HAL
open science

A Polynomial Algorithm for Decentralized Markov Decision Processes with Temporal Constraints

Aurélie Beynier, Abdel-Illah Mouaddib

To cite this version:

Aurélie Beynier, Abdel-Illah Mouaddib. A Polynomial Algorithm for Decentralized Markov Decision Processes with Temporal Constraints. Fourth International joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'05), 2005, Utrecht, Netherlands. pp.963-969, 10.1145/1082473.1082619 . hal-01344441

HAL Id: hal-01344441

https://hal.science/hal-01344441v1

Submitted on 5 Sep 2016

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 Polynomial Algorithm for Decentralized Markov Decision Processes with Temporal Constraints

Aurélie Beynier, Abdel-Illah Mouaddib
GREYC-CNRS, University of Caen
Bd Marechal Juin, Campus II, BP 5186
14032 Caen cedex, France
{abeynier, mouaddib}@info.unicaen.fr

ABSTRACT

One of the difficulties to adapt MDPs for the control of cooperative multi-agent systems, is the complexity issued from Decentralized MDPs. Moreover, existing approaches can not be used for real applications because they do not take into account complex constraints about the execution. In this paper, we present a class of DEC-MDPs, OC-DEC-MDP, that can handle temporal and precedence constraints. This model allows several autonomous agents to cooperate so as to complete a set of tasks without communication. In order to allow the agents to coordinate, we introduce an opportunity cost. Each agent builds its own local MDP independently of the other agents but, it takes into account the lost in value provoked, by its local decision, on the other agents. Existing approaches solving DEC-MDP are NEXP complete or exponential, while our OC-DEC-MDP can be solved by a polynomial algorithm with good approximation.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Markov processes*

General Terms

Algorithms

Keywords

Multi-agent systems, Markov decision Processes, Planning, Uncertainty

1. INTRODUCTION

Markov decision processes (MDPs) and partially observable Markov Decision processes (POMDPs) have proved to be efficient tools for solving problems of mono-agent control in stochastic environments. That's why extensions of these models to cooperative multi-agent systems, have become an important point of interest. Nonetheless, decentralization of the control and knowledge among the agents increases significantly the problem complexity, and effective methods to solve optimally decentralized MDPs are lacking. It has been proved that solving optimally DEC-MDP and DEC-POMDP is NEXP complete [3]. Boutilier [4] has proposed an extension of MDPs : the multi-agent Markov decision processes which is polynomial but, it is based on full observability of the global state by each agent. Other attempts allow the agents to communicate in order to exchange local knowledge : the DEC-POMDP with communication (DEC-POMDP-Com) proposed by Goldman and Zilberstein [5], the Communicative Multi-agent Team Decision Problem (COM-MTDP) described by Pynadath and Tambe [8], the multi-agent decision process by Xuan and Lesser [9], the Partial Observable Identical Payoff Stochastic Game (POIPSG) proposed by Peshkin et al. [7]. Becker et al. has identified two classes of decentralized MDPs that can be solved optimally : the Decentralized Decision Process with Event Driven Interaction (ED-DEC-MDP) [1] and the Transition-Independent Decentralized MDPs (TI-DEC-MDP) [2]. They are both exponential in the number of states, but they only deal with very small problems. Figure 1 sums up the complexity of existing approaches to solve decentralized MDPs.

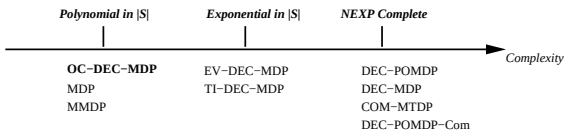

Figure 1: Classes of complexities of DEC-MDPs

In this paper, we present a class of DEC-MDPs, Opportunity Cost DEC-MDP (OC-DEC-MDP), that can handle temporal and precedence constraints. This class of DEC-MDPs allows several autonomous agents to cooperate in order to complete a set of tasks (a mission) without communication. This OC-DEC-MDP can be solved by a polynomial algorithm and can deal with large mission graphs.

The mission of the agents we consider is an acyclic graph 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 should be included in this window. Each task t_i has predecessors : the tasks that must be executed before t_i can start, and a reward : the reward the agent obtains when it has accomplished the task. The execution times and the resource consumptions of the tasks are uncertain, each task has a set of possible execution times and their probabilities, and a set of possible resource consumptions and their probabilities. Their representation is discrete. Then, $P_c(\delta_c)$ is the probability that the activity takes δ_c time units for its execution, and $Pr(\Delta r)$ is the probability that an activity con-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.
.

sumes Δr units of resources. 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 $(\Delta r, \delta_c)$ such that $P((\Delta r, \delta_c))$ is the probability that the activity takes δ_c time units and consumes Δr resources). Each node of the graph is a task and edges stand for precedence constraints. Figure 2 gives a mission example involving three agents (robots) that have to explore a radioactive land. Agent 1 has to complete radioactivity measurements and then compress the collected data. Agent 2 must snap target 2 , complete radioactivity measurements and analyze data. Agent 3 has to move to the target 1, and snap it. Edges stand for precedence constraints : the agents must have snapped the targets before agent 1 completes its radioactivity measurements. Temporal constraints are given thanks to temporal windows [EST, LET], mentioned by intervals : the radioactivity measurements on target 2 can't start before 5 and must be finished at 7. Our model can deal with more important and complex missions, and with others applications such as rescue teams of robots, satellites, ...

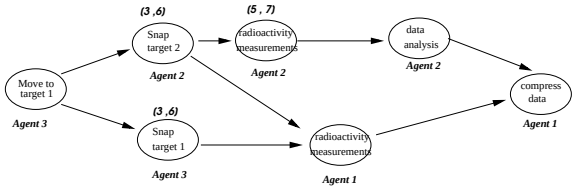

Figure 2: A mission graph

Given these information, the problem is to choose, in an autonomous way, the best decision about which task to execute and when to execute it. This decision is based on the executed tasks and on the temporal constraints. Our approach consists of solving the multi-agent decision problem thanks to a Decentralized Markov Decision Process. The system is composed on agents each of which constructs a local MDP and derives a local policy, taking into account the temporal dependencies. Indeed, in the local MDP we introduce an opportunity cost (OC) provoked by the current activity of the agent by delaying the activities of the other agents related to this activity. It measures the loss in value when starting the activity with a delay Δt. This cost is the difference between the value when we can start on time and the value when our execution is delayed by Δt. Each agent develops a local MDP independently of the local policies of the other agents but, it introduces in its expected value the opportunity cost of the possible delay provoked by his activity.

Goldman and Zilberstein [6] have identified properties of the DEC-MDP and DEC-POMDP that influence the complexity of the problem. Our OC-DEC-MDP is locally fully observable : each agent fully observes its local state. Moreover, it is observation independent : the probability an agent observes O is independent of the other agents' observations. But, because of the precedence constraints, our OC-DEC-MDP is not transition independent : the transition probability of an agent to move from a state s to s' relies on its predecessor agents. The opportunity cost allows to assess the effect of the decision of an agent on the expected value of the other agents. Therefore, the agents do not have to build belief-state MDPs that consider the agent's belief about the other agents, nor to communicate during the execution. All the information an agent needs is the OC and its observation of its local state, thus the complexity of the problem is reduced. Moreover communication is not necessary during the execution.

The construction of OC-DEC-MDP (and of the local MDPs) is based, first, on the construction of the state spaces, second, on the computation of the opportunity cost for each task and the computation of the value of each state to construct the local policy. Section 2 describes some pre-computing algorithm, used to build the OC-DEC-MDP. Section 3 provides a definition of the OC-DEC-MDP and describes how to build the local MDPs. Section 4 presents the value equation and the algorithm used to solve the problem. We also present the complexity of our algorithm. Section 5 illustrates how the algorithm performs. The contribution of this work and remarks for future research direction are given in section 6.

2. PRELIMINARIES

In order to build the local MDPs, we need further information that can be computed off-line thanks to propagation algorithms which compute each task's possible intervals of execution and their probabilities.

2.1 Temporal Propagation

Given the temporal and precedence constraints, and the execution durations for each task, we determine the set of temporal intervals during which a task can be executed, by propagating the temporal constraints through the mission graph. The graph is organized into levels such that : L_0 is the root of the graph, L_1 contains all the successors of the root (successors(root)), . . ., L_i contains the successors of all nodes in level L_{i-1}. For each node (task) t_i in a given level L_i, we compute all its possible intervals of time. The start times are in $[EST, LST]$ where $LST = LET - min(\delta_c)$ and δ_c are the possible durations of the task. However an agent can start the execution of t_i if all the predecessors of t_i have finished their execution. Thanks to the end times of the predecessors, we can validate the possible start times in $[EST, LST]$. In the following, LB_i stands for the first valid start time of t_i (Lower Bound), and UB_i is the last valid start time (Upper Bound). For each node, the intervals are computed by :

- *level* L_0 : the start time and the end times of the root node (the first task of the mission) are computed as follows :

$$st(root) = max\{EST(root), start_time\}$$

$$= LB_{root} = UB_{root}$$

$$ET(root) = \{st(root) + \delta_c^{root} \leq LET, \forall \delta_c^{root}\}$$

  where δ_c^{root} is the computation time of the first activity (task) of the mission and $start_time$ is the system "wake up" time. Consequently, intervals of activities of the root are given by $I = [st(root), et(root)]$, where $et(root) \in ET(root)$.

- *level* L_i : Each node in level L_i starts its execution when all its predecessors end their own activities. Since each predecessor has different possible end times, the possible start times for nodes in level L_i are also numerous. Firstly, we compute the first possible start time of the considered node. To do that, we compute the set of first end times (FET) of the predecessors :

$$FET(node) = \bigcup_{pred \in parents(node)} (\min_{et \in ET(pred)} (et))$$

  and then the maximum of these first end times represents the first possible start time for the node.

$$FETAP(node) = \max\{EST, \max_{et \in FET(node)} (et)\}$$

$$LB_{node} = FETAP(node)$$

Secondly, we compute the other possible start times of the node. To do that we consider all the other possible end times of the predecessors. Indeed, any predecessor *pred* finishing after FETAP (at end_time) is a possible start time of the node because it represents a situation where all the other predecessors have finished before FETAP and predecessor *pred* has finished after FETAP at end_time. Consequently, the other possible start times are defined as follows :

$$OST(node) = \{et \in ET(pred), et > FETAP\}$$

$$UB_{node} = max\{OST(node)\}$$

The set of the possible start times of a node are :

$$ST(node) = \{st \in FETAP(node) \bigcup OST(node)\}$$

such that $EST_{node} \leq st \leq LST_{node}$

The set of the possible end times of the node is then given by : $\forall node \in level_i$, $[ET(node) = \bigcup_{\forall \delta_c^{node}, st} \{st + \delta_c^{node} < LET_{node}\}]$ where δ_c^{node} is the computation time of the activity (task) at node $node$ and $st \in ST(node)$.

2.2 Probability Propagation

We describe, in this section, how we can weight each of these computed intervals with a probability. This probabilistic weight allows us to know the probability that an activity can be executed during an interval of time with success. For that, a probability propagation algorithm among the graph of activities is described using the execution time probability. This algorithm has to use the precedence constraints that affect the start time of each node, and the uncertainty of execution time that affects the end time of the node.

2.2.1 Probability on start time

The computation of conditional start time has to consider the precedence constraints. Indeed, they express the fact that an activity cannot start before its predecessors finish.

Let's consider that the initial policy of an agent starts the activities as soon as the predecessors finish. Consequently, the probability $DP(t)$ that an agent starts an activity at t is the product of the probability that all predecessor agents terminate their activities before time t and there is, at least, one of them that finishes at time t. More formally speaking :

•for the root : $DP(t) = 1, t = UB_{root} = LB_{root}$

•for the other nodes :

$$DP(t) = \prod_{a \in predecessors(c)} P_{end}^{a}(\delta_e \leq t) - \sum_{t_1 \leq t} DP(t_1)$$

Where a is an activity of a predecessor agent of the agent performing the considered activity c and $P_{end}(\delta_e \leq t)$ is the probability that predecessor a finishes before or at time t. $DP(t)$ is called "absolute probability" because it is not relative to any decision on the start time of the other agents, the agent starts as soon as possible.

On the other hand, let's suppose that the agent decides to start an activity at time t even if it can have started before t. The probability that this situation occurs is $\sum_{t' \leq t} DP(t')$: the probability that the predecessors have finished at t or before. This probability is called " relative probability" because it is relative to the decision of starting the execution at t.

In the following, $P_w^{abs}(I)$ is the probability that the execution occurs in the interval I if the agent follows its initial policy (start as soon as possible). $P_w^{rel}(I)$ stands for the relative probability that the execution occurs in the interval I if the policy dictates to the agent to start the activity at st.

Then, $P_{end}^{a}(\delta_e \leq t)$ can be given as follows :

$$P_{end}^{a}(\delta_e \leq t) = \sum_{I_1 \in intervalles(a), et(I_1) \leq t} P_w^{abs}(I_1)$$

This probability is the sum of probabilities that each predecessor a executes its task in an interval I_1 with an end time $et(I_1)$ less than t.

Now we are able to show how we can compute the probability that an execution occurs in an interval I.

2.2.2 Probability on a temporal interval of an activity

The probability that the execution duration of an activity t_{i+1} occurs during $I = [st, et]$ is the probability that t_{i+1} starts at st and it takes $\Delta t = et - st$ to finish. Furthermore, an agent starts t_{i+1} when it finishes t_i. Consequently, it knows the end time, let it be $et(I')$, of t_i. That's why we need to compute the probability that t_{i+1} occurs in I when the agent knows that t_i finishes at $et(I')$. To do that, we compute the probability $P_w^{abs}(I|et(I')_{t_i})$ and $P_w^{rel}(I|et(I')_{t_i})$ such that :

$$P_w^{abs}(I|et(I')_{t_i}) = DP(st(I)|et(I')_{t_i}).P_c(et(I) - st(I))$$

$$P_w^{rel}(I|et(I')_{t_i}) = \sum_{t' \leq st(I)} DP(t'|et(I')_{t_i}).P_c(et(I) - st(I))$$

Now, the probability $DP(st(I)|et_{t_i}(I'))$ used below means that we know activity t_i has finished at $et(I')$, and only the probability that the other predecessor activities have been finished, should be considered.

$$DP(st(I)|et(I')_{t_i}) = \prod_{a \in predecessors(t_{i+1}) - t_i} P_{end}^{a}(\delta_e \leq st(I)|et(I')_{t_i})$$

$$- \sum_{t_1 < st(I')} DP(t_1|et(I')_{t_i})$$

$$P_{end}^{a}(\delta_e \leq t|et(I')_{t_i}) = \sum_{I_1 \in intervalles(a), et(I_1) \leq t} P_w^{abs}(I_1|et(I')_{t_i})$$

3. FORMAL FRAMEWORK DESCRIPTION

3.1 The OC-DEC-MDP model

From the graph of tasks (mission) that the agent should accomplish, we isolate for each agent the tasks it has to execute, named agent's graph in figure 3. This agent's graph is then used to build the local MDP of the agent taking into account the precedence constraints. We name the set of local MDPs, OC-DEC-MDP.

DEFINITION 1. *An OC-DEC-MDP for n agents is a set of n local MDPs.*

A local MDP for an agent A_i is defined by a tuple $< S_i, T_i, P_i, R_i >$ where :

- *S_i is the finite set of states of the agent A_i*
- *T_i is the finite set of tasks of the agent A_i*
- *P_i is the transition function where $P(s_i'|s_i, t_i)$ is the probability of the outcome state s_i' for agent A_i when it executes the task t_i in state s_i*

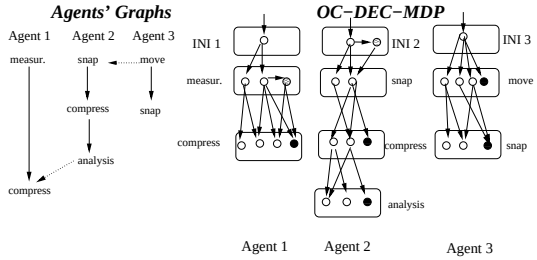

Figure 3: Relationship between the graphs of tasks and the OC-DEC-MDP

- *R_i is the reward function. $R_i(t_i)$ is the reward obtained when the agent has executed t_i.*

As we will explain later, dependencies between local MDPs (due to temporal and precedence constraints) are taken into account thanks to the Opportunity Cost. In the rest of the section, we give more details about each component of the local MDPs.

3.1.1 States

At each decision step, A_i has executed a task t_i during an interval I and it has r resources. The decision process constructs its decision given these three parameters. The states are then triplets $[t_i, I, r]$ where t_i is the last executed task and I is the interval during which the task t_i has been executed and r is the rate of available resources.

Figure 3 shows that a fictitious initial task is added in each local MDP. Each box gathers the states associated to a task t_i. Each node stands for a state $[t_i, I, r]$.

3.1.2 Tasks - Actions

At each decision step, the agent must decide when to start the next task. The actions to perform consist of*"Executing the next task t_i at time st : $E(t_i, st)$"*, that is the action to start executing task t_i at time st when the task t_{i-1} has been executed. Actions are probabilistic since the processing time and the resource consumption of the task are uncertain.

3.1.3 Transitions

We assume that the actions of one agent is enough to achieve a task. The action of an agent allows the decision process to move from state $[t_i, I, r]$ to a success state $[t_{i+1}, I', r']$ or to a failure state. The transitions are formalized as follows :

- *Successful transition:* The action allows the process to change to a state $[t_{i+1}, I', r']$ where task t_{i+1} has been achieved during the interval I' respecting the EST and LET time of this task. $r' = r - \Delta r$ is the remaining resources for the rest of the plan. The probability to move to the state $[t_{i+1}, I', r']$ is :

$$P_1 = \sum_{r \geq \Delta r} \sum_{et(I') \leq LET} P_r(\Delta r).P_w^{rel}(I'|et(I)_{t_i})$$

- *Too early start time Transition :* The agent starts too early before all the predecessor agents have finished their tasks. We consider that when an agent starts before its predecessor agents finish, it realizes it immediately. This means that the agent, at $st + 1$, realizes that it fails. This state is a non-permanent failure because the agent can retry later. We represent this state by $[t_i, [st, st + 1], r']$ where $r' = r - \Delta r'_{failure}$ such that $\Delta r'_{failure}$ is a penalty in resource when an agent fails. The probability to move to this state is the probability that the predecessors have not finished and the agent has enough resources to be aware of it. The probability $P_{not_end}(st)$ that the agent's predecessors have not finished at st is the probability that the predecessors will finish later or will never finish.

$$P_{not_end}(st) = (\prod_{a \in predecessors(t_{i+1}) - t_i} \sum_{I_a \,: et(I_a) > LET - min\delta_i}$$

$$P_w(I_a|et(I)_{t_i}) - \sum_{t' \leq st} DP(t'|et(I)_{t_i}))+$$

$$(1 - \prod_{a \in predecessors(t_{i+1}) - t_i} \sum_{I_a} P_w(I_a|et(I)_{t_i}))$$

The probability to move to a state $[t_i, [st, st + 1], r']$ is : $P_2 = P_{not_end}(st)$ where $r' \geq 0$ and $st \leq UB$.

- *Insufficient resource Transition :* The execution requires more resources than available or the predecessors have not finished and the necessary resources to be aware of it are not sufficient ($r' < 0$). This transition moves to the state $[failure_{t_{i+1}}, [st, +\infty], 0]$. The probability to move to this state is :

$$P_3 = \Big(\sum_{t' \leq st} DP(t'|et(I)_{t_i}) \sum_{r < \Delta r} P_r(\Delta r) + P_{not_end} \Big)$$

- *Too late start time Transition :* The task starts too late and the execution meets the deadline LET. The agent moves to the state $[failure_{t_{i+1}}, [st, +\infty], r]$. This case arises when $st > LST$. Otherwise ($st \leq UB$), $P_4 = 0$. The probability to move to this state is : $P_4 = P_{not_end}(st > UB)$

- *Deadline met Transition :* The action starts an execution at time st but the duration $\delta_c^{t_{i+1}}$ is so long that the deadline is met. This transition moves to the state $[failure_{t_{i+1}}, [st, +\infty], r]$. The probability to move to this state is :

$$P_5 = \sum_{r \geq \Delta r} \sum_{st + \delta_c^{ti1} > LET} P_r(\Delta r).DP(st|et(I)_{t_i}).P_c^{t_{i+1}}(\delta_c)$$

It is straightforward that $P_1 + P_2 + P_3 + P_4 + P_5 = 1$ and the transition system is complete.

3.1.4 Rewards

Each agent, A_i, receives a reward R_i presumably based on the executed task. The reward function is assumed given $R_i(t_i)$ and $R_i([failure_{t_i}, *, *]) = 0$.

Once the local MDPs have been built, each agent's policy can be computed. The next section describes how to solve the OC-DEC-MDP.

4. PROBLEM RESOLUTION

A policy is computed for each agent, thanks to a modified Bellman equation and a polynomial dynamic programming algorithm. The use of OC in the Bellman equation allows each agent to take into account the other agents, and therefore to coordinate without communication. This section presents the modified Bellman equation we use and introduces the notion of OC. It also gives some results about the valuation algorithm we have developed.

4.1 Value Function

4.1.1 Bellman Equation

Given the different transitions described previously, we can adapt our former to Bellman equation as follows :

$$V[t_i, I, r] = \overbrace{R_i(t_i)}^{\text{immediate gain}} - \overbrace{\sum_{k \in \text{succ}} OC_k(et(I) - LB_k)}^{\text{Opportunity Cost}}$$

$$+ \overbrace{max_{E(t_{i+1}, st), st > current_time}(V')}^{\text{Expected value}}$$

where $succ$ are the successors of t_i executed by an other agent, $et(I)$ is the end time of the interval I and LB_k is the first possible start time of the task t_k. $OC_k(et(I) - LB_k)$ is the opportunity cost provoked on t_k when its possible start times are reduced to $[LB_k + \Delta t, UB_k]$.

This equation means that the agents' rewards $R_i(t_i)$ are reduced by an opportunity cost due to the delay provoked in the successor agents.

The best action to execute in state $[t_i, I, r]$ is given by :

$$\overbrace{argmax_{E(t_{i+1}, st), st > current_time}(V')}^{\text{Expected value}}$$

V' is such that : $V' = V1 + V2 + V3 + V4$. Each part of V' stands for a type of transitions :

- *Successful transition :* $V1 = P_1.V([t_{i+1}, I', r'])$
- *Too early start time Transition :*

$$V2 = P_2.V([t_i, [st, st+1], r'])$$

- *Insufficient resource Transition :*

$$V3 = P_3.V([failure_{t_{i+1}}, [st, +\infty], 0])$$

- *Too late start time or Deadline met Transitions :*

$$V4 = (P_4 + P_5).V([failure_{t_{i+1}}, [st, +\infty], *])$$

The value of the failure state associated to t_i is given by :

$$V^*_{fail(t_i)} = -R_{t_i} - \sum_{suc \notin agent(t_i)} OC(fail)_{succ} - \sum_{suiv \in agent(t_i)} R_{suiv}$$

where $-R_{t_i}$ is the immediate penalty due to the failure of t_i, $-\sum_{suc \notin agent(t_i)} OC(fail)_{suc}$ is the Opportunity Cost on the successors of t_i ($succ$ are the successors of t_i executed by other agents), and $-\sum_{suiv \in agent(t_i)} R_{suiv}$ is the lost in value due to the failure of all the remaining tasks executed by the same agent as t_i.

4.1.2 Approximation of Opportunity Cost

Opportunity cost comes from economics. We use it to measure the effect of the decision of an agent A_i, about the start time of t_i, on a successor agent A_j. Let t_j be the successor task of t_i executed by A_j. Agent A_i can select a start time st_i of t_i in $[LB_i, UB_i]$ while agent A_j selects the start time st_j of t_j in $[LB_j, UB_j]$. The fact that A_i decides to start at st_i and it finishes at et_i, can reduce the possible start times of t_j to $[LB_j + \Delta t, UB_j]$ where $\Delta t = et_i - LB_j$ (t_j can start after t_i finishes). Let V_j^{*0} be the expected value of A_j when st_j can be selected from $[LB_j, UB_j]$, and $V_j^{*\Delta t}$ the expected value when st_j can be selected from $[LB_j + \Delta t, UB_j]$.

$$OC_{t_j}(\Delta t) = V^{*0}_{t_j} - V^{*\Delta t}_{t_j} \quad (1)$$

This cost is computed for all Δt from 0 to $UB_j - LB_j$.

$$V^{*\Delta t}_{t_j} = \sum_{\Delta t_c^{t_j}} P_c(\delta_c^{t_j}).V[t_j, [st_j, st_j + \delta_c^{t_j}], r^{max}_{t_j}]$$

$[t_j, [st_j, st_j + \delta_c^{t_j}], r^{max}_{t_j}]$ describes the states that can be reached when the execution of the task t_j starts at st_j. st_j is the best start time of t_j in $[LB + \Delta t, UB]$ which is the set of possible start times when the task is delayed by Δt. $[st_j, st_j + \delta_c^{t_j}]$ describes the possible execution intervals. When $st_j + \delta_c^{t_j} > LET$, we have :

$$V[t_j, [st_j, st_j + \delta_c^{t_j}], r^{max}_{t_j}] = V(failure(t_j))$$

When $\Delta t > UB_{t_j} - LB_{t_j}$ the execution starts before the last possible start time UB_{t_j}, temporal constraints are violated and $V^{*\Delta t}_{t_j} = -\Big(R(t_j) + \sum_{a \in AllSucc(t_j)} R(a)\Big)$.

The expected values $V^{*0}_{t_j}$ and $V^{\Delta t}_{t_j}$ have been computed for states $[t_j, [st_j, st_j + \delta_c^{t_j}], r^{max}_{t_j}]$ where $st_j \in [LB + \Delta t, UB]$. We assume (assumption) that agent A_j has its maximum resources $r^{max}_{t_j}$ because there is no effect of the decision of the agent A_i on the resources of agent A_j. Also, we need to assess the effect of agent A_i's decision on failing agent A_j when this failure is not due to the lack of resources of agent A_j. That's why, we assume that A_j has all its maximum resources :

$r^{max}_{t_j} = r_{ini} - \sum_{t_k \in Pred(t_j)} min_{t_k}(\Delta_r)$ where $Pred(t_j)$ is the set of tasks executed by the agent before t_j, and $min_{t_k}(\Delta_r)$ is the minimal resource consumption of t_k.

Our modified Bellman equation is used in a valuation algorithm which computes the values of all the agents' states and thus determines a policy for each agent.

4.2 Resolution of the OC-DEC-MDP

4.2.1 Valuation algorithm

In order to evaluate each state in the local MDPs and to compute each agent's policy, a dynamic programming algorithm is used. Because of dependencies between the graphs of the agents, the algorithm evaluates the local MDPs in parallel. In figure 3, if we want to compute the values of the states associated to "snap target 1", we need the value of the opportunity cost for "radioactivity measurements by agent 1". To compute the opportunity cost for "radioactivity measurements by agent 1", we must have valued the states associated to this task, and consequently we need the OC for "analyze data". Therefore the local MDPs of agents 1, 2 and 3 must be valued at the same time. The states of the OC-DEC-MDP (union of the states of the local MDPs), are organized into levels. The first level contains the root of the mission graph. The level L_n contains all the successors, in the mission graph, of the tasks in L_{n-1}. For each level, from the last level to the root, the valuation algorithm evaluates the states associated to each task in the current level. Thus, states from different local MDPs are valued at the same time. The "pseudo-code" of the valuation algorithm is :

The computation of a value $V([t_i, I, r])$ for a state $[t_i, I, r]$ is done thanks to the previous Bellman Equation. It allows to determine the policy for this state.

Each agent knows the mission graph and then computes its policy. This algorithm can be executed by each agent in a centralized way if the agents can not communicate before the execution. If the agents can communicate before the execution, the algorithm can be executed in a decentralized way : each agent computes the values of its own states and sends the OC values of each of its task t_i to the predecessors of t_i. The mission is always executed in a distributed way and the agents do not communicate.

Algorithm 1 Evaluation Algorithm

1: **for** level L_n from the leaves to the root **do**
2: **for all** task t_i in level L_n **do**
3: Compute V for the failure state: $[failure(t_i), *, *]$
4: **for all** start time st from UB_{t_i} to LB_{t_i} **do**
5: **for all** resource rate r_{t_i} of a partial failure **do**
6: Compute V for non-permanent failure state : $[t_i, [st, st+1], r_{t_i}]$
7: **end for**
8: **for all** duration Δt_1 of t_i **do**
9: **for all** resource rate r_{t_i} of t_i's safe execution **do**
10: Compute V for the safe states : $[t_i, [st, st + \Delta t_1], r_{t_i}]$
11: **end for**
12: **end for**
13: Compute $V_k^{*\Delta t}$ where $\Delta t = st - LB_{t_i}$
14: **end for**
15: **for all** $V_k^{*\Delta t}$ computed previously **do**
16: Compute $OC(\Delta t) = V_k^{*0} - V_k^{*\Delta t}$
17: **end for**
18: **end for**
19: **end for**

4.2.2 Problem Complexity

An upper bound on the complexity of OC-DEC-MDP can be computed. It takes into account the state space size of the OC-DEC-MDP which relies on : $\#n_{tasks}$ the number of tasks of the agent, $\#n_{Max_Interv}$ the maximum of intervals per task, $\#n_{Max_Res}$ the maximum of resource levels per task and $\#n_{Max_Start}$ the maximum of start time per task.

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

$$|S| = \#n_{tasks}.(\#n_{Max_Interv} + n_{Max_Start}).\#n_{Max_Res} \quad (2)$$

In the worst case, each value in [LB, UB] is a possible start time and each duration is possible. Then,

$$\#n_{Max_Interv} = (UB - LB) \times \#n_{Max_Dur} \quad (3)$$

and $\#n_{Max_Start} = UB - LB$ where $\#n_{Max_Dur}$ is the maximum number of durations for a task.

THEOREM 1. *An OC-DEC-MDP with temporal and precedence constraints is polynomial in the number of states.*

Proof : The algorithm described previously solves the OC-DEC-MDP. It passes through the state space of each local MDP and values each state. The valuation of a state "Compute V" has a complexity of $O(1)$. Lines 4-10 value all the states associated to a task t_i. The complexity of this phase is $O((\#n_{Max_Interv} + \#_{Max_Start}).\#n_{Max_Res})$. Lines 11-15 compute the OC value for each delay of t_i. In the worst case, the number of delays is equal to the number of possible start times. That's why, the complexity is $O(UB - LB)$. Lines 4-15 are executed for each task. Moreover, $O(UB - LB) << O(\#n_{Max_Interv})$. The overall complexity of the algorithm is then $O(\#n_{tasks}.(\#n_{Max_Interv} + \#n_{Max_Start}).\#n_{Max_Res})$.

Given equation 2, the complexity is $O(|S|)$ and the algorithm is polynomial in $|S|$ where S is the set of states. □

To sum up, our valuation algorithm of OC-DEC-MDP is polynomial in $|S|$ whereas other approaches are in best case exponential in $|S|$. The states do not include observations about the other agents, so the number of states is quite fair. Moreover, most of existing approaches do not consider temporal and precedence constraints even if such constraints are often encounter in MAS.

THEOREM 2. *Adding communication during the execution of the agents with unlimited resources, does not improve the performance of the OC-DEC-MDP.*

Proof : Let us assume that agents send messages to interested agents when they finish a task. And, an agent α does not start the execution of a task t_i until it receives a notification from all agents executing tasks t_j preceding task t_i. Does this communication between agents improve the performance of the OC-DEC-MDP ?

Let i be the time at which all the messages from agents achieving tasks t_j preceding the task t_i and $[LB, UB]$ be the lower and upper bounds of start time of task t_i. Three cases are possible in which we compare respectively the cost $K1$ of OC-DEC-MDP and the cost $K2$ of OC-DEC-MDP-COM (OC-DEC-MDP augmented with a mechanism of communication).

- $i \in [LB, UB]$: OC-DEC-MDP-COM has a a cost :

  $$K2 = \sum_{k \in pred(\alpha)} cost_of_Com_k + cost_wait(i - LB)$$

  OC-DEC-MDP without communication has as a cost :

  $$K1 = (i - LB).Cost_Partial_failure$$

  We assume in our case that the cost of partial failure is 1 time unit and that the communication, in best case, costs (cheapest) also 1 time unit. In such situation, we have :$K2 = (i - LB) + pred(\alpha)$ and $K1 = (i - LB)$

  We can see that in this case OC-DEC-MDP outperforms OC-DEC-MDP-COM.

- $i < LB$: in this case, as soon as the OC-DEC-MDP tries to start it successes. Then the cost is 0. However, the cost of communication in OC-DEC-MDP-COM is at least 1 time unit. Consequently, the OC-DEC-MDP outperforms OC-DEC-MDP-COM.

- $i > UB$: in OC-DEC-MDP, agent α will temporary fail $UB - LB$ times before failing definitively while in OC-DEC-MDP-COM, it will wait up to UB before failing definitively. In addition to that, messages are sent but they won't be used. Consequently, OC-DEC-MDP outperforms OC-DEC-MDP-COM.

  $$K2 = \sum_{k \in pred(\alpha)} cost_of_Com_k + cost_wait(UB - LB)$$

  $$K1 = (UB - LB).Cost_Partial_Failure = UB - LB \square$$

5. EXPERIMENTS

Experiments have been developed in order to test the scalability and performance of our approach. As mentioned previously, the state space size relies on several parameters. For instance, changes in the temporal constraints influence the number of intervals per task. In the worst case, the number of intervals for a task is given by equation 3. If the temporal constraints are tight, the interval [LB , UB] 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.

When we increase the number of agents that execute the mission, the state space size decreases : each agent has less tasks to execute, the number of triplets $[t_i, I, r]$ is smaller. For a mission of 200 tasks and 3 agents, the number of states per local MDP is about 250 000. If we increase the number of agents to 15, then the state space size of each local MDPs is about 75 000 states. A peak in the number of states can be observed when starting to increase the number of agents. It is due to an augmentation in the number of available resources for each task. In our experiments, initial resources are tight. Moreover, we don't consider the possible rates less than zero. When we increase the number of agents, each one has less tasks to execute and the initial resources become wider. The lack of resources ($r \leq 0$) becomes scarce and the number of possible resource rates, greater than zero that have to be considered, increase. Figure 4 shows the changes in the state space size through the mission's size and the number of agents. If we increase the initial resource rate, resources become wider and agents don't lack of them, all the resource combinations are positive. The augmentation in the number of agents, don't increase the number of possible resource rates greater than zero. If resources are unlimited there is no peak.

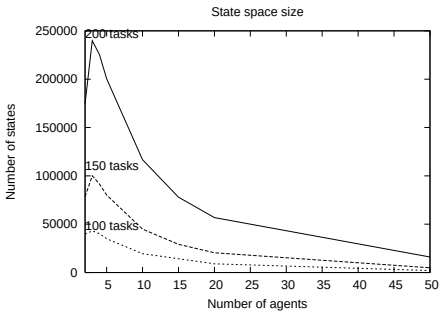

Figure 4: Changes in the state space size

When we increase the number of precedence constraints per task, the number of states decreases. The constraints reduce the number of possible intervals of execution per task, and therefore the number of states diminishes. For instance, given 3 agents and 200 tasks to execute, the state space size is about 500 000 states for 25 constraints and 300 000 states for 45 constraints. Figure 5 shows these changes for missions of 100, 150 or 200 tasks executed by 3 agents (worst case in figure 4).

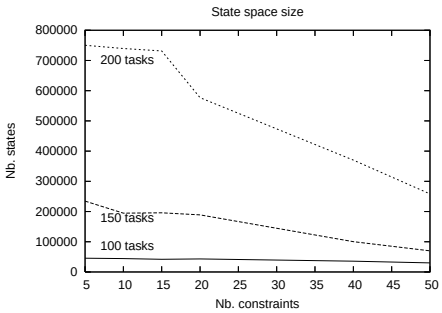

Figure 5: Changes in the state space size

Experiments have shown that our approach can deal with large mission graphs. Experiments with larger graphs are under development, even if first experiments have proved that our approach is suitable for the applications we deal with. Different scenari have also been implemented on simulators, they involve several robots that have to explore a planetary site or deal with a crisis situation. First experiments have shown that our approach allows the agents to accomplish their tasks with weak number of partial failures. Although, this approach is not optimal. The current experimental results show that it's a good approximation.

6. CONCLUSION

The framework of decentralized MDPs has been proposed to solve decision problem in cooperative multi-agent systems. Nevertheless, most of DEC-MDP are NEXP-Complete. In this paper we have identified the OC-DEC-MDP class which has a polynomial complexity. Moreover, OC-DEC-MDP model can handle temporal and precedence constraints that are often encountered in real world applications, but not considered in most of existing models. Each agent builds its own local MDP, taking into account its own tasks. Each local MDP is a standard MDP valued off-line thanks to our algorithm that uses a modified Bellman equation and computes the OC values. Thanks to the opportunity cost no communication is needed during the execution, the agents can work even if communications are not possible or too expensive. Moreover, the agents do not have to complete observations about the others, so the complexity of the problem is reduced and a polynomial algorithm has been proposed to solve such OC-DEC-MDP.

Future work will concern the improvement of the performance of this approach by better estimating OC. In the current approach, OC is over-estimated and the agents assume that for each decision there is a cost. In fact, there is a likelihood that this cost can happen. That's why, a new approach should be considered with an expected opportunity cost.

7. ACKNOWLEDGEMENTS

The authors would like to thank Shlomo Zilberstein for his helpful comments. This project was supported by the national robotic program Robea and "Plan Etat-Région".

8. REFERENCES

[1] R. Becker, V. Lesser, and S. Zilberstein. Decentralized Markov Decision Processes with Event-Driven Interactions. In *AAMAS*, volume 1, pages 302–309, NYC, 2004.

[2] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-Independent Decentralized Markov Decision Processes. In *AAMAS*, pages 41–48, Melbourne, Australia, July 2003.

[3] D. Bernstein and S. Zilberstein. The complexity of decentralized control of mdps. In *UAI*, pages 819–840, 2000.

[4] G. Boutilier. Sequential optimality and coordination in multiagents systems. In *IJCAI*, pages 478–485, 1999.

[5] C. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multiagent systems. In *AAMAS*, pages 137–144, 2003.

[6] C. Goldman and S. Zilberstein. Decentralized control of cooperative systems : Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, to appear.

[7] L. Peshkin, K. Kim, N. Meuleu, and L. Kaelbling. Learning to cooperate via policy search. In *UAI*, pages 489–496, 2000.

[8] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, pages 389–423, 2002.

[9] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multiagent cooperation. In *Autonomous Agents*, pages 616–623, 2000.