existing methods fail because they suffer from many limitations where some of them consist of: (1) they only handle simple time constraints, (2) they assume a simple model of uncertainty concerning action durations and resource consumption. In many domains, such as space applications (rovers, satellites), these assumptions are not valid.We present an approach that relaxes those assumptions by using a more complex model of tasks with complex dependencies in the sense that execution time and resource consumption are uncertain. In this approach, we consider the following: (1) a mission is an acyclic graph of tasks that leads to complex dependencies between tasks, (2) a task has a temporal interval during which it can be executed, (3) there is uncertainty about the durations and resource consumption of tasks. This class of problems is found in some scenarios of rover domains where the objective of the rover is to maximize the overall value of the mission. Given the graph of the mission, as is the case in space applications, we want the mission to be executed optimally by an autonomous agent.

The major objectives of the space future missions [8, 5] consist of maximizing science return and enabling certain types of science activities by using a robust approach. To show the significance of problems we deal with, let us give some examples of robotic vehicle domains where we express different constraints we consider in this paper:

• *Time windows*: a number of tasks of robotic vehicles need to be done at particular, but approximate times: for example, “about noon,” “at sunrise,” “at sunset.” There is no explicit time window, but we can represent these using time windows or soft constraints on a precise time. The examples involve measurements of the environment – a ”gravity wave” experiment that needs to be done ”preferably in the morning”, and atmospheric measurements at sunrise, sunset, and noon (look at the sun through the atmosphere). Another constraint is that the rover will be power-constrained, so much of a solar-powered rover’s operations will be in the window 10:00-15:00 to make sure that there is enough sunlight to operate. Certainly driving will happen during that part of the day. Communication has also to start at a particular time, since that requires synchronization with Earth. There are operations that cannot be done outside of a time window – not enough solar energy, for exam-

[*] NASA contractor with RIACS

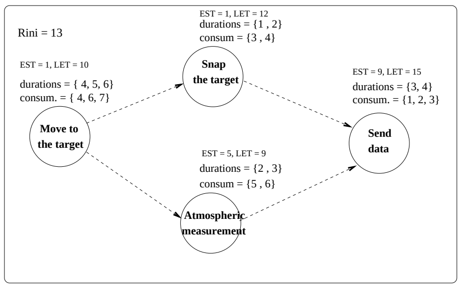

Figure 1. An acyclic graph of tasks

ple, or night-time operation of cameras or daytime operation of the night-time spectrometer. Other constraints can be considered such as illumination constraints (which involves time of day and position), setup times (warm-up or cool-down time for instruments), potentially time separation between images to infer 3D shape from shadows.

• *Bounded resources*: for the achievement of many activities, there is a need for power (moving from a location to another one) and data storage capacity (taking pictures and storing them with different resolutions).

These scenarios are mainly characterized by:

1- Single rover activities have an associated temporal window.

2- There is uncertainty on time realization tasks and the temporal interval activity of rovers.

3- The temporal constraints are, in general, soft.

4- The mission can be represented thanks to an acyclic graph, as is the case in space applications.

5- Precedence dependencies between tasks exist.

Existing work on planning under uncertainty for rovers encounter difficulties in application to real problems because most of them handle only simple time constraints and instantaneous actions. More sophisticated techniques have been developed to deal with uncertain duration [12, 13, 9], but they fail to optimally control the execution.

The requirements for a rich model of time and action are more problematic for those planning techniques [5]. The techniques based on MDP and POMDP representations can be used to represent actions with uncertainty on their outcomes [10, 7], but they have difficulties when those actions involve complex temporal dependencies and uncertain durations. Given all the temporal constraints of the decision process we need for the rover, which are not purely Markovian, the approaches should handle irregular, multi-step transitions. Semi-Markov Decision Processes SMDPs and temporal MDPs (TMDPs) [4] offer a rich model of time and action. An SMDP [11] is a special kind of MDP appropriate for modeling continuous-time discrete-event systems. The actions in an SMDP take variable amounts of time and are intended to model temporally-extended actions. This kind of decision process could be appropriate for our requirements. However, it does not express the temporal information as a characteristic of the state. This property is needed in our case, in order to know the interval of time during which a task has been executed. In the next section, we present the important features of the approach based on the construction of an MDP taking temporal constraints and uncertainty into account. This MDP uses a state representation that can express the temporal information needed for the decision-making process.

2. Principles of the approach

The mission of the agent is a graph of tasks where each task is characterized by the execution time window and the uncertainty about the execution time and resource consumption. In the rest of the paper, we assume that the graph of the mission is given. For each task, one is given a probability distribution over its finite set of execution times, and a probability distribution over its finite set of possible amounts of resource. The representation of execution time and resource consumption are discrete.

Let's consider a planetary rover that have a mission to complete. First, it must move to a target. Then, depending on the time and available resources, it can snap the target or complete atmospheric measurements. To end its mission, the rover must send the data it has collected. As shown in figure 1, this small mission can be represented by an acyclic graph. Each node stands for a task. Edges represent temporal constraints. For instance, the rover must move to the target before it can snap it. Each task has several resource consumptions and durations. "Rini" is the initial resource rate. For each task, temporal constraints are represented thanks to time windows [EST, LET] where EST is the Earliest Start Time and LET is the Latest Start Time of the task. Larger and more complex missions can easily be represented using acyclic graphs.

Given the mission graph, the problem of the agent is to choose the best decision about which task to execute and when to execute it. This decision is based on the available resources and the temporal constraints. Respecting the temporal constraints requires knowledge of the time interval during which the current task has been executed.

The decision process of the agent bases its decision on the current state of the last task executed, the remaining resources and the interval during which this task has been executed. A state of this decision process is then, [a_i, r, I] that corresponds to the last executed task a_i, the remaining resource r and the interval of time.

Given the uncertainty on the execution time, there exist many possible intervals of time during which a task could be executed, as shown in figure 2. Moreover, many possible resource levels can be available for each task. In order to explore the entire state space of the decision process, we need to know for each task in the graph the set of its possible execution interval times and its possible resource levels. To do that, we develop an algorithm that computes for each task in the graph all the possible time intervals by propagating different execution times. We also develop two algorithms to compute all the possible resource levels. These algorithms are explained in detail in the following sections.

The decision process needs to know at each step the probability that execution will occur during an interval. This information is computed by an algorithm that computes for each task and for each interval its probability. Finally, when the decision process propagates all temporal constraints and their probabilities through the graph, it can develop the state space of its decision. Note that the decision depends only on the current state and thus this process has a Markov property. Consequently, the model is based on the following steps:

• Propagating the temporal constraints and computing the set of possible time intervals for execution of each task (node in the graph). We use the terms task, activity, and node interchangeably.

• Discretizing the resource consumption levels.

• Computing the probability of the transitions.

• Constructing the state space of the Markov Decision Process and defining the transition model (described in the section "A decision model: MDP").

• Using the value iteration algorithm to solve the MDP.

In the rest of the paper, we describe the details of all those steps.

3. Preliminaries

In the previous section, we describe the overall basis of the model we develop and present its main characteristics. This model allows an agent to construct an optimal policy for executing its plan (graph of tasks) taking into account temporal and resource constraints and the uncertainty about execution duration and resource consumption. These uncertainties lead to uncertain start and end times of the remaining tasks in the plan and uncertainty about the available resources. To represent this model, we use a probability distribution on computation time and resources, and probabilities on start and end times of tasks. We also define a time-dependent utility function of task achievement.

We denote by $\delta_i(a)$ and $\delta_r(a)$ the computation time and resource consumption, respectively, of an activity a. When it is unambiguous, we will suppress the activity argument a for conciseness.

Uncertain execution time The uncertainty on execution time has been considered in several approaches developed in [10, 14, 7]. All those approaches ignore the uncertainty on the start time. We show in this paper how extensions can be considered in those approaches taking different temporal constraints into account.

Definition 1 *A probabilistic execution time distribution, $P_c(t_c) = Pr(\delta_i = t_c)$ is the probability that the activity takes t_c time units for its execution.*

The representation adopted for this distribution is discrete. We use a set of pairs (t_c, p), where each pair means that there is a probability p that the execution will take t_c time units.

Uncertain resource consumption The consumption of resources (energy, memory, etc ...) are uncertain. We assume a probability distribution on the resource consumptions of a rover when performing an activity.

Definition 2 *A probabilistic resource consumption is a probability distribution, $P_r(\Delta r) = Pr(\delta_r = \Delta r)$ of resource consumption measuring the probability that an activity consumes Δr units of resources.*

The representation adopted of this distribution is discrete. We use a set of pairs $(\Delta r, p)$ where each pair means that there is a probability p that the execution will consume Δr units of resource.

We assume that resource consumption and execution time are independent. But, this assumption does not affect the genericity of the model (we can use a probability distribution of $(\Delta r, t_c)$ such that $P((\Delta r, t_c))$ is the probability that the activity takes t_c time units and consumes Δr resources).

Temporal window of Tasks Each task is assigned a temporal window [EST,LET] during which it should be executed. EST is the earliest start time and LET is the latest end time. The temporal execution interval of the activity (start time and the end time) should be included in this interval.

Time-dependent utility Given the fact that the utility of the task achievement is time dependent, we define a time-dependent utility function.

Definition 3 *A time-dependent utility $u(a, t)$ measures the utility of finishing the achievement of task a at time t.*

4. Temporal interval , resource level and probability propagation

4.1. A simple temporal interval propagation algorithm

Given the possible transition times of different tasks, we determine the set of temporal intervals during which a task can be realized by propagation of temporal constraints through the graph. The set of possible start times is a subset of $\{EST, EST+1, EST+2, \ldots, LET - \min \delta_i\}$ (EST: Earliest Start Time, LET: Latest End Time) determined by absolute temporal constraints. We denote the Latest Start Time LST as $LST = LET - \min \delta_i$ where $\min \delta_i$ is the minimum duration of the task.

We can compute off-line all the possible end times of all of an activity's predecessors and consequently compute its possible start times. The algorithm is similar to the one described in [6]. The possible intervals I of execution are determined with a single forward propagation of temporal constraints in the graph. This propagation organizes the graph into levels such that: l_0 is the root of the graph, l_1 contains all nodes that are constrained only by the root node, ..., l_i contains all nodes whose predecessors include nodes at level l_{i-1} and all of whose predecessors are at level l_{i-1} or before. For each node in given level l_i, we compute all its possible interval times from its predecessors (a problem widely studied in literature).

- *level* l_0: the start time and the end times of the root node (the first task of the mission) are computed as follows:

  - start time of the root : $st(root) = EST(root)$

  - the set of end times of the root : $ET(root) = \{st(root) + \delta_i(root), \forall \delta_i(root)\}$

  where $\delta_i(root)$ is the computation time of the first activity (task) of the mission. Consequently, possible execution intervals of the root are given by $I = [st(root), et(root)]$, where $et(root) \in ET(root)$. Note that there is potentially a non-zero probability that some end times violate the deadline LET.

- *level* l_i: for each node in level i, its possible start times are computed as all the times at which the predecessors activities can finish. We first compute all the possible start times, and then we eliminate the start times that do not respect the constraints of earliest start time (EST) : $st < EST$, and the latest start time (LST) : $st > LST$.

  For each possible start time, we compute all the possible end times thanks to the durations of the node. Note that there is the potential that $st + \delta_i(node) > LET(node)$

Figure 2 gives an example of temporal interval propagation.

This greedy algorithm can be improved by taking advantage of information from some already computed intervals, and it is sometimes not necessary to recompute the intervals of the following tasks. Many classical algorithms exist in literature, like PERT, but most of them don't deal with uncertainties of execution time and resource constraints.

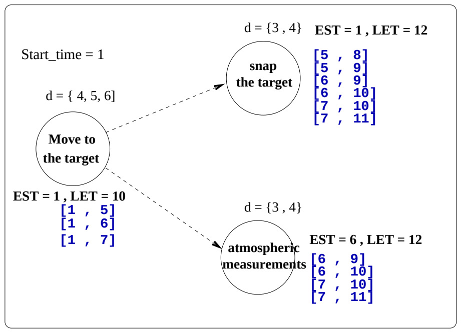

Figure 2. Temporal interval propagation example

4.2. Resource level propagation algorithm

The execution intervals are computed with temporal propagation. Two algorithms can be used in order to compute the different resource levels available at the end of each task.

Explicit resource level propagation The first method is combinatorial and computes exactly the available resource levels. For each task a_i executed by the agent, the available resource levels are computed using :

• The resource levels that can be available at the end of the last task executed by the agent, before it performs a_i.

• The resource levels that can be consumed by the execution of a_i.

It is necessary to combine the values taken by these two items. Thus, with a subtraction, we get exactly all the possible resource levels that can be available at the end of a_i's execution. An example is shown in figure 3.

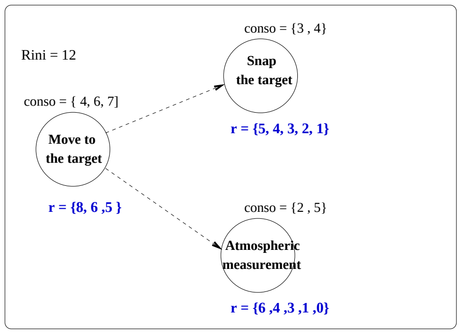

Figure 3. Explicit resource levels propagation example

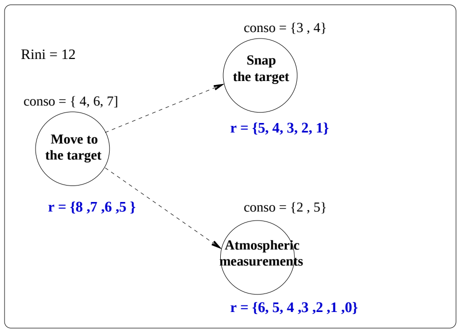

Figure 4. MinMax resource level propagation example

MinMax resource levels propagation Although the previous algorithm can compute exactly the available levels of resources, it can not be used as soon as the graph of the mission is greater than 20 tasks. Indeed, the combinatorics of considering all possible starting and ending resource levels became too difficult. Therefore, we have developed a second algorithm, allowing larger problem sizes. For each task a_i executed by the agent k, we compute the maximum (max) and the minimum (min) resource levels available at the end of a_i's execution. Then we consider that all the values in the interval $[min, max]$ (not the specific combinations computed explicitly) describe a legal resource level available for a_i. As a result, we get more possible resource levels with this algorithm. The " MinMax resource level algorithm " avoids doing all the combinations and so it can be used for larger missions. Figure 8 outlines that with the MinMax technique, our approach can be used for missions of a hundred tasks.

4.3. A Probability propagation algorithm

All possible execution intervals of each activity (or node) are computed off-line. We describe in this section how we can weight each of those intervals with a probability. This probabilistic weight allows us to know the probability that an activity will be executed during a given interval of time. For that, a probability propagation algorithm among the graph of activities is described using the execution time probability and the temporal constraints EST, LST, and LET.

This algorithm takes into account the uncertainty of execution time that affects the end time of the node and thus the start time of the next task. The probability of an execution interval P_w depends on its start time (the end time of the previous task) and the probability of execution time P_c. We can compute the probability that the execution of an activity occurs during an interval I where *st(I)* is the start time and *et(I)* be the end time.

Definition 4 *The probability of an execution interval I is the probability $P_w(I|et(I'))$ that interval I is the interval during which an activity is executed if the las executed task ends at $et(I')$. This probability measures the probability that an activity starts at $st(I)$ and it ends at $et(I)$.*

- *If* $st(I) \geq et(I') : P_w(I|et(I')) = P_c(et(I) - st(I))$
- *Else :* $P_w(I|et(I')) = 0$

where $st(I) = et(I')$, the end time of the last executed task. This probability is needed in the transition model that we will describe below.

5. A decision model: MDP

As mentioned above, we model this problem with a Markov Decision Process. To do that, we need to define the state space, the transition model, and the value function. The states need to take three parameters of the application into account : the tasks to perform, the available resource for the remaining task, and the interval of time during which a task should be performed. These parameters are important for our transition model. However, when representing the state with *[executed task, remaining resource, interval of time]*, we assume that states are fully observable because the temporal intervals are determined off-line and the resource consumption is observed by the agent as feedback of the environment.

The consequence of representing all the intervals and resource levels is that the state space becomes fully observable and the decision process can perform its maximization action selection using the Bellman equation defined below. However, the maximization action selection uses an uncertain start time that is computed from an uncertain end time of the predecessors.

The start time selected by the policy to execute one of the next activities leads to many cases. If the start time is later than the LST, this situation leads to a "deadline reached" situation that the policy should handle. Indeed, when this situation occurs the policy moves to a failure state. This situation can also occur when the execution duration is too long. We have also to consider execution that consumes more resources than are available. In the transition model, we describe in detail all these situations.

State Representation The agent observes its resource levels and the progress made in achieving its tasks which represent the state of the agent. The state is then a triplet *[task, remaining resource, interval activity]*. The state in this framework is assumed to be completely observable by the agent since the interval of the activity is completely determined off-line and the resource level is observed by the agent.

For each task, we develop a set of states by combining the intervals of execution and the remaining resource.

Transition model
The agent has the ability to act to achieve its tasks. The agent should make a decision on which task to execute and when to start its execution. The set of actions to perform consists of *Executing the next task* k at time st ($\mathsf{E_k(st)}$), where k is a successor of the last executed task. This action is probabilistic since the processing time and resource consumption of the task are uncertain. This action allows to move from state $[a_i, r, I]$ to new state s'. There are four possible transitions that we describe in the following :

- *Successful Transition:* The action allows the policy to transition to a $[a_{i+1}, r', I']$ where task a_{i+1} has been achieved during the interval I' respecting the EST and LET of this task and r' is the remaining resource. The probability of moving to the state $[a_{i+1}, r', I']$ is :

$$\sum_{\Delta_r \le r} \sum_{et(I') \le LET} P_r(\Delta_r).P_w(I'|st(I') = et(I))$$

- *Too late start time (TLST) Transition:* The action starts too late and the execution exceeds the deadline LET. In such case, the action allows the policy to transition to a $[failure, r, [st, +\infty]]$. The probability to move to this state is : $Pr(st > LST)$.
- *Deadline met Transition:* The action starts an execution at time st but the duration δ_i is so long that the deadline is met. This transition moves to the state $[failure, r, [st, +\infty]]$ (in fact, the resource and interval arguments are unimportant). The probability of moving to this state is :

$$Pr(st \le LST) \cdot \sum_{\Delta r \le r} \sum_{LET - t_c < st \le LST} .P_r(\Delta_r) \cdot P_c(t_c)$$

- *Insufficient resource Transition:* The execution action requires more resources than available. This transition moves to the state $[failure, 0, [st, +\infty]]$. The probability of moving to this state is : $\sum_{\Delta_r > r} P_r(\Delta_r)$

Figure 5 gives a representation of the relationship between the original graph structure, and the state space and transitions of the MDP. The left part of the figure stands for the mission graph. As seen previously, the nodes stand for the tasks, and edges represent the precedence constraints. The right part of the figure represents the state space of the MDP and its transitions. Each box groups together the states associated to a task a_i. Each node stands for a state. Each edge represents a transition. It links a state associated to a task a_i, with a failure state or a state associated to a task a_{i+1}, where a_{i+1} is a successor of a_i in the mission graph. The transitions between two states depend on the executed task a_{i+1}, its start time, the execution duration and the resource consumption. The task executed

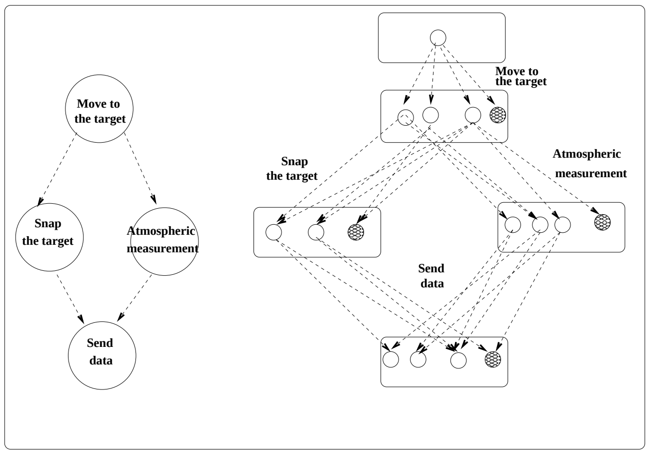

Figure 5. Relationship between the original graph structure and the MDP state space

for each transition is specified on the figure. Some nodes have no successor : they are terminal states. Failure states (striped nodes) are considered as terminal state. It is straightforward to adapt our system to non terminal failure state. States associated to the last task of the mission are also terminal states.

Utility and Value functions considering temporal constraints Given different transitions, we adapt our former to Bellman equation as follows:

$$V([a_i, r, I]) = u(a_i, et(I)) + \max_{\mathsf{E_k(st=et(I)),k=successor(a_i)}} (V') \tag{1}$$

The agent gets local reward $U([a_i, r, I]) = u(a_i, et(I))$ from being in state [a_i,r, I], and V' is the expected value from visiting future states after this state. We decompose V' as the following :

$$V' = V1 + V2 + V3 + V4$$

such that :

- *Expected Value of successful transition :*

$$V1 = \sum_{\Delta_r \le r} \sum_{et(I') \le LET} P_r(\Delta_r) P_w(I'|et(I))$$
$$\cdot V([a_{i+1}, r - \Delta_r, I'])$$

This value is when an agent starts and finishes with success, there is enough resource and I' is the interval of execution of the next activity where its start time $st(I')$ is the end time $et(I)$ of the last executed task.

- *Expected Value of TLST Transition :*

$$V2 = Pr(st > LST) \cdot V([\textit{failure}, r, [st, +\infty]])$$

This value is when the agent starts too late because the last executed task has finished too late. The execution never starts, so no resource is consumed.

- *Expected Value of Deadline met Transition:*

$$V3 = Pr(st \leq LST) \cdot \sum_{\Delta r \leq r} \sum_{LET - t_c < st \leq LST} .P_r(\Delta_r) \cdot P_c(t_c)$$

$$\cdot V([\textit{failure}, r, [st, +\infty]])$$

This value is when the deadline is met. As soon as we meet the time LET, we stop the execution.

- *Expected Value of Insufficient resource Transition:*

$$V4 = \sum_{\Delta_r > r} P_r(\Delta_r) \cdot V([\textit{failure}, 0, [st, +\infty]])$$

This value is when there is not enough resource.

The value of terminal states $V([a_n, r, I'])$ is the local utility $u(a_n, et(I'))$ of finishing the last task a_n at time $et(I')$. The failure states are also terminal and have a value of negative infinity.

$$V([a_n, r, I']) = u(a_n, et(I')) \text{ and } V([\textit{failure}, *, *]) = -\infty$$

The obtained MDP is easily solved using a standard dynamic programming, like *policy iteration*, by propagating backwards the value from terminal states to initial state. The policy obtained is optimal for this class of MDP (finite horizon with no loops). Consequently, the execution of such model of planning is optimal because of the one to one correspondence between the state of execution and the state of the obtained MDP.

6. Experimental Results

6.1. Objectives

The proposed model overcomes the difficulties we described in the introduction by proposing a rich model of decision process that can deal with complex temporal constraints, limited resources, and uncertainty. However, we need to determine the gain obtained using this approach and the scale of problems we can solve. The first experiments consist of showing the number of tasks our model can solve. We show in the following that our approach is powerful enough for a hundred tasks required by robotic applications (we can deal with more).

• *Scalability*: The objective as described in [5] for this experiment is to overcome the problem with a hundred tasks. We show in the experiments how we achieve this objective with some adaptation in discretizing resources and the *Sensitivity* of the approach to this discretization.

•*Performance*: We compare our approach with the approaches based on temporal planning techniques. This comparison is made using the expected values obtained respectively by our approach and an approach similar to the one explained in [12].

6.2. Experiments on scalability

A benchmark, composed of several missions of different sizes, has been used. For instance, the mission composed of four tasks can be represented by the figure 1.

Temporal constraints are given for each task. Thanks to these informations, we can compute the state space. Its size relies on :

• the number of tasks

• the number of intervals for each task

• the number of resource rates that can be available after the execution of each task

These parameters (intervals per task, range of available resources) affect the complexity of the problem. In the worst case, the state space size is given by the following equation :

$$\#n_{states} = \#n_{tasks} . \#n_{Max\_Interv} . \#n_{Max\_Res} \quad (2)$$

where $\#n_{tasks}$ is the number of tasks of the mission, $\#n_{Max\_Interv}$ is the maximum of intervals per task, $\#n_{Max\_Res}$ is the maximum of resource levels per task.

Number of tasks : As it can be seen in equation 2, if the number of tasks increases, the state space size grows. Figures 7 and 8 illustrate this evolution.

Number of intervals for each task : In the worst case, the number of intervals for a task is given by :

$$\#n_{Interv} = (LET - min\delta_i - EST).\#n_{durations}$$

where $\#n_{durations}$ is the number of possible execution durations for the task. If the temporal constraints are tight, the temporal window [EST, LET] is reduced. We then compute few intervals and the number of states decreases.

When we increase the size of the temporal windows, the state space size grows. Indeed, temporal constraints are less tight, and new execution plans (involving new states) can be considered. Figure 6 gives an example of this evolution considering a graph of one hundred tasks. Size“1” is the initial size of the temporal windows. Size “2” stands for sizes of windows twice larger than the initial size. On figure 6, the state space size rises and then, levels off at 1.5. Indeed, temporal windows become more and more large, and temporal constraints get more and more relaxed. At 1.5, temporal windows don’t constraint any more the execution of the agent. All the possible execution plans (and possible states) are considered and the maximum of the state space size is reached. Keeping on

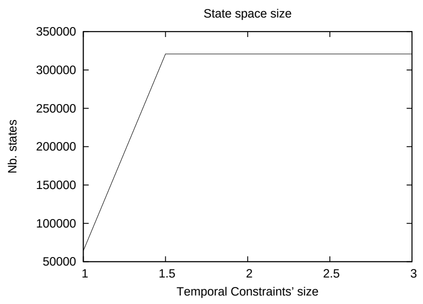

Figure 6. State space size while relaxing the temporal constraints

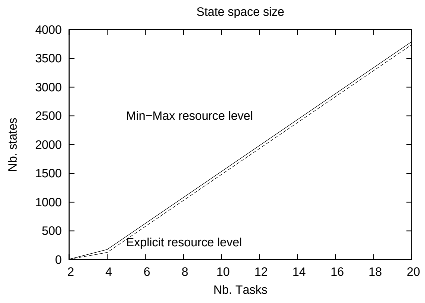

Figure 7. State space size using different resource level algorithms

relaxing the constraints doesn't increase the state space size. Thus, figure 6 shows that temporal constraints allow to prune the possible start times and end times. The more tighten the constraints are, the less execution intervals are possible and the smaller the state space is.

Number of resource levels : In the worst case, the number of resource levels for a task a_i is given by :

$$\#n_{Res} = (\sum_{a_k \in pred(a_i)} \#n_{Res}).\#n_{consum}$$

where $\#n_{consum}$ is the number of possible resource consumptions for a task. This equation gives the number of available resource levels when using the explicit resource level algorithm.

If we use the “ MinMax resource level ”, the number of resource levels for a task a_i is given by :

$$\#n_{Res} = \big((max_{a_k \in pred(a_i)} r) - (min_{a_k \in pred(a_i)} r)\big).\#n_{consum}$$

where $max_{a_k \in pred(a_i)} r$ is the maximum resource level available at the beginning of the task, and $min_{a_k \in pred(a_i)} r$ is the minimum resource level.

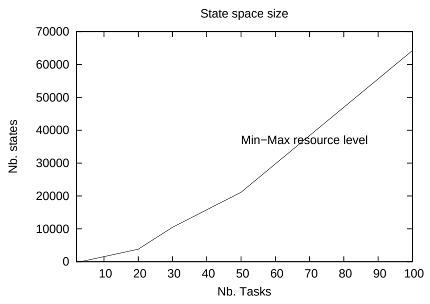

Figure 8. State space size using the “ Min-Max resource level ” algorithm

Figure 7 represents the state space size for different size of the graph, using the explicit and Min-Max resource level propagation. The explicit resource level algorithm can not be used as soon as the graph mission is greater than 20. The state space size is greater when using the “ MinMax resource level” algorithm. Indeed, with “ MinMax resource level”, some resource levels are considered to be available, but are not legal (possible). They are called “impossible” levels. The difference in state space size between the two algorithms relies on these “impossible” levels. For each task, let's consider min_c the minimum resource consumption, and max_c the maximum. If each value between min_c and max_c is a possible resource consumption for the task, all the resource levels computed by the “ MinMax resource level ” algorithm are possible. There is no difference between the explicit and the “ MinMax resource level” algorithms. While some values in $[min_c, max_c]$ are not resource consumption values, the number of “impossible” resource levels computed by the“ Min-Max resource level ” increases. Therefore, the difference between the state space size using the “ MinMax resource level” or the explicit algorithm is more important.

When we compute the value of each state, even with “ Min-Max resource level ” algorithm, we only consider the possible resource levels: impossible resource levels are not taken into account. Thus, states corresponding to resource values that are not actually possible, are not valued. We only consider the legal resource levels in the interval $[min, max]$. Therefore, the values are the same whatever the algorithm we use.

Figure 8 shows the state space size when we use the “ Min-Max resource level” algorithm. As seen previously, the state space size increases when the number of tasks grows.

If the initial resource level is large, a lot of resource levels are available for each task. When we compute the resource levels we only consider levels greater than zero. If the initial resources increase, the number of negative resource levels decreases and more levels must be taken into account. Fig-

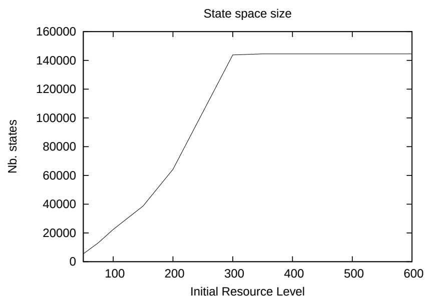

Figure 9. State space size for different initial resource level

ure 9 gives the state space size, for a one hundred task mission, when increasing the initial resource level. When the initial resources are greater than 350 units, it can be seen that the state space size levels off. Indeed, the agent has a large initial resource level, and never lacks of resources. When we compute the resource levels, each level is positive and is taken into account : increasing the initial level, does not increase the number of levels that have to be considered. If the initial resources are scarce, a lot of resource levels are pruned while computing the legal resource levels. Then, the state space size diminishes.

Branching factor If we increase the number of possible predecessors for task a_i, the number of possible resource levels available before a_i execution increases, and the number of possible start times as well. The number of branchings in the mission graph influences the state space size. More experiments about the number of branchings are under development.

MDPs smaller than one hundred states can be solved easily. Thus, these experiments shows the scalability of our approach.

6.3. Experiments on performance

For experiments on performance, we have compared our approach to a probabilistic temporal planning that uses a standard strategy using the most likely duration and resource consumption.

Table 1 gives the initial state's value using our approach and a standard strategy [1]. The value of the standard strategy is a simple adaptation of equation 1 as follows:

$$V' = \max_{et(I')\leq LET, r\geq \Delta r} (P_w(I'|et(I)_{a_i}).P_r(\Delta r)).V([a_{i+1}, r-\Delta r, I'])$$

Table 1 shows that our approach outperforms the probabilistic planning approach. More experiments are needed to produce more definitive results.

number of tasks	value (our approach)	value (standard strategy)
2	5	1.8
4	15	2.69
20	148.9	2.8
30	327361	-162.72
50	1.22	-325.38

Table 1. Initial state's values

7. Related Work

There has been considerable work in planning under uncertainty that lead to two categories of planners: conformant planners and contingent planners. These planners are characterized along to important criteria: *representation of uncertainty* and *observability*. The first criterion consisting of *uncertainty representation* has been addressed in two ways in many planners using *disjunction* or *probability* while the second criterion consisting of *Non-observability* (NO), *partial observability* (PO) or *a full observability* (FO) of planners. A survey on all classes of planners can be found in Blythe [2] and Boutilier [3] where details are given on NO, PO, or FO disjunctive planners and on NO, PO or FO probabilistic planners. Let us just recall some of those planners: C-PLAN NO-disjunctive planner, Puccini PO-disjunctive planner, Warplan FO-disjunctive planner, Buridan NO probabilistic planner, POMDP, C-MAXPLAN PO probabilistic planners and JIC, MDP FO probabilistic planners. In this section we focus on why those planners are unsuitable for our concern and why our work is a contribution to overcome those limits.

These planners encounter some difficulties in our domain of interest:

• *Model of time*: the existing planners do not support explicit time constraints nor complex temporal dependencies.

• *Model of actions*: the existing planners assume that actions are instantaneous.

• *Scalability*: the existing planners don't scale to larger problems. For rover operations, a daily plan can involve on the order of a hundred operations, many of which have uncertain outcomes.

The approach we present in this paper meets the requirements for a rich model of time and actions and for scalability. It complements the work initiated in [6] by using a similar model of time and utility distribution and by using a decision-theoretic approach. The advantage of using such approach is to achieve optimality. Another contribution consists of handling uncertainty on resource consumption combined with uncertainty on execution time.

In the MDP we present, actions are not instantaneous as in the previous planners and can deal with complex time constraints such as a temporal window of execution and temporal precedence constraints. We also show that our approach can solve large problems with a hundred operations, many of which are uncertain. Another requirement needed by the rover applications consists of continuous time and resources. We experimentally show that when we use different approaches for resource units, our approach has a minor error at run-time while we win in scalability of the approach. The tradeoff between the scalability and the time granularity shows that we can discretize the time and the outcomes of actions regarding a small error in execution. Another requirement mentioned for the rover applications is concurrent actions. This problem is under development, taking advantage of some multiagent concepts where as soon as the rover needs to execute two actions we consider those two actions are two concurrent agents. This new line of research consists of bridging the gap between the multiagent systems and the distributed MDP.

8. Conclusion and future work

In this paper we presented an MDP planning technique that allows for a plan where operations have complex dependencies and complex time and resource constraints. The operations are organized in an acyclic graph where each operation has a temporal window during which it can be executed and an uncertain resource consumption and execution time. This approach is based on an MDP using a rich model of time and resources and complex dependencies between actions. This technique allows us to deal with the variable duration of actions. We present experimental results that show that our approach can scale to large robotic problems (a hundred of operations). Our approach overcomes some of the limitations described in [5]. Indeed, our model is able to handle more complex time constraints and uncertainty on tasks' durations and resource consumptions. Moreover, as required in [5], our system can consider plans of more than a hundred tasks.

However, this approach needs to be extended to other requirements such as continuous variables. In our current version of the approach we use a discrete representation of time and resource. We show experimentally that with such representation the errors in execution are small and this representation can be tolerated. We continue experiments in this line of work to reduce the errors. We are specially interested in finding tradeoffs between the scalability, errors in execution and discretization. The other extension we are developing consists of the use of multiagent system with complex temporal dependencies using a distributed MDP.

Currently, we use a discrete representation assuming that most of states are stored. The other are manipulated as one of stored states using 1-NN techniques. This approximation does not affect the quality of the system. Future works will concern the construction of plans that we consider given in this paper.

References

[1] B. Baki, A. Beynier, M. Bouzid, and A. Mouaddib. Temporal probabilistic planning. 2004.

[2] J. Blythe. *Planning under uncertainty in Dynamic domains*. PhD, Carnegie Mellon University, 1999.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural asumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1:1–93, 1999.

[4] J. Boyan and M. Littman. Exact solutions to time-dependent mdps. In *NIPS*, 2000.

[5] 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.

[6] J. Bresina and R. Washington. Expected utility distributions for flexible contingent execution. In *AAAI Workshop on Representation issues for Real World Planning system*, 2000.

[7] S. Cardon, A. Mouaddib, S. Zilberstein, and R. Washington. Adaptive control of acyclic progressive processing task structures. In *IJCAI*, pages 701–706, 2001.

[8] T. Estlin, A. Tobias, G. Rabideau, R. Castana, S. Chien, and E. Mjolsness. An integrated system for multi-rover scientific exploration. In *AAAI*, pages 613–613, 1999.

[9] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In 17^{th} *IJCAI-01*, 2001.

[10] A.-I. Mouaddib and S. Zilberstein. Optimal scheduling for dynamic progressive processing. In *ECAI-98*, pages 499–503, 1998.

[11] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps : A framework of temporal abstraction in reinforcement learning. In *Report*, 1999.

[12] I. Tsamardinos, M. Pollack, and S. Ramakrishnan. Assessing the probability of legal execution of plans with temporal uncertainty. In *ICAPS Workshop on Planning under uncertainty and Incomplete information*, 2003.

[13] T. Vidal and M. Ghallab. Dealing with uncertain durations in temporal constraints networks dedicated to planning. In 12^{th} *ECAI*, 1996.

[14] S. Zilberstein and A.-I. Mouaddib. Reactive control for dynamic progressive processing. In *IJCAI-99*, pages 1269–1273, 1999.