Solving efficiently Decentralized MDPs with temporal and resource constraints

Aurélie Beynier, Abdel-Illah Mouaddib

Noname manuscript No.
(will be inserted by the editor)

Solving Efficiently Decentralized MDPs with Temporal Constraints

Aurélie Beynier · Abdel-Illah Mouaddib

the date of receipt and acceptance should be inserted later

Abstract Optimizing the operation of cooperative multi-agent systems that can deal with large and realistic problems has become an important focal area of research in the multi-agent community. In this paper we first present a new model, the OC-DEC-MDP (Opportunity Cost Decentralized Markov Decision Processes), that allows for representing large multi-agent decision problems with temporal and precedence constraints. Then, we propose polynomial algorithms to efficiently solve problems formalized by OC-DEC-MDPs. The problems we deal with consist of a set of agents that have to execute a set of tasks in a cooperative way. The agents cannot communicate during execution and they have to respect some resource and temporal constraints. Our approach is based on Decentralized Markov Decision Processes (DEC-MDPs) and uses a concept of opportunity cost borrowed from economics to obtain approximate control policies. Currently, the best existing techniques can only solve optimally small problems. Experimental results show that our approach produces good quality solutions for complex problems which are out of reach of existing approaches.

Keywords Decentralized Markov Decision Processes, Multiagent Systems, Decision-theoretic Planning, Uncertainty

1 Introduction

There has been significant progress recently with the development of extensions of Markov Decision Processes (MDP) for planning and control of teams of cooperative agents (Boutilier, Dean, and Hanks, 1999). Decentralized Partially Observable Markov Decision Processes (DEC-POMDPs) and Decentralized Markov Decision Processes (DEC-MDPs) have been proposed to solve problems of multi-agent decentralized control. Nonetheless, they suffer from high complexity. Indeed, it has been shown that solving optimally a DEC-POMDP is NEXP-hard (Bernstein, Zilberstein, and Immerman, 2002). Thus, optimal algorithms can only solve very small problems in practice (Hansen, Bernstein, and Zilberstein, 2004; Szer, Charpillet, and Zilberstein, 2005). This has created a growing interest in developing good

GREYC-CNRS
Bd Marechal Juin, Campus II
BP 5186
14032 Caen cedex, France E-mail: {abeynier,mouaddib}@info.unicaen.fr

approximate solution techniques. Some of these approximations reach local optima, for example, the Joint Equilibrium based Search for Policies (JESP) by Nair et al. (2003). Other models use heuristic methods to approximate the optimal policy Goldman and Zilberstein (2003). Another approximation approach is to use on-line learning as in the Partial Observable Identical Payoff Stochastic Game (POIPSG) proposed by Peshkin et al. (2000). On the other hand, Goldman and Zilberstein (2004) have identified properties of DEC-MDPs and DEC-POMDPs that can reduce complexity such as transition and observation independence. Becker et al. have identified two classes of DEC-MDPs that are no harder than exponential in the number of states: the Decentralized MDPs with Event Driven Interaction (ED-DEC-MDP) (Becker, Lesser, and Zilberstein, 2004a) and the Transition-Independent Decentralized MDPs (TI-DEC-MDP) (Becker, Zilberstein, Lesser, and Goldman, 2003). These classes take into account dependencies between tasks and can be solved optimally by the Coverage Set Algorithm (Becker, Zilberstein, Lesser, and Goldman, 2004b).

Techniques developed so far have been used to solve relatively small problems. The issue of scalability remains a serious challenge even for approximation methods. Moreover, these approaches propose limited time and action representations (for instance, all actions are supposed to have the same duration). In this paper, we introduce a new model, the OC-DEC-MDP (Opportunity Cost Decentralized Markov Decision Processes) that can deal with large problems where temporal, precedence and resource constraints must be respected. Furthermore, we present polynomial algorithms that can compute coordinated policies of agents with a good quality using a value function augmented with an opportunity cost. We consider problems where communication during the execution is impossible and the agents must be able to coordinate without communicating. Our algorithms allows each agent to build its own local policy for executing its own tasks. In order to coordinate the agents' policies, we introduce an opportunity cost value function which allows each agent to value the consequences of his decisions on the other agents.

The paper is structured as follows. First, we detail existing works for solving DEC-MDPs and we provide a formal definition of the problem we deal with. Next, we define the OC-DEC-MDP model and we detail its components. We introduce the notion of opportunity cost and a modified Bellman equation to make coordinated decision. This decision principle is then use to develop polynomial algorithms to solve OC-DEC-MDPs. Finally, experimental results describe the performance of our approach. To conclude, we discuss the contributions of this work and we give some future research directions.

2 Decentralized control in multiagent systems

Decentralized control of a cooperative multiagent system consists of a set of agents where each agent must decide of its own actions so as to maximize a common performance measure. As the agents often have a partial view of the system, the states of the system and the other agents' states may be partially observable. Moreover, there may exist uncertainty about the other agents' actions and the evolution of the system.

2.1 Applicative domain

We motivate the class of problems that we consider by two multirobot domains: multi-rover exploration of Mars and rescue robots. These applications involve a set of robots that

must autonomously execute a set of tasks. While executing the tasks, each robot must autonomously decide how to act in order to maximize the team's performance. Moreover, communication between the robots are often impossible during task execution.

This class of problems can be met in NASA scenarios dealing with rovers operating on Mars. In order to increase scientific feedback, researchers plan to send fleets of robots that could operate autonomously and cooperatively. Since the environment is unknown, robots must deal with uncertainty regarding the duration of actions and consumption of resources. Once a day, a mission of a hundred tasks is sent to the robots via satellite. Due to orbital rotation of the satellite, the agents can communicate during a fixed amount of time. For the rest of the day, the rovers must complete their tasks and cannot communicate via the satellite which is unavailable. In addition, because of bandwidth limitations, distance between the robots and obstacles, the robots are unable to communicate directly. To guarantee scientific feedback, temporal constraints must be respected. For example, pictures must be taken at sunset or sunrise because of illumination constraints. Moreover, each robot may have specific skills imposing precedence constraints. For instance, as digging a hole modifies the ground topology of a site, a photograph-robot must have taken a picture of the site before a digger-robot can start to dig the ground. Finally, executing a task requires power, storage (storing pictures or measurements) or bandwidth (data communication). Resource constraints must therefore be respected: each agent must have enough of the required resources to complete a task. Military domains describe similar problems: UAV exploration of enemy territories share many characteristics (uncertainty, partial observability,...) with Mars exploration. Here, the agent cannot communicate because of the high cost of revealing information to the enemy.

Our second motivating example is the problem of controlling large teams of rescue robots (Morimoto, 2000; The RoboCup Rescue Technical Committee, 2000). These involve different kinds of robots that must act in disaster environments, for instance, after an earthquake occurs. Each kind of robots has special skills: fire-agents can extinguish fires, police-agents can unblock roads and ambulance-agents can give first aid to injured persons and drive them to hospital. The agents' objective is to maximize the number of rescued persons and to minimize the area of burned buildings. This objective can be defined as a common value function to maximize. As communications often breakdown in such disaster environments, it is assumed that the agents cannot communicate. Moreover, the environment is unknown and partially observable so, the agents must deal with uncertainty in action outcome and with uncertainty about the state of the system. Because resources are limited, the agents must consider resource constraints (for instance, fire-agents have a limited amount of water). Dependencies between the agents arise from their skills and precedence constraints can be identified. For example, fire-agents must have unblocked roads in order ambulance-agents to drive injured person to hospital. Moreover, temporal constraints must be respected: a fire must be extinguished before the building is completely burnt.

2.2 Problem statement

Based on the previous study of multi-robot decision problems, we define a mission $\mathcal{X}$ as a set of agents that must complete a set of tasks.

Definition 1 *A **mission** $\mathcal{X}$ is defined as a couple $< \mathcal{A}g, \mathcal{T} >$ where:*

- $\mathcal{A}g = \{\mathcal{A}g_1, \cdots, \mathcal{A}g_n\}$ *is a set of n agents $\mathcal{A}g_i \in \mathcal{A}g$.*
- $\mathcal{T} = \{t_1, \cdots, t_p\}$ *is a set of tasks the agents $\mathcal{A}g_i \in \mathcal{A}g$ have to execute.*

Each time an agent finishes executing a task with respect to constraints, it obtains a reward which depends on the executed task. The essence of the problem is to find a joint policy that maximizes the sum of the rewards of the agents.

The execution of a task is uncertain and must respect temporal, precedence and resource constraints. Thus, each task is assigned a probabilistic set of durations, a probabilistic set of resource consumptions, a set of predecessors and a temporal window.

Definition 2 *Each **task** $t_i \in \mathcal{T}$ is defined by:*

- *An **agent** $\mathcal{A}g_i$ that must execute the task.*
- ***Different possible durations** associated with **probabilities**. $P_c(\delta_c^i)$ is the probability the execution of the task t_i takes δ_c^i time units.*
- ***Different possible resource consumptions** associated with **probabilities**. $P_r(\Delta_r^i)$ is the probability the execution of the task t_i consumes Δ_r^i resources.*
- *A **temporal window** $TC_i = [EST_i, LET_i]$ during which the tasks must be executed. EST_i stands for the Earliest Start Time of the task and LET_i is its Latest End Time.*
- *A **set of predecessor tasks** $Pred_i$: the tasks that must be finished before t_i can start.*

$$\forall t_i \in \mathcal{T}, t_i \notin root \;\; \Leftrightarrow \exists t_j \in \mathcal{T} : t_j \in Pred(t_i)$$

  where $root$ are the first tasks to be executed (tasks without predecessors).
- *A **reward** $\mathcal{R}_i$ obtained by the agent when the task t_i has been executed respecting temporal, precedence and resource constraints. The agents aim at maximizing the sum of their obtained rewards.*

Each agent have to execute at least one task and each task is assigned an agent. Task allocation must take into account each agent's skills and must result in a feasible mission. For instance, given a temporal window $TC_i = [EST_i, LET_i]$, there must be at least one possible interval of execution for the task t_i which respects precedence constraints. Due to temporal constraints, precedence constraints and properties of the problem (such as the topology of the environment, distance between the sites to visit, etc.), tasks can often be easily ordered. Hanna and Mouaddib (2002), and more recently Abdallah and Lesser (2005), have developed MDP based algorithms that can perform such an allocation. Allocation of tasks among physical robots have also been studied by Gerkey and Matarić (2002) and Esben et al. (2002) using auction principles.

A mission $\mathcal{X}$ can be represented as an acyclic graph of tasks where edges stand for precedence constraints. Figure 1 presents the graph of a mission derived from RoboCup Rescue scenarios. A similar problem has been described by Marecki and Tambe (2007). Although we can represent large mission graphs involving hundreds of tasks, we describe a small mission for clarity reason. This scenario involves three agents (a fire-agent, an ambulance-agent and a police-agent) that have to rescue civilians from a burning building, after an earthquake occurs. Dependencies between the tasks can be identified. So as to evacuate civilians, the fire-agent must have put out the flames. Civilians have to be evacuated and roads must be unblocked in order the ambulance-agent to drive injured people to hospital. Moreover, temporal constraints have to be respected: it can be estimated that the fire-agent has 10 minutes to evacuate civilians and 30 minutes before the building is entirely burned. Finally, each agent has a limited amount of resources.

A similar representation language - multi-agent influence diagrams (MAIDs) - is described by Koller and Milch (2003) and applied to multi-agent games. One could view our graph representation as a MAID where decision variables are tasks and without chance variables. Utility variables would be the rewards of the tasks. Precedence constraints between

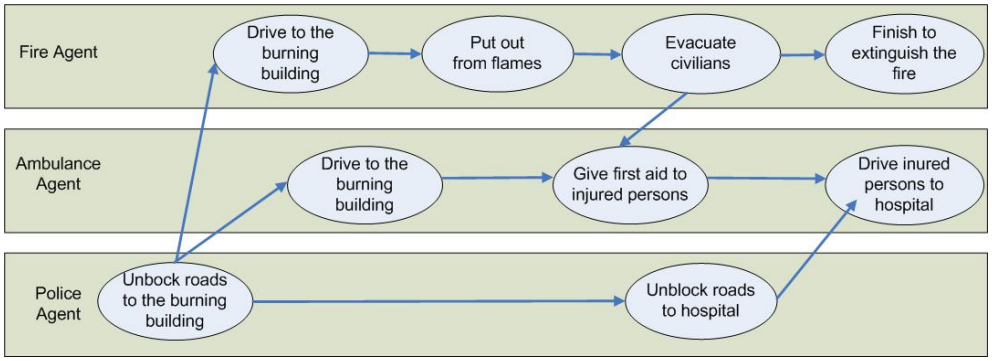

Fig. 1 A mission graph involving rescue agents

the tasks can be formalized in a MAID by probabilistic dependencies between decision variables. Thus, MAID edges would represent precedence constraints. By extending MAID to account for uncertainty on task execution (captured by conditional probabilities and random variables) and temporal constraints (encoded with the utility function), we obtain a similar representation to our mission graphs.

Precedence constraints that associate different kinds of agents, stand for coordination relationships. Such relationships are encountered in other frameworks such as TAEMS - Task Analysis, Environment Modeling and Simulation - (Decker and Lesser, 1993) which allows to describe task structures of multiagent systems. TAEMS "enable" relationship stands for our precedence constraints. Nonetheless, TAEMS framework assumes that resource are not consumed when the execution of a task partially fails. Moreover, TAEMS framework allows precedence constraints not to be met.

Given a mission $\mathcal{X}$, our purpose is to plan the execution of the mission so as to allow each agent to decide which task to execute and when, without communicating (during task execution) and with respect to constraints. We aim at developing an off-line planning approach based on DEC-MDPs.

2.3 Related work

In order to limit on-line computation and to deal with uncertainty, we are interested in off-line stochastic planning. Some classical planning approaches, such as STRIPS or GPS, have been adapted for planning under uncertainty (Blythe, 1999a; Bresina et al., 2002). Most of these approaches search for a plan that meets a threshold probability of success or that exceeds a minimum expected utility. During task execution, if the agent deviates from the computed plan, a new plan has to be re-computed. To limit re-planning, some approaches compute a contingent plan that encodes a tree of possible courses of actions. Nonetheless, a contingent plan may not consider all possible courses of actions so, re-planning remains and optimality is not guaranteed.

Markov Decision Processes (MDP) provide a stochastic planning approach that allows for computing optimal policies. As a policy maps each possible state of the agent to an action, there is no need for on-line re-planning. The agent's objectives are expressed as a utility function and efficient algorithms have been developed to efficiently compute a policy that maximizes the utility (Puterman, 2005; Howard, 1960). MDPs have been successfully

applied to many domains such as mobile robots (Bernstein et al., 2001), spoken dialog managers (Roy et al., 2000) or inventory management (Puterman, 2005). Then, MDPs have been extended to deal with multiagent settings and Decentralized Partially Observable Markov Decision Processes (DEC-POMDP) (Bernstein et al., 2002) have been defined.

2.3.1 Decentralized Markov Decision Processes

So as to modelize partial observability and uncertainty, the DEC-POMDP model is composed of a set of observations, a probabilistic observation function and a probabilistic transition function. A reward function to maximize formalizes the objectives of the system.

Definition 3 *A **Decentralized Partially Observable Markov Decision Process (DEC-POMDP)** for n agents is defined by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \Omega, \mathcal{O}, \mathcal{R} >$ where :*

- *$\mathcal{S}$ is a finite set of system states.*
- *$\mathcal{A} = \langle \mathcal{A}_1, \cdots, \mathcal{A}_n \rangle$ is a set of joint actions, $\mathcal{A}_i$ is the set of actions a_i that can be executed by the agent $\mathcal{A}g_i$.*
- *$\mathcal{P} = \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition function. $\mathcal{P}(s, a, s')$ is the probability of the outcome state s' when the agents execute the joint action a in s.*
- *$\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$ is a finite state of observations where Ω_i is agent $\mathcal{A}g_i$'s set of observations.*
- *$\mathcal{O} = \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \Omega \rightarrow [0, 1]$ is the observation function. $\mathcal{O}(s, a, s', o = \langle o_1, \cdots, o_n \rangle)$ is the probability that each agent $\mathcal{A}g_i$ observes o_i when the agents execute the joint action a from state s and the system moves to state s'.*
- *$\mathcal{R}$ is a reward function. $\mathcal{R}(s, a, s')$ is the reward the system obtains when the agents execute joint action a from state s and the system moves to state s'.*

Definition 4 *A **Decentralized Markov Decision Process (DEC-MDP)** is a special kind of DEC-POMDPs where the system state is jointly observable. This property can be formalized as:*

$$\textit{If } \mathcal{O}(s, a, s', o = \langle o_1, \cdots, o_n \rangle) > 0 \textit{ then } Prob(s' | \langle o_1, \cdots, o_n \rangle) = 1$$

Note that this property does not imply that the agents can observe their states. Thus, a DEC-MDP has the same components as a DEC-POMDP.

2.3.2 DEC-POMDP resolution

Solving a DEC-MDP consists in computing a set of individual policies. Each individual policy of an agent $\mathcal{A}g_i$ takes into account every possible state of the agent while methods based on classical planners find a sequence of actions based on a set of possible initial states (Blythe, 1999b). Recent works have focused on developing off-line planning algorithms to solve problems formalized by DEC-POMDPs and DEC-MDPs. They consist in computing a set of individual policies, one per agent, describing the agents' behaviors. Each individual policy maps the agent's information (its state, its observations or its belief state) to an action. Since solving optimally a DEC-POMDP (or a DEC-MDP) is a very hard problem (NEXP-hard) (Bernstein et al., 2002), most approaches search for methods that reduce the complexity of the problem. Two kinds of approaches can be identified. The first set of approaches aims at identifying properties of DEC-POMDPs that reduce their complexity. Thus, Goldman and Zilberstein (2004) have introduced transition independence and observation independence.

Definition 5 *A **transition independent** DEC-MDP is such that the probability an agent $\mathcal{A}g_i$ moves from a state s_i to a state s'_i only depends of the action a_i the agent has executed. Formally, a DEC-MDP is transition independent if the set of states $\mathcal{S}$ can be decomposed into n components $\mathcal{S}_1, \cdots, \mathcal{S}_n$ and the transition function can be decomposed as a product of probabilities such as: $\mathcal{P} = \prod_{i=1}^{n} \mathcal{P}_i$ where $\mathcal{P}_i = Pr(s'_i|s_i, a_i)$.*

Definition 6 *A DEC-MDP is observation independent if the set of states $\mathcal{S}$ can be decomposed into n components $\mathcal{S}_1, \cdots, \mathcal{S}_n$ and there exists, for each agent $\mathcal{A}g_i$, an observation function $\mathcal{O}_i$ such as:*

$$Pr(o_i|\langle s_1, \cdots, s_n\rangle, \langle a_1, \cdots, a_n\rangle, \langle s'_1, \cdots, s'_n\rangle, \langle o_1, \cdots, o_{i-1}, o_{i+1}, \cdots, o_n\rangle) = \mathcal{O}_i(o_i|s_i, a_i, s'_i)$$

Properties such as transition or observation independence allows for identifying classes of problems that are easier to solve (Goldman and Zilberstein, 2004). For instance, it has been proved that a DEC-MDP with independent transitions and observations is NP-complete. Based on this study, an optimal algorithm, the Coverage Set Algorithm (CSA), has been developed to solve DEC-MDPs with independent observations and transitions (Becker et al., 2003).

Other attempts to solve DEC-POMDPs have focused on finding approximate solutions instead of computing the optimum. Thus, alternatives to Hansen's exact dynamic programming algorithm (Hansen et al., 2004) have been proposed by Bernstein et al. (2005) and Amato et al. (2007b). They use memory bounded controllers to limit the required amount of space to solve the problem. Nair et al. (2003) describe an approach, the Joint Equilibrium Based Search for Policies (JESP), to solve transition and observation independent DEC-MDPs. JESP relies on co-alternative evolution: the policies of a set of agents are fixed and the policies of the remaining agents are improved. Policy improvement is executed in a centralized way and only a part of the agents' policies is improved at each step. Finally, the algorithm converges to a Nash equilibrium. Chadès et al. describe a similar approach based on the definition of subjective MDPs and the use of empathy (Chadès et al., 2002). Improvements of JESP have also been proposed: DP-JESP (Nair et al., 2003) speeds up JESP algorithm using dynamic programming and LID-JESP (Nair et al., 2005) combines JESP and distributed constraints optimization algorithms. Thus, LID-JESP exploits the locality of interactions to improve the efficiency of JESP. SPIDER (Varakantham et al., 2007) also exploits the locality of interactions to compute and approximate solution. Moreover, SPIDER uses branch and bound search and abstraction to speed up policy computation. Peshkin et al. (2000) propose a distributed learning approach based on gradient descent method that also allows finding a Nash equilibrium. Emery-Montemerlo et al. (2004) approximate DEC-POMDP solutions by decomposing the initial problem into local MDPs which are solved separately. The problem is transformed into a set of one-step Bayesian games that are solved using a heuristic.

Finally, some approaches introduce direct communication so as to increase each agent observability (Goldman and Zilberstein, 2003; Xuan et al., 2001; Pynadath and Tambe, 2002). The agents can communicate to inform the other agents of their local state or observation. If communication is free and instantaneous, the problem is reduced to a Multiagent Markov Decision Process (MMDP) (Boutilier, Dean, and Hanks, 1999) that is easier to solve. Otherwise, the problem complexity remains unchanged and heuristic methods are described to find near optimal policies.

2.3.3 Existing Work limitations

Despite recent advances in DEC-POMDP domain, solving multi-robot decision problems using DEC-POMDPs (or DEC-MDPs) remains a serious challenge. Indeed, recent works have focused on problems, like the multiagent tiger problem (Nair et al., 2003), that do not take into account properties of real-world problems. Thus, DEC-POMPs fail to modelize precedence and temporal constraints. Moreover, each action is assumed to have the same duration of one time unit and uncertainty on action duration cannot be formalized. Such difficulties to formalize real-world problem using Markovian models have been identified in the single agent case by Bresina et al. (2002). In order to increase the expressiveness of DEC-MDPs, Becker et al. (2004a) have introduced Event-Driven Decentralized Markov Decision Processes (ED-DEC-MDPs) that allow for modeling precedence and temporal constraints. But this model remains limited to small problems.

Other difficulties arise while solving DEC-POMDPs (and DEC-MDPs). Due to the complexity of optimally solving DEC-POMDPs (Bernstein et al., 2002), there is no existing optimal approaches that can solve problems involving more than 2 agent and 5 actions. For instance, Hansen et al. (2004) optimally solve a two-agent and two-action decision problem up to horizon 4. Szer et al. (2005) optimally solve the same problem up to horizon 5. Approximate approaches are able to solve larger problems but remains limited to 2 agents and less than 10 actions. Nair et al. (2003) solves the multi-agent tiger problem (2 agents and 3 actions) up to horizon 7. More recently, Seuken and Zilberstein (2007) and Amato et al. (2007a) have proposed approaches that can solve this problem up to horizon 100. Nonetheless, their approaches consider the classical DEC-POMDP model thus, limiting time and action representation. It is therefore difficult to solve large problems like the ones encountered in real-world domains. Indeed, existing Mars rover missions are composed of 2 agents but researchers plan to send more rovers. Thus, planning approach must be able to consider missions that are composed of 3 or more robots which have to execute a hundred of tasks with complex contraints and dependencies. Rescue missions may involve more than ten robots and hundreds of tasks.

In order to increase the applicability of DEC-POMDP approaches, there is a need to improve the problem modelization and to develop efficient tools to solve large problems. In this paper, we first present a model that improves time and action modelizations in DEC-MDPs. Then, we turn to problem resolution and we tackle the high complexity of DEC-MDPs. We propose an efficient planning approach that computes each agent's policy even for large missions.

3 Formal model for constraints representation

In this section, we focus on improving time and action representations in DEC-MDPs. We therefore define a class of DEC-MDPs, Opportunity Cost Decentralized Markov Decision Processes (OC-DEC-MDPs), that allows us to consider several possible durations for each task taking into account temporal, resource and precedence constraints.

3.1 The OC-DEC-MDP model

As we consider problems where each action has several possible durations, we cannot define a joint action as the set of the agents' individual actions. Indeed, at each time step t, some

agents try to execute new actions and the others continue executing their previously started actions. The transition probability therefore relies on the time each agent has already spent to execute its current action. In order to fulfill Markov property (Puterman, 2005), joint actions can be defined as a set of couples (a_i, Δ_c^i) where a_i is the current action of $\mathcal{A}g_i$ and Δ_c^i the time $\mathcal{A}g_i$ has already spent executing a_i. Nonetheless, defining such joint actions leads to large state and action spaces which are exponential in the number of agents and in the number of time steps. This limits the size of problems that can be considered.

In order to allow for representing large problems, we suggest splitting the initial multiagent decision problem into a set of MDPs that are easier to solve. Each MDP stands for one agent's decision problem. Thus, individual states and actions are considered and exponential growth of joint action set and state set is avoided. Decomposition of planning problems into smaller problems has been widely studied in the single agent case. Techniques have been proposed to decompose large Markov decision processes into local MDPs that are easier to solve (Dean and Lin, 1995; Boutilier et al., 1997). These local MDPs are then solved and exploited to find good approximate global solutions (Meuleau et al., 1998; Singh and Cohn, 1998; Poupart et al., 2002). Nonetheless, decomposition of decentralized Markov decision problems into local MDPs are lacking. We introduce the OC-DEC-MDP framework that performs such a decomposition and proposes a richer model of time and action formalizing previously enumerated constraints. Moreover, this framework allows for limiting the state and actions spaces of the agents.

Definition 7 *An n-agent **OC-DEC-MDP** is a set of n MDPs, one for each agent. The MDP of an agent $\mathcal{A}g_i$ is defined as a tuple $< \mathcal{S}_i, \mathcal{T}_i, \mathcal{P}_i, \mathcal{R}_i >$ where:*

- *$\mathcal{S}_i$ is the finite set of states of the agent $\mathcal{A}g_i$,*
- *$\mathcal{T}_i$ is the finite set of tasks of the agent $\mathcal{A}g_i$,*
- *$\mathcal{P}_i$ is the transition function of the agent $\mathcal{A}g_i$,*
- *$\mathcal{R}_i$ is the reward function of the agent $\mathcal{A}g_i$.*

The components of each MDP are defined in order to model constraints on task execution. We now review each of these components.

3.1.1 Tasks - Actions

At each of his decision steps, the agent $\mathcal{A}g_i$ must decide when to start his next task. The actions to perform consist of *"Executing task t_i at time st : $E(t_i, st)$"*, that is the action to start executing task t_i at time st. We assume that each agent's tasks are completely ordered. So the next task to execute is easily decided. The question is therefore to decide when the agent must start its next task and the set $\mathcal{A}$ of actions to execute from a state s_i is equal to the set of start times for the next task.

Nonetheless, this assumption can be relaxed to consider situations where the agent must choose between several possible next tasks. Then, the number of actions that can be executed from a state s_i increases. In fact, it is exponential in the numbers of possible next tasks and start times.

Start times and end times of each task can be deduced by propagating temporal constraints through the mission graph from the roots to the leaves. The graph is organized into levels such that : L_0 contains the roots of the graph, L_1 contains all the successors of the roots (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, if all t_i's predecessors have already been considered, we compute its possible start times and end times. The start times of t_i are in $[EST_i, LST_i]$

where LST_i is t_i's Latest Start Time which is defined by $LST_i = LET_i - \delta_c^{i,min}$ ($\delta_c^{i,min}$ is the shortest duration of t_i). However, $\mathcal{A}g_i$ 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 $[EST_i, LST_i]$. 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, its start times and end times are computed by:

• *level* L_0: The start times and the end times of each root node t_i are computed as follows:

$$ST(t_i) = max(\{EST_i, start\_time\}) = LB_i = UB_i$$

$$ET(t_i) = \{st(t_i) + \delta_c^i \leq LET_i\}$$

where δ_c^i is the computation time of t_i and $start\_time$ is the system "wake up" time. Consequently, intervals of activities of t_i are given by $I = [st, et]$, where $st \in ST(t_i)$ and $et = st + \delta_c^i, et \leq LET_i$.

• *level* L_i: Each node t_i in level L_i can start its execution if, and only if, all its predecessors have finished 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:

$$FET_i = \min(ET(pred))$$

Then, the maximum of these first end times represents the first possible start time for the node t_i:

$$LB_i = \max(\{EST_i, \max(FET_i)\})$$

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. Let *pred* be a predecessor of t_i. Any end time et of *pred* such as $et > LB_i$ is a possible start time of the node because it represents a situation where all the other predecessors have finished before LB_i and *pred* has finished after LB_i at et. Consequently, the other possible start times are defined as follows:

$$OST_i = \{et \in ET(pred), et > LB_i\}$$

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

$$ST(t_i) = \{LB_i \bigcup OST_i\}$$

and

$$UB_i = max(ST(t_i))$$

The set of the possible end times of the node is then given by:

$$ET(t_i) = \{et | et = st + \delta_c^i, et < LET_i \text{ and }, \forall st \in ST(t_i)\}$$

Finally, the intervals of execution I of the node are such as:

$$I = [st \in ST(t_i), st + \delta_c^i \leq LET_i]$$

3.1.2 States

When an agent tries to execute a task t_{i+1} at st, it succeeds if temporal, precedence and resource constraints are respected. If precedence constraints are violated (the agent tries to execute t_{i+1} before its predecessors have finished their execution), the agent fails and can retry to execute the task later. Then, if precedence constraints are respected while retrying to execute t_{i+1}, the agent may succeed. Failures due to precedence constraints are therefore called partial failures. On the other hand, if temporal or resource constraints are violated while executing t_{i+1}, the execution of t_{i+1} fails permanently and the agent moves to a failure state. Three kinds of states can therefore be identified: success states, partial failure states and failure states.

When an agent $\mathcal{A}g_i$ has just finished executing task t_{i+1} during an interval I and it has $r_{t_{i+1}}$ remaining resources, it moves to the success state $[t_{i+1}, I, r_{t_{i+1}}]$.

When the agent has just tried to execute task t_{i+1} at st but failed because of precedence constraints, it moves to the partial failure state $[t_i, [st, st+1], et(I')_{t_i}, r_{t_i}]$ where t_i is the agent's last successfully executed task, $et(I')_{t_i}$ is the end time of t_i (I' is t_i's execution interval), and r_{t_i} is the agent's remaining resources after it partially fails. As the end times $et(I')$ of t_i influences the agent's transition, it is registered in failure states. 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.

When the execution of t_{i+1} fails permanently, the agent moves to the failure state associated with t_{i+1}.

As each agent knows his last executed task t_{i+1}, the interval of execution of t_{i+1} and how many resources he still has, states are locally fully observable. Nonetheless, each agent does not know the other agents' states nor actions. The problem is therefore for the agents to coordinate despite this lack of knowledge about the system's state. Note that the system's states can be deduced from combining the agents' partial views and an OC-DEC-MDP is jointly fully observable. We therefore consider DEC-MDPs instead of DEC-POMDPs.

3.1.3 Transitions

The action of an agent allows him to move from one state to another. Given a state s_i of an agent $\mathcal{A}g_i$, the agent can move to three different kinds of states. Thus, we identify three kinds of transitions: success transitions (from s_i to a success state), partial failure transitions (from s_i to a partial failure state) and failure transitions (from s_i to a failure state). Figure 2 represents the different kinds of possible transitions when an agent starts to execute a task t_{i+1} at st from a state s_i.

Due to precedence constraints between the agents, the problems we consider are transition dependent. Indeed, the probability of an agent's success in executing a task t_{i+1} depends on the end times of t_{i+1}'s predecessors which may be executed by other agents. Thus, the transition of an agent relies on the other agents' actions and states. Because our problems are not transition independent, the transition function of the system cannot be directly decomposed into local independent functions. Next section will detail our approach for computing one individual transition function per agent.

3.1.4 Rewards

When agent $\mathcal{A}g_i$ successfully executes a task t_{i+1}, it moves to a success state and receives the reward associated with t_{i+1}. When an agent fails executing a task (partial failure or

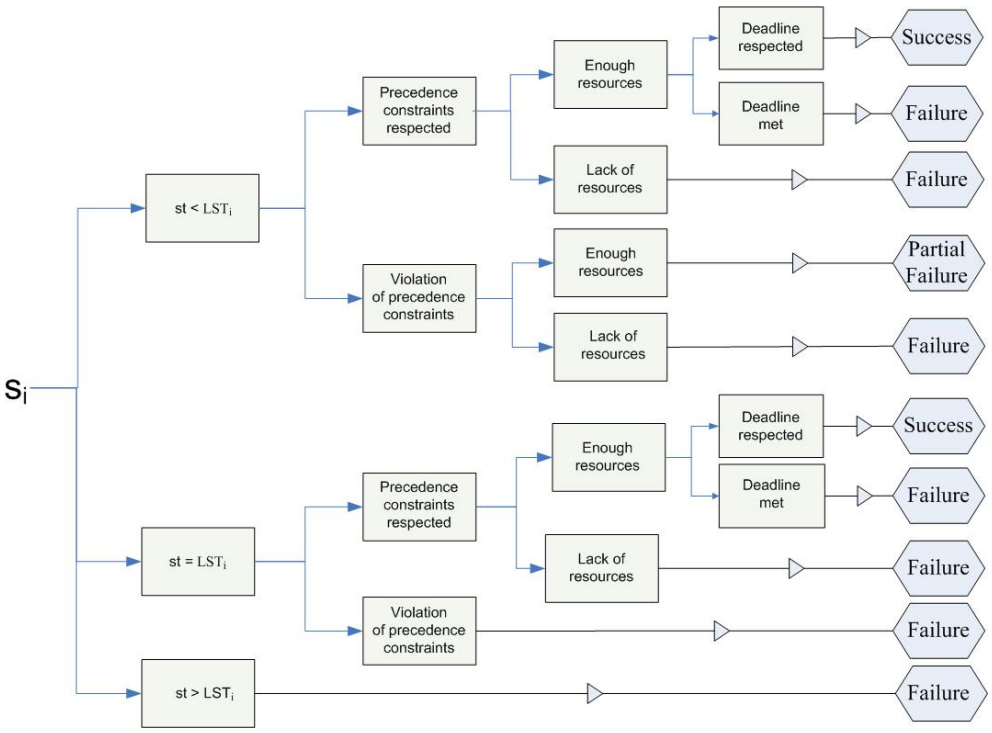

Fig. 2 Transition model of an agent from state s_i

permanent failure), no reward is obtained. Thus, the agents' goal consists in maximizing the sum of their obtained rewards.

3.2 Transition Decomposition

In this section, we describe a method for decomposing the transition function of the system ($\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A}$) into a set of individual transition functions. Each individual transition function is associated with an agent and dictates the probability that the agent moves from one state to another when he executes an individual action ($\mathcal{P}_i : \mathcal{S}_i \times \mathcal{S}_i \times \mathcal{A}_i$). Actions are probabilistic since the processing time and the resource consumption of each task are uncertain. Moreover, as precedence constraints must be respected, the probability an agent $\mathcal{A}g_i$ moves from a state s_i to a state s_i' when it starts executing t_{i+1} at st relies on: the executed task t_{i+1}, the start time of t_{i+1}, the end times of t_{i+1}'s predecessors, the resource consumption of t_{i+1}, the available resources before the agent $\mathcal{A}g_i$ starts executing t_{i+1}, and the duration of t_{i+1}. Moreover the start times of each task t_{i+1} relies on t_{i+1}'s execution policy.

Probabilities on start times, end times and available resources have to be known in order to compute transition probabilities. As these probabilities relies on the agents' policies, we assume an initial set of policies for the agents (one policy per agent) and we propagate temporal and resource probabilities through the mission graph. For each node t_i, temporal

and resource probabilities are computed based on probabilities on t_i's predecessors, the execution policy of t_i and probabilities on execution time and resource consumption of t_i.

3.2.1 Temporal probabilities

The probability that agent $\mathcal{A}g_i$ starts executing a task t_{i+1} at st depends on the end times of t_{i+1}'s predecessors, on the state of $\mathcal{A}g_i$ and on $\mathcal{A}g_i$'s policy. Task t_{i+1}, can be executed from a partial failure state or from a success state. Let t_i be the last successfully executed task of $\mathcal{A}g_i$. $\mathcal{A}g_i$ can start executing t_{i+1} from a partial failure state $s_i = [t_i, [*, *], et(I')_{t_i}, r_{t_i}]$, or from a success state $s_i = [t_i, [, *, et(I')_{t_i}], r_{t_i}]$. The start time of t_{i+1} therefore depends on the agent's policy from s_i. $\mathcal{A}g_i$ starts executing t_{i+1} at st if:

- his policy dictates him to start t_{i+1} at st and the predecessors of t_{i+1} have finished their execution or,
- his policy dictates him to start t_{i+1} at st' ($st' < st$), $\mathcal{A}g_i$ partially fails (t_{i+1}'s predecessors have not finished their execution) and after 0 to N partial failures he starts executing t_{i+1} at st.

Thus, the probability $P_{\mathcal{ST}}^{t_{i+1}}(st|et(I')_{t_i}, r_{t_i})$ that the agent starts executing t_{i+1} at st, when it has r_{t_i} resources and t_i ends at $et(I')_{t_i}$ is defined as follows:

$$
\begin{aligned}
P_{\mathcal{ST}}^{t_{i+1}}(st|et(I')_{t_i}, r_{t_i}) = & P^r_{enough}(Pred(t_{i+1})). \prod_{t_j \in P_{red}(t_{i+1})} \sum t \leq st(I) P_{\mathcal{ET}}^{t_j}(t|et(I')_{t_i}) \\
& \cdot \Big(P_{\pi_i}(st|t_i, et(I')_{t_i}, r_{t_i}) + \sum_{st' \in [et(I'), st[}^{st} P_{\pi_i}(st', task_{i+1}|t_i, et(I')_{t_i}, r_{t_i}) \\
& \cdot P_{not\_end}(st') \cdot P_{\mathcal{ST}}^{s_i}(st, s_i = [t_i, [st, st+1], et(I')_{t_i}, r_{t_i} - \Delta r']) \Big)
\end{aligned} \tag{1}
$$

where $P_{not\_end}(st')$ stands for the probability that the predecessors of t_{i+1} have not finished their execution at st'. As described later, $P_{\mathcal{ET}}^{t_j}(t|et(I')_{t_i})$ is the probability t_j ends at t. $P_{\pi_i}(st', task_{i+1}|t_i, et(I')_{t_i}, r_{t_i})$ is the probability the agent $\mathcal{A}g_i$ decides to execute t_{i+1} at st' when t_i finishes at $et(I')_{t_i}$ and the agent has r_{t_i}. Since we consider deterministic policies, this probability is 0 or 1. $\Delta r'$ stands for the resource consumption of a partial failure. For purpose of good understanding, we consider only deterministic resource consumptions of partial failures. Nonetheless, equations can be extended to deal with uncertainty on partial failures' resource consumptions.

The probability that t_{i+1}'s predecessors have finished their execution at st is:

$$
P^r_{enough}(Pred(t_{i+1})) \cdot \prod_{t_j \in P_{red}(t_{i+1})} \sum t \leq st(I) P_{\mathcal{ET}}^{t_j}(t|et(I')_{t_i})
$$

where $P^r_{enough}(Pred(t_{i+1}))$ stands for the probability that the predecessors had enough resources.

As described by Equation 1, we first consider cases where the agent's policy dictates to start t_{i+1} at st ($P_{\pi_i}(st|t_i, et(I'), r_{t_i})$). Then, we consider cases where the agent's policy

dictates to start the task before st, the agent partially fails and then starts executing the task at st.

$P^{s_i}_{\mathcal{ST}}(st, s_i)$ is the probability that the agent will start the execution of t_{i+1} at st when the agent is in state $s_i = [t_i, [st, st+1], et(I')_{t_i}, r_{t_i} - \Delta r']$. Before the agent starts at st, it may try to execute the task and partially fail. Thus, we obtain:

$$P^{s_i}_{\mathcal{ST}}(st(I), s_i) = P_{\pi_i}(st|t_i, st+1, et(I')_{t_i}, r_{t_i} - \Delta r') + \sum_{st' \in [st+1, st]} P_{\pi_i}(st', task_{i+1}|t_i, [st, st+1], et(I')_{t_i}, r_{t_i} - \Delta r') \cdot P_{not\_end}(st') \cdot P^{s_i}_{\mathcal{ST}}(st, s_i = [t_i, [st', st'+1], et(I')_{t_i}, r_{t_i} - \Delta r' - \Delta r'])$$

where $P_{\pi_i}(st', task_{i+1}|t_i, [st, st+1], et(I')_{t_i}, r_{t_i} - \Delta r')$ is the probability the agent $\mathcal{A}g_i$ decides to execute t_{i+1} at st' when t_i finishes at $et(I')_{t_i}$ and the agent partially fails the execution of t_{i+1} at st. Then, the agent realizes that it fails at $st+1$ and it has $r_{t_i} - \Delta r'$ resources.

Finally, $P^r_{enough}(Pred(t_{i+1}))$ is defined as:

$$P^r_{enough}(Pred(t_{i+1})) = \prod_{t_k \in Pred(t_{i+1}) \setminus t_i} P^r_{enough}(t_k)$$

where $P^r_{enough}(t_k)$ stands for the probability agent $\mathcal{A}g_k$ had enough resources to execute t_k. This probability is computed as follows:

- If t_k is the last task that agent $\mathcal{A}g_k$ has to execute:

$$P^r_{enough}(t_k) = \sum_{r_{t_k} \geq 0} \sum_{\Delta^k_r | r_{t_k} - \Delta r \geq 0} P^{t_k}_{ra}(r_{t_k}) \cdot P_r(\Delta^k_r)$$

  where $P^{t_k}_{ra}(r_{t_k})$ is the probability $\mathcal{A}g_k$ has r_{t_k} resources when it starts to execute t_k. We detail computation of this probability in the next section.
- Otherwise:

$$P^r_{enough}(t_k) = \sum_{r_{t_{k+1}} \geq 0} P^{t_{k+1}}_{ra}(r_{t_{k+1}})$$

  where t_{k+1} is the next task of agent $\mathcal{A}g_k$.

From conditional probabilities $P^{t_{i+1}}_{\mathcal{ST}}(st|et(I')_{t_i}, r)$, we can deduce conditional probabilities $P^{t_{i+1}}_{\mathcal{ST}}(st|et(I')_{t_i})$ and probabilities $P^{t_{i+1}}_{\mathcal{ST}}(st)$. Thus,

$$P^{t_{i+1}}_{\mathcal{ST}}(st|et(I')_{t_i}) = \sum_{r_{t_i}} P^{t_i}_{ra}(r_{t_i}).P^{t_{i+1}}_{\mathcal{ST}}(st|et(I')_{t_i}, r_{t_i}) \quad (2)$$

We deduce:

$$P^{t_{i+1}}_{\mathcal{ST}}(st) = \sum_{et(I')_{t_i} \in ET(t_i)} P^{t_i}_{\mathcal{ET}}(et(I')_{t_i}) \sum_{r_{t_i}} P^{t_i}_{ra}(r_{t_i}).P^{t_{i+1}}_{\mathcal{ST}}(st|et(I')_{t_i}, r_{t_i}) \quad (3)$$

Once start time probabilities of t_{i+1} are known, end time probabilities $P^{t_{i+1}}_{\mathcal{ET}}$ can be deduced. The probability the execution of t_{i+1} ends at et is the probability that it starts at st and its execution takes $\delta^{i+1}_c = et - st$ time units:

$$P^{t_{i+1}}_{\mathcal{ET}}(et) = \sum_{st \in ST(t_i)} \sum_{\delta^{i+1}_c | st + \delta^{i+1}_c = et} P^{t_{i+1}}_{\mathcal{ST}}(st) \cdot P_c(\delta^{i+1}_c) \quad (4)$$

Moreover:

$$P^{t_{i+1}}_{\mathcal{ET}}(et|et(I')_{t_i}, r_{t_i}) = \sum_{st \in ST(t_i)} \sum_{\delta_c^{i+1}|st+\delta_c^{i+1}=et} P^{t_{i+1}}_{\mathcal{ST}}(st|et(I')_{t_i}, r_{t_i}) \cdot P_c(\delta_c^{i+1}) \quad (5)$$

And,

$$P^{t_{i+1}}_{\mathcal{ET}}(et|et(I')_{t_i}) = \sum_{st \in ST(t_i)} \sum_{\delta_c^{i+1}|st+\delta_c^{i+1}=et} P^{t_{i+1}}_{\mathcal{ST}}(st|et(I')_{t_i}) \cdot P_c(\delta_c^{i+1}) \quad (6)$$

3.2.2 Resource probabilities

Let $P^{t_{i+1}}_{ra}(r_{t_{i+1}})$ be the probability that agent $\mathcal{A}g_i$ has $r_{t_{i+1}}$ resources before it tries to execute t_{i+1}. As agents do not share resources, this stands for the probability that $\mathcal{A}g_i$ has $r_{t_{i+1}}$ resources after executing t_{i+1}'s previous task (let t_i be this previous task). Probabilities P_{ra} on available resources are therefore defined as follows:

1. **If t_{i+1} is the first task of the agent ($t_i = \emptyset$) :**

$$P^{t_{i+1}}_{ra}(r_{t_{i+1}}) = \begin{cases} 0 \text{ If } r_{t_{i+1}} \neq R_{ini} \\ 1 \text{ Otherwise} \end{cases}$$

   where R_{ini} is $\mathcal{A}g_i$'s initial rate of resources.

2. **If t_{i+1} is the second task of the agent ($t_{i-1} = \emptyset$ and $t_i \neq \emptyset$)** (t_{i-1} is the task that $\mathcal{A}g_i$ has executed before t_i) :

$$P^{t_{i+1}}_{ra}(r_{t_{i+1}}) = \overbrace{\sum_{r_{t_i}} P^{t_i}_{ra}(r_{t_i})}^{\text{available resources before } t_i\text{'s execution}} \cdot \overbrace{\sum_{nbEP} P_{EP}(nbEP, r_{t_i})}^{\text{nr. of partial failures}} \cdot \overbrace{\sum_{\Delta_r^i | r_{t_i} - nbEP \cdot \Delta r' - \Delta_r^i = r_{t_{i+1}}} P_r(\Delta_r^i)}^{\text{resource consumptions}}$$

   Probabilities on available resources before trying to execute t_{i+1} are deduced from probabilities on available resources after the execution of t_i. These are based on available resources before trying to execute t_i, the number of partial failures of t_i, partial failure resource consumption $\Delta r'$, and t_i's resource consumption Δ_r^i. For each available resource rate before trying to execute t_i, we consider every possible number of partial failures. Let $P_{EP}(nbEP, r_{t_i})$ be the probability that $nbEP$ partial failures of t_i occur before successful execution of t_i, when the agent has r_{t_i} resources before trying to execute t_i.
   $P_{EP}(nbEP|r_{t_i})$ is computed by considering each possible start time st_i of t_i and the number of partial failures that may have occurred when the task successfully starts t_i at st_i. $EP(st_i)$ is the set of the numbers of partial failures $nbEP$ that may have occurred when the task is successfully executed at st_i. Then,

$$P_{EP}(nbEP, r_{t_i}) = \frac{\sum_{st \in ST(t_i)|nbEP \in EP(st)} P^{t_i}_{\mathcal{ST}}(st|r_{t_i}) \cdot \sum_{\delta_c^i | st+\delta_c^i \leq LET_i} P_c(\delta_c^{i+1})}{(1 - \sum_{st \in ST(t_i)} \cdot P^{t_i}_{\mathcal{ST}}(st)) \cdot (1 - \sum_{\delta_c^i | st+\delta_c^i \leq LET} P_c(\delta_c^i))}$$

   where $ST(t_i)$ is the set of possible start times st of t_i. For each of these start times st_i, we compute the probability that the task is successfully executed when it starts at st_i.

This is the probability that the task ends before the deadline ($\sum_{\delta_c^{i+1}|st_i+\delta_c^{i+1}\leq LET_{i+1}} P_c(\delta_c^{i+1})$). As we consider successful execution of t_i, the sum of probabilities is normalized by the probability that t_i is successfully executed.

3. **Otherwise**:

$$P_{ra}^{t_{i+1}}(r_{t_{i+1}}) = \overbrace{\sum_{r_{t_i}} P_{ra}^{t_i}(r_{t_i})}^{\text{available resources before } t_i\text{'s execution}} \cdot \overbrace{\sum_{et_{i-1}} P_{\mathcal{ET}}^{t_{i-1}}(et_{i-1}) \sum_{nbEP} P_{EP}(nbEP|et_{i-1}, r_{t_i})}^{\text{nr. of partial failures}}$$

$$\cdot \overbrace{\sum_{\Delta r_{t_i}|r_{t_i}-nbEP\cdot\Delta r'-\Delta_r^i=r_{t_{i+1}}} P_r(\Delta_r^i)}^{\text{resource consumptions}}$$

This equation is nearly the same as the one introduced in the previous case. Nonetheless, $\mathcal{A}g_i$ has executed a task t_{i-1} before t_i. As the end time et_{i-1} of t_{i-1} influences the start time of t_{i-1}'s following tasks, it is taken into account while considering probabilities on t_i's start times. $P_{EP}(nbEP|et_{i-1}, r_{t_i})$ is then defined as follows:

$$P_{EP}(nbEP|et_{i-1}, r_{t_i}) = \frac{\sum_{st_i\in ST(t_i)|nbEP\in EP(st_i|et_{i-1})} P_{\mathcal{ST}}^{t_i}(st_i|et_{i-1}, r_{t_i})}{\beta} \cdot \frac{\sum_{\delta_c^{i+1}|st_i+\delta_c^{i+1}\leq LET_{i+1}} P_c(\delta_c^{i+1})}{\beta}$$

And

$$\beta = (1 - \sum_{st_i\in ST(t_i)} P_{\mathcal{ST}}^{t_i}(st_i|et_{i-1})) \cdot (1 - \sum_{\delta_c^{i+1}|st_i+\delta_c^{i+1}\leq LET_{i+1}} P_c(\delta_c^{i+1}))$$

where $EP(st_i|et_{i-1})$ is the set of numbers of partial failures that may occur before the execution of t_i successfully starts at st_i, when t_{i-1} ends at et_{i-1}.

3.2.3 Transition probabilities

Once temporal and resource probabilities are known, individual transition functions can be deduced. As mentioned previously, each individual transition function can be decomposed into three components that stand for each kind of transitions. Let $\mathcal{A}g_i$ be in a state s_i where it tries to execute t_{i+1} at st_{i+1}. We assume that his previous executed task t_i finished at $et(I')_{t_i}$ and $\mathcal{A}g_i$ has r_{t_i} remaining resources. We now detail how to compute individual transition probabilities for each kind of transitions.

Success transitions When agent $\mathcal{A}g_i$ starts to execute t_{i+1} at st_{i+1}, it succeeds if:

- precedence constraints are respected (t_{i+1}'s predecessors have finished their execution),
- $\mathcal{A}g_i$ has enough resources to execute the task ($\Delta_r^{i+1} \leq r_{t_i}$),
- temporal constraints are respected: $st_{i+1} + \delta_c^{i+1} \leq LET_{i+1}$.

The probability that the agent successfully executes t_{i+1} is then defined as:

$$P_{suc}(st_{i+1}|et(I')_{t_i}, r_{t_i}) = P^r_{enough}(Pred(t_{i+1})) \cdot \prod_{t_k \in P_{red}(t_{i+1})} \sum_{t|t \leq st_{i+1}} P^{t_k}_{\mathcal{ET}}(t|et(I)_{t_i})$$
$$\cdot \sum_{\Delta^{i+1}_r | r_{t_i} \geq \Delta^{i+1}_r} \sum_{\delta^{i+1}_c | st_{i+1} + \delta^{i+1}_c \leq LET_{i+1}} P_r(\Delta^{i+1}_r).P_c(\delta^{i+1}_c)$$

where $P_r(\Delta^{i+1}_r)$ is the probability that the execution of t_{i+1} consumes Δ^{i+1}_r resources and $P_c(\delta^{i+1}_c)$ stands for the probability that the execution takes δ^{i+1}_c time units.

Partial failure transitions The probability the agent partially fails executing the task is equal to the probability that the predecessors have not finished their execution and the agent $\mathcal{A}g_i$ has enough resources to realize that it partially fails. Indeed, if the agent lacks resources, the execution of the task permanently fails. Note that if $st_{i+1} = UB_{i+1}$ and the predecessors have not finished their execution, the agent permanently fails because it could not retry to execute the task later (there is no other possible start time for the task).

The probability $P_{not\_end}(st_{i+1})$ that the agent's predecessors have not finished at st_{i+1} is the probability that the predecessors will finish later or will never finish. The predecessors may never finish because they violated temporal constraints or they lacked resources. The probability that such events occur is given by:

$$1 - P^r_{enough}(Pred(t_{i+1})) \cdot \prod_{t_k \in P_{red}(t_{i+1}) \setminus t_i} \sum_{et \in ET(t_k) | et \leq UB_{i+1}} P^{t_k}_{\mathcal{ET}}(et|et(I')_{t_i})$$

The probability that the predecessors finish after st_{i+1} is:

$$P^r_{enough}(Pred(t_{i+1})) \cdot \prod_{t_k \in P_{red}(t_{i+1}) \setminus t_i} \sum_{et \in ET(t_k)} P^{t_k}_{\mathcal{ET}}(et|et(I')_{t_i})$$
$$- P^r_{enough}(Pred(t_{i+1})). \prod_{t_j \in P_{red}(t_{i+1}) \setminus t_i} \sum_{et' \in ET(t_j) | et' \leq st_{i+1}} P^{t_j}_{\mathcal{ET}}(et'|et(I')_{t_i})$$

Thus, $P_{not\_end}(st_{i+1})$ is defined as follows:

$$\begin{aligned} P_{not\_end}(st_{i+1}) = & P^r_{enough}(Pred(t_{i+1})) \cdot \prod_{t_k \in P_{red}(t_{i+1}) \setminus t_i} \sum_{et \in ET(t_k) | et \leq UB_{i+1}} P^{t_k}_{\mathcal{ET}}(et|et(I')_{t_i}) \\ & - P^r_{enough}(Pred(t_{i+1})). \prod_{t_k \in P_{red}(t_{i+1}) \setminus t_i} \sum_{et' \in ET(t_k) | et' \leq st_{i+1}} P^{t_k}_{\mathcal{ET}}(et'|et(I')_{t_i}) \\ & + 1 - P^r_{enough}(Pred(t_{i+1})) \cdot \prod_{t_k \in P_{red}(t_{i+1}) \setminus t_i} \sum_{end \in ET(t_k) | et \leq UB_{i+1}} P^{t_k}_{\mathcal{ET}}(end|et(I')_{t_i}) \end{aligned}$$

Given P_{not_end}, partial failure probabilities P_{PCV} (Precedence Constraints Violated) is deduced:

- If $st_{i+1} < UB_{i+1}$:

$$P_{PCV} = \sum_{\Delta'_r | r_{t_i} \geq \Delta'_r} P_r(\Delta'_r) \cdot P_{not\_end}(st_{i+1})$$

- Otherwise:

$$P_{PCV} = 0$$

Failure transition The execution of t_{i+1} permanently fails if the agent lacks resources or he violates temporal constraints.

• **Lack of Resources (LR):**

Agent $\mathcal{A}g_i$ may lack resources during the execution of t_{i+1} or when he partially fails. The probability $\mathcal{A}g_i$ lacks resources while executing a task t_{i+1} is the probability that the predecessors have finished their execution (so the agent can start to execute the task) and the execution requires more resources than available ($r_{t_i} < \Delta_r^{i+1}$). The probability the agent lacks resources when it partially fails is the probability the predecessors have not finished their execution and the necessary resources to be aware of it are not sufficient ($r_{t_i} < \Delta_r'$). Recall that $\Delta r'$ is the resource consumption of a partial failure. Thus, the probability P_{LR} the agent fails because of insufficient resources, is computed as follows:

– If $st_{i+1} < UB_{i+1}$:

$$P_{LR} = P^r_{enough}(Pred(t_{i+1}))\cdot \prod_{t_k \in P_{red}(t_{i+1})\setminus t_i} \sum_{t \in ET(t_k) | t \leq st_{i+1}} P^{t_k}_{\mathcal{ET}}(t|et(I)_{t_i})\cdot \sum_{\Delta_r^{i+1} | r_{t_i} < \Delta_r^{i+1}} P_r(\Delta_r^{i+1})$$

$$+P_{not\_end}(st_{i+1})\cdot \sum_{\Delta_r' | r_{t_i} < \Delta_r'} P_r(\Delta_r')$$

– Otherwise ($st_{i+1} = UB_{i+1}$):

$$P_{LR} = P^r_{enough}(Pred(t_{i+1}))\cdot \prod_{t_k \in P_{red}(t_{i+1})\setminus t_i} \sum_{t \in ET(t_k) | t \leq st_{i+1}} P^{t_k}_{\mathcal{ET}}(t|et(I)_{t_i})\cdot \sum_{\Delta_r^{i+1} | r_{t_i} < \Delta_r^{i+1}} P_r(\Delta_r^{i+1})$$

Because $st_{i+1} = UB_{i+1}$, there is no partial failure. If the agent fails, he moves to a permanent failure state.

• **Violation of temporal constraints** $TC(t_{i+1})$**:**

Temporal constraints restrict the start times and end times of each task. They are violated if the agent starts the execution of a task t_{i+1} too late (after the Latest Start Time of the task - LST_{i+1}) or it finishes the execution after the task's deadline (Latest End Time - LET_{i+1}).

Possible start times st_{i+1} of t_{i+1} belong to the interval $[LB_{i+1}, UB_{i+1}]$ and respect temporal constraints. Too late start time failure arises when the agent starts to execute a task t_{i+1} at UB_{i+1} and he partially fails. Then, there is no other possible start time that respects temporal constraints and the agent cannot respect LST_{i+1} deadline. The probability of such a failure is computed as follows:

– If $st_{i+1} < UB_{i+1}$: $P_{TL} = 0$
– Otherwise: $P_{TL} = P_{not\_end}(st_{i+1})$

The agent can also violate temporal constraints if the execution of the task is so long that it finishes after its Latest End Time. In this cas, task duration is such as: $st_{i+1} + \delta_c^{i+1} > LET_{i+1}$. The probability P_{DM} that the deadline is met while executing t_{i+1} is:

$$P_{DM} = P^r_{enough}(Pred(t_{i+1}))\cdot \prod_{t_j \in P_{red}(t_{i+1})-t_i} \sum_{t \in ET(t_j) | t \leq st_{i+1}} P^{t_j}_{\mathcal{ET}}(t|et(I)_{t_i})\cdot \sum_{\Delta_r^{i+1} | r_{t_i} \geq \Delta_r^{i+1}} P_r(\Delta_r^{i+1})$$

$$\cdot \sum_{\delta_c^{i+1} | st_{i+1} + \delta_c^{i+1} > LET_{i+1}} P_c(\delta_c^{i+1})_{i+1}$$

Finally, the probability P_{fail} the agent moves to a failure state is:

$$P_{fail} = P_{LR} + P_{TL} + P_{DM}$$

Decomposition of the system's transition function allows us to modelize the initial multiagent decision problem as a set of individual MDPs that represent each agent's decision problem. Each MDP of the OC-DEC-MDP model formalizes temporal, precedence and resource constraints on task execution. The model also allows for modeling several possible durations for each task. It thus provides a richer model of time and action than the one proposed by standard DEC-MDP framework. As we consider individual actions and states, large problems, that are out of range of other approaches, can be modelized by OC-DEC-MDPs. We detail the scalability of our approach in section 5.1.

3.3 Complexity Analysis

Theorem 1 *Optimally solving an OC-DEC-MDP has a complexity exponential in the number of states.*

Proof: Each agent's state is locally fully observable. Thus, a joint policy is a mapping from world states $S = \langle S_1, \cdots, S_n \rangle$ to joint actions $A = \langle A_1, \cdots, A_n \rangle$. The number of joint policies is exponential in the number of states $|S|$. Evaluating a joint policy can be done in polynomial time through the use of dynamic programming, the OC-DEC-MDP is therefore exponential in the number of states $|S|$. □

Figure 3, classifies some existing models based on DEC-POMDPs. Unlike Transition Independent Decentralized Markov Decision Processes (TI-DEC-MDPs) and Observation Independent Decentralized Markov Decision Processes (OI-DEC-MDPs), we do not assume some independence properties about the transition or observation functions. The Event Driven Decentralized Markov Decision Processes (ED-DEC-MDPs) (Becker et al., 2004a) model is OC-DEC-MDP's nearest framework. Indeed, ED-DEC-MDPs and OC-DEC-MDPs both deal with constraints on task execution and both consider several possible durations for each task. These properties are not taken into account by other models. ED-DEC-MDPs formalize a problem as a factored DEC-MDP where each agent's state keeps track of information about dependencies. This leads to larger set of states than the ones defined in OC-DEC-MDP. Such a definition of states allows for representing a wider range of dependencies but it limits the size of problems that can be solved. On the other hand, OC-DEC-MDP's definition exploits constraints and dependencies to restrict the action and state spaces allowing for representing large size of problems (see section 5.1).

4 OC-DEC-MDP resolution

Given the decomposition of the problem, we aim at solving each local MDP and deducing each agent's policy. Nevertheless, we consider cooperative multiagent systems where interactions between the agents arise from precedence and temporal constraints. Thus, maximizing a common reward requires the agents to coordinate and MDPs cannot be independently solved.

Temporal constraints restrict the possible intervals of execution and precedence constraints partially order the tasks. If the execution of a task t_{i+1} starts before its predecessors

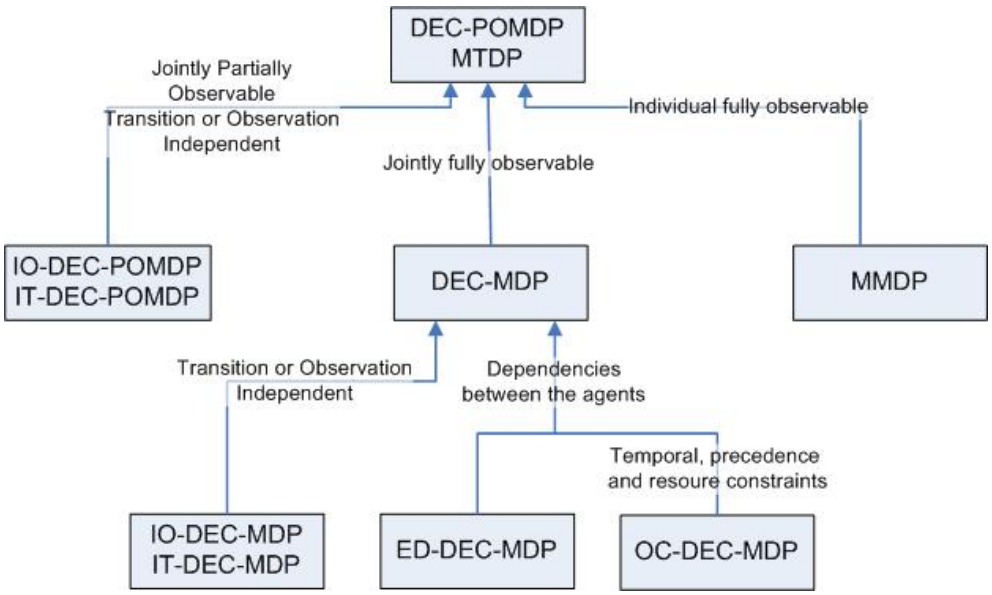

Fig. 3 Relationships between DEC-POMDP based models

finish, it partially fails. Partial failures consume restricted resources and can lead to insufficient resources. If an agent lacks resources it will be unable to execute its remaining tasks. Consequently, the agents tend to avoid partial failures. One way to restrict partial failures consists in delaying the execution of the tasks. As a result, the likelihood that the predecessors have finished when an agent starts to execute a task increases and less resources are consumed by partial failures. Nonetheless, the more the execution of a task is delayed, the more the successors are delayed and the higher the probability of violating temporal constraints. In fact, the probability the deadline is met increases. If temporal constraints are violated during task execution, the agent fails permanently executing the task and does not obtain the reward associated with the task. Note that this assumption can be easily relaxed without modifications of our approach.

The problem is to find a local policy for each agent that maximizes the sum of the rewards of all the agents. The agents must trade off the probability of partially failing and consuming resources against the consequences of delaying the execution of a task. To maximize the sum of the expected rewards, each agent must consider the consequences of a delay on itself and on its successors. For purpose of coordinating the agents, we introduce the notion of Opportunity Cost (OC) and we propose valuation measures that lead to cooperative behaviors. We then introduce algorithms that use these measures to evaluate each agent's policy and solve the decision problem.

4.1 Opportunity cost and policy evaluation

Opportunity cost is borrowed from economics and refers to hidden indirect costs associated with a decision (Wieser, 1889). In our approach, we use opportunity cost to measure the effect of an agent's decision on the other agents. More specifically, the opportunity cost is

the loss of expected value of the other agents resulting from delaying the execution of their tasks. Taking this cost into account leads to better coordination among the agents: it allows each agent to consider how its decisions influence the other agents.

When an agent $\mathcal{A}g_i$ decides when to start the execution of a task t_{i+1}, its decision influences all the tasks that can be reached from t_{i+1} in the mission graph (direct or indirect successors of t_{i+1}). In order to obtain coordinated behaviors, each agent must therefore consider the influence of its actions on the other agents. This is measured by expected opportunity cost (EOC) and policies are valued using two equations: a standard Bellman equation and a modified Bellman equation. The modified Bellman equation allows the agent to select the best action to execute in a state s_i, considering its expected utility and the expected opportunity cost induced on the other agents. The best action to execute from a given state results from a trade-off between the agent's expected utility and the EOC provoked on the other agents.

The first equation is a standard Bellman equation that computes for each state s_i of each agent $\mathcal{A}g_i$, the agent's utility. It is based on Bellman optimality principle:

$$V(s_i) = \overbrace{R(s_i)}^{\text{Immediat Gain}} + \overbrace{max_{E(t_{i+1},st),st\geq t}(V')}^{\text{Expected Utility}} \tag{7}$$

where $s_i = [t_i, [st(I')_{t_i}, et(I')_{t_i}], r_{t_i}]$ (and $et(I')_{t_i} = t$) or $s_i = [t_i, [t-1, t], et(I')_{t_i}, r_{t_i}]$. If $s_i = [t_i, [st(I')_{t_i}, et(I')_{t_i}], r_{t_i}]$ (success state), $t = et(I')_{t_i}$. Moreover, $R(s_i) = \mathcal{R}(t_i)$. Otherwise, $R(s_i) = 0$.

Different kinds of transitions must be considered to compute the expected value, thus V' is such that: $V' = V_{suc} + V_{PCV} + V_{fail}$. Each part of V' stands for a kind of transitions:

• **Success transition:**

$$V_{suc} = P_{suc}(st|et(I')_{t_i}, r) \cdot V([t_{i+1}, I, r_{t_i} - \Delta r])$$

• $Pred(t_{i+1})$ **violated:**

$$V_{PCV} = P_{PCV} \cdot V([t_i, [st, st+1], et(I')_{t_i}, r_{t_i} - \Delta r'])$$

• **Failure transition:**

$$V_{fail} = P_{fail} \cdot V([failure_{t_{i+1}}, *, *])$$

where $V([failure_{t_{i+1}}, *, *])$ is the value of the failure state associated with t_{i+1}:

$$V([failure_{t_{i+1}}, *, *]) = -\mathcal{R}(t_{i+1}) - \sum_{t_{suiv} \in \mathcal{T}_{suiv}(t_{i+1})} \mathcal{R}(t_{suiv})$$

where $-\mathcal{R}(t_{i+1})$ is the immediate penalty due to the failure of t_{i+1} and $\sum_{t_{suiv} \in \mathcal{T}_{suiv}(t_{i+1})} \mathcal{R}(t_{suiv})$ is the loss in value due to the failure of all the remaining tasks executed by the $\mathcal{A}g_i$ (the tasks that can be reached from t_{i+1} in the mission graph). This is deduced from the mission graph.

The second equation computes the policies of the agents. We use an augmented Bellman equation in which an expected opportunity cost is introduced:

$$\pi_i(s_i) = argmax_{E(t_{i+1},st),st\geq et(I')_{t_i}} \Big(\overbrace{V'}^{\text{Expected value}} - \overbrace{EOC(t_{i+1}, st)}^{\text{Expected Opportunity Cost}} \Big) \tag{8}$$

where $argmax$ denotes an operator which returns the action of $E(t_{i+1}, st), st \geq et(I')_{t_i}$ which maximizes V'. As mentioned previously, $E(t_{i+1}, st)$ is the action which consists in

executing t_{i+1} at st. $s_i = [t_i, [st(I')_{t_i}, et(I')_{t_i}], r_{t_i}]$ (and $et(I')_{t_i} = t$) or $s_i = [t_i, [t-1, t], et(I')_{t_i}, r_{t_i}]$. $EOC(t_{i+1}, st)$ is the EOC the execution of t_{i+1} will induce if the agent starts at st and $V' = V_{suc} + V_{PCV} + V_{fail}$.

Thus, the most valuable foregone action is selected by considering:

- The expected value, computed using a standard Bellman equation (Equation 7). It takes into account the expected value of executing the agent's remaining task.
- The expected opportunity cost provoked on the other agents.

The following of the section will deal with the definition and computation of the expected opportunity cost.

Let two tasks t_{i+1} and t_j be respectively executed by two agents $\mathcal{A}g_i$ and $\mathcal{A}g_j$, and such that t_j is a successor of t_{i+1}. Moreover, let suppose that $\mathcal{A}g_j$ has r_{t_j} available resources when it starts to execute t_j. Two possible executions of t_{i+1} are presented in Figure 4. The first one starts at $st1$ and ends at $et1 = LB_j$. Thus, the execution of t_j can start in the interval $[LB_j, UB_j]$, $\mathcal{A}g_j$ will choose the best start time st_j in this interval. The second execution of t_{i+1} starts at $st2$ and ends at $et2$. Then, t_j could not start before $et2$. If we consider that $et2 - LB_j = \Delta t$, the execution of t_j will start in the interval $[LB_j + \Delta t, UB_j]$. The interval of the possible start times is reduced by Δt and $\mathcal{A}g_j$ will choose the best start time st'_j in the interval $[LB_j + \Delta t, UB_j]$. If $st_j = st'_j$, the fact that t_{i+1} ends later does not affect the best possible start time of t_j. Otherwise ($st_j \neq st'_j$), the expected utility $V_{t_j}^{0,r_{t_j}}$ the agent can obtain when it starts t_j at st_j (best start time in $[LB_j, UB_j]$) with r_{t_j} resources, is greater than $V_{t_j}^{\Delta t,r_{t_j}}$ the expected utility when it starts the execution of t_j at st'_j (best start time in $[LB_j + \Delta t, UB_j]$) with r_{t_j} resources. Then, the fact that t_{i+1} ends at $et2$ leads to a loss in $\mathcal{A}g_j$'s expected value.

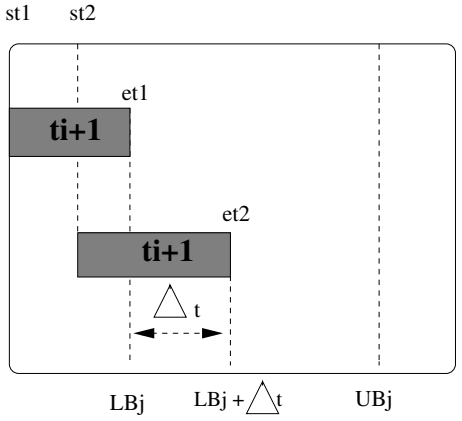

Fig. 4 Delay example

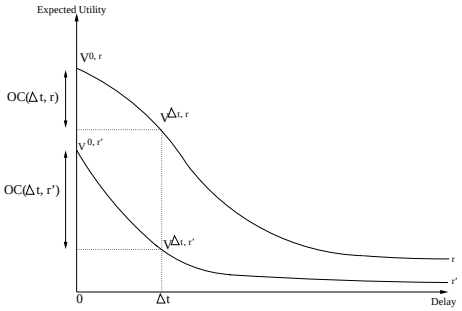

Fig. 5 Influence of resources and delay on Expected Value

The OC induced by $\mathcal{A}g_i$ on $\mathcal{A}g_j$ is the loss in value when $\mathcal{A}g_j$ has r_{t_j} resources and its first possible start time is delayed by Δt. The OC is measured by the difference between $V_{t_j}^{0,r_{t_j}}$, the expected value if t_j can start in $[LB_j, UB_j]$ with r_{t_j} resources , and $V_{t_j}^{\Delta t,r_{t_j}}$ the expected value if t_j can start in the interval $[LB_j + \Delta t, UB_j]$ with r_{t_j} resources. The

opportunity cost is then given by:

$$OC_{t_j}(\Delta t, r_{t_j}) = V_{t_j}^{0,r_{t_j}} - V_{t_j}^{\Delta t,r_{t_j}} \tag{9}$$

As the rate of resource r_{t_j} influences the expected value, an opportunity cost is computed for each resource rate and each delay Δt (see Figure 5). For instance, if the resource is very tight, the probability the agent will fail because of lack of resources, is very high. Then, delaying a task has a low cost because whatever its start time, the likelihood its execution fails is very high.

Claim: OC is always positive or equal to zero.

Proof: Let assume that $V_{t_j}^{0,r_{t_j}} < V_{t_j}^{\Delta t,r_{t_j}}$. st_j is the best possible start time in the interval $[LB_j, UB_j]$ and allows to obtain $V_{t_j}^{0,r_{t_j}}$. st'_j is the best possible start time in the interval $[LB_j + \Delta t, UB_j]$ and allows to obtain $V_{t_j}^{\Delta t,r}$. If st_j belongs to $[LB_j + \Delta t, UB_j]$ then, it is the best possible start time in $[LB_j + \Delta t, UB_j]$. Therefore $st_j = st'_j$ and $V_{t_j}^{0,r_{t_j}} = V_{t_j}^{\Delta t,r_{t_j}}$. We cannot have $V_{t_j}^{0,r_{t_j}} < V_{t_j}^{\Delta t,r_{t_j}}$.

If st_j does not belong to $[LB_j + \Delta t, UB_j]$ then, there is another best possible start time st'_j in $[LB_j + \Delta t, UB_j]$ and $V_{t_j}^{0,r_{t_j}} > V_{t_j}^{\Delta t,r_{t_j}}$, otherwise st'_j would be the best start time in $[LB_j, UB_j]$. □

Expected Opportunity Cost

Let suppose that a task t_{i+1} starts at st. Since actions are not deterministic, we do not know exactly when the task t_{i+1} will end and the opportunity cost it will induce. Different kinds of transitions must be taken into account so as to consider the possible end times of the task. That's why we define expected opportunity cost and Equation 8 considers the expected opportunity cost provoked by a task t_{i+1} on the other agents when it starts at st. Given a start time st, the exepcted opportunity cost consider, for each possible end time $et_{t_{i+1}}$ of t_{i+1}, the probability t_{i+1} ends at $et_{t_{i+1}}$ and the OC provoked on the other agent in such a case. The expected opportunity cost induced on the other agents when t_{i+1} starts at st is defined as follows:

$$\begin{aligned}
EOC(t_{i+1}, st) =& P_{suc} \times \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} EOC_{\mathcal{A}g_j, t_{i+1}}(et_{i+1}) \\
&+ P_{fail} \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} EOC_{\mathcal{A}g_j, t_{i+1}}(fail) + P_{PCV} \times EOC(t_{i+1}, t = next_start) \\
=& P^r_{enough}(Pred(t_{i+1})) \cdot \prod_{t_k \in P_{red}(t_{i+1})} \sum_{t | t \leq st_{i+1}} P^{t_k}_{\mathcal{ET}}(t | et(I)_{t_i}) \\
&\cdot \sum_{\Delta_r^{i+1} | r_{t_i} \geq \Delta_r^{i+1}} \sum_{\delta_c^{i+1} | st_{i+1} + \delta_c^{i+1} \leq LET_{i+1}} P_r(\Delta_r^{i+1}).P_c(\delta_c^{i+1}).EOC_{\mathcal{A}g_j, t_{i+1}}(et_{i+1}) \\
&+ (P_{LR} + P_{TL} + P_{DM}) \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} EOC_{\mathcal{A}g_j, t_{i+1}}(fail) \\
&+ P_{PCV} \times EOC(t_{i+1}, t = next_start)
\end{aligned} \tag{10}$$

where et_{i+1} is a possible end time of t_{i+1}, $EOC_{\mathcal{A}g_j, t_i}(et_{i+1})$ is the EOC induced on the agent $\mathcal{A}g_j$ when it could not start before et_{i+1} (the end time of t_{i+1}), and $EOC(t_{i+1}, t = next\_start)$ is the opportunity cost when the execution of t_{i+1} partially fails and the agents re-try to execute the task at t (the next start time given for t_{i+1}). The execution of t_{i+1} can lead to different transitions, therefore all these transitions must be considered while

computing $EOC(t_{i+1}, st)$. For instance, if the execution of the task succeeds, the EOC will be different from the EOC induced on the other agents if the execution fails.

When an agent $\mathcal{A}g_i$ finishes to execute a task t_{i+1} at $et_{t_{i+1}}$, we consider the expected opportunity cost provoked on each other agent $\mathcal{A}g_j$. We thus compute the delay provoked on the nearest task t_j that will be executed by $\mathcal{A}g_j$. This is the nearest task in the mission graph that will be influence by the delay of t_{i+1}. Distance between two tasks t_i and t_j is given by the number of nodes that belong to the shortest path between t_i and t_j in the mission graph. Let consider the task "Put out from flames" described on Figure 1. The ambulance agent's nearest task from the task "Put out from flames" is "Give first aid to injured people".

This expected opportunity cost provoked by $\mathcal{A}g_i$ on $\mathcal{A}g_j$ is denoted $EOC_{\mathcal{A}g_j,t_{i+1}}(et_{t_{i+1}})$. If there is no intermediate task between t_{i+1} and t_k (t_k is a direct successor of t_{i+1}), the delay of t_{i+1} is the delay provoked on t_k. Thus,

$$EOC_{\mathcal{A}g_j,t_{i+1}}(et_{t_{i+1}}) = \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) \times OC_j(et_{t_{i+1}} - LB_j, r_{t_j}) \tag{11}$$

Otherwise, the tasks between t_{i+1} and t_k may increase or decrease the delay provoked on t_k. The expected provoked on t_k when t_{i+1} ends at $et_{t_{i+1}}$ is then recursively computed. Let t_l be the successor task of t_{i+1} on the path from t_{i+1} to t_k. We obtain :

$$EOC_{\mathcal{A}g_j,t_{i+1}}(et_{t_{i+1}}) = \sum_{r_{t_l}} P_{ra}^{t_l}(r_{t_l}) \sum_{\delta_c^l} P_c(\delta_c^l) EOC_{\mathcal{A}g_j,t_l}(st^* + \delta_c^l) \tag{12}$$

where st^* is t_l's start time when the $\mathcal{A}g_l$ who executes t_l has r_{t_l} resources before executing t_l and it cannot start executing t_l before $et_{t_{i+1}}$.

Given a start time st of t_{i+1}, Equation 10 computes the probabilities on t_{i+1} end times. For all t_{i+1}'s end times and each agent $\mathcal{A}g_i$, Equations 11 and 12 computes the delay provoked on $\mathcal{A}g_j$'s nearest task t_j, assuming t_{i+1} ends at $et_{t_{i+1}}$. Given this delay, Equation 9 computes the OC on t_j. These equations allow us to deduce the expected opportunity cost introduced in the augmented Bellman equation (Equation 8). Note that the OC is always positive, then the EOC is always positive. While deciding the best action to execute in a state s_i, the expected value is reduced by $-EOC(t_{i+1}, st)$ which always stands for a cost.

4.2 Revision algorithm

Optimally solving a general DEC-MDP is a double exponential problem (Bernstein et al., 2002). It is therefore intractable to find an optimal solution for large size of problems (Becker et al., 2004a). In order to solve large realistic problems that may be composed of hundreds tasks and more than ten agents, we aim at finding an approximate solution. In this section, we present a revision algorithm that uses the coordination mechanism described in the previous section to evaluate each agent's policy and solve the problem.

The revision algorithm consists in improving the initial set of policies which has been previously used to compute transitions probabilities. Two versions of the algorithm have been developed. A centralized one (Algorithm 1) allows a central entity to revise all the agents policies and improves the policy of each task using Equation 8. Decentralized version of the algorithm has also been proposed: each agent improves its own policy, thus allowing for considering several tasks at the same time.

4.2.1 Centralized revision algorithm

Because of dependencies between the agents, the centralized revision algorithm (Algorithm 1) evaluates the local MDPs at the same time. The algorithm passes through the mission graph from the leaves to the roots. The tasks of the mission are organized into levels. The first level contains the leaves of the graph. The second level contains the predecessors of the leaves whose successors have already been evaluated. Level L_n contains the predecessors of the tasks belonging to level L_{n-1} whose successors have already been evaluated. For each task t_{i+1} of a level L_n, the execution policy of t_{i+1} is revised. In fact, we consider the states from which t_{i+1} can be executed. Let t_{i+1} be executed by $\mathcal{A}g_i$ and let t_i be the task that $\mathcal{A}g_i$ executes just before t_{i+1}. While revising the execution policy of t_{i+1}, we consider the partial failures and the success state associated with t_i. Then, Equation 8 is applied to select the best action to execute from these states.

Once the policy of a task t_{i+1} has been revised, opportunity cost values associated with t_{i+1} are computed. These values will be used while revising the predecessors of t_{i+1}. Notice that leaves have no successor so delaying their execution do not provoke any loss in value on the other agents (opportunity cost on the other agents is zero). Nonetheless, delaying a leaf t_i may provoke a loss in value on the agent which executes t_i (because of temporal constraints). While considering a level L_n, we only consider the tasks whose successors have already been evaluated, we therefore guarantee that we know the opportunity cost the execution of t_{i+1} will provoke on the other agents.

In order to measure, the EOC provoked by the execution of t_{i+1} (Equation 10), we need to know the delay provoked on the nearest tasks t_k of each agent $\mathcal{A}g_k$ such as $k \neq i$. This delay depends on the policies of the tasks between t_{i+1} and t_k. As the algorithm passes through the graph from the leaves to the node, the policies of the tasks between t_{i+1} and t_k have been revised when t_{i+1} is considered. If the initial policy is used to estimate the delay provoked on t_k, it may lead to inaccurate results because of policy changes. We have therefore developed an update method that guarantees the accuracy of expected opportunity cost values. Each time the algorithm finishes revising the policy of a task t_j executed by $\mathcal{A}g_j$, opportunity cost values are computed using Equation 9. Then, for each nearest tasks t_k of each agent $\mathcal{A}g_k$, expected opportunity cost values $EOC_{t_j}(\Delta t_{t_j}, t_k)$ are updated using Equations 11 and 12 and t_j's new policy. These values will then be used to revise the policies of t_j's predecessors. Let t_{i+1} be a predecessor of t_j. While revising the policy of task t_{i+1}, we know that all the successors of t_{i+1} have already been evaluated, EOC values of the successors $EOC_{t_{i+1}}(\Delta t_{t_{i+1}}, t_k)$ that are used to compute $EOC_{t_{i+1}}(\Delta t_{t_{i+1}}, t_k)$ are therefore guaranteed to be computed using the revised policy.

Theorem 2 *The time complexity of the centralized revision algorithm is polynomial in* $|\mathcal{SU}| \times |\mathcal{A}|$ *where* $|\mathcal{A}|$ *is the maximum number of actions that can be executed from a state* s_i. $|\mathcal{A}| = |ST|$ *where* $|ST|$ *is the maximum number of possible start times for a task.* $\mathcal{SU}$ *is the union of the agents's states:* $|\mathcal{SU}| = \sum_{\mathcal{A}g_i \in \mathcal{A}g} |\mathcal{S}_i| < |\mathcal{S}|$.

Proof: Let $|\mathcal{S}(t_i)|$ be the number of states associated with a task t_i. The centralized revision algorithm passes through the state space of each agent. For each state s_i, a value $V(s_i)$ and a policy $\pi_i(s_i)$ are computed. Their complexity is $O(|ST|)$. Indeed, each action has to be considered and in the worst case, there are $|ST|$ actions for each state. Moreover, for each task t_i there are $|\mathcal{S}(t_i)|$ states to consider.

Let $|ST(t_i)|$ be the number of possible start times for a task t_i and let $\#r_i$ the maximum of possible resource rates per task. Lines 10 to 15 of the algorithm compute the OC values associated with a task t_i. The complexity of computing $V^{\Delta t, r_{t_i}}$ and $OC(\Delta t, r_{t_i})$ is

Algorithm 1 Centralized Revision Algorithm

Require: the OC-DEC-MDP, an initial set of policies $\pi =< \pi_1, \cdots \pi_n >$
Ensure: a new set of policies π
1: **for all** level L_n from the leaves to the roots of the mission graph **do**
2: **for all** task t_{i+1} in level L_n **do**
3: Compute V for the failure state: $[failure(t_{i+1}), *, *]$
4: **for all** partial failure states $[t_i, [st, st+1], et(I')_{t_i}, r_{t_i}]$ associated with t_i **do**
5: Compute V and π for $[t_i, [st, st+1], et(I')_{t_i}, r_{t_i}]$
6: **end for**
7: **for all** success state $[t_i, [st, st+\delta^i_c], r_{t_i}]$ associated with t_i **do**
8: Compute V and π for $[t_i, [st, st+\delta^i_c], r_{t_i}]$
9: **end for**
10: **for all** start time st of t_{i+1} from $UB_{t_{i+1}}$ to $LB_{t_{i+1}}$ **do**
11: **for all** resource rate r_{t_i} available after a successful execution of t_i **do**
12: Compute $V_{t_i}^{\Delta t, r_{t_i}}$ where $\Delta t = st - LB_{t_i}$
13: Compute $OC(\Delta t, r_{t_i}) = V_{t_i}^{0, r_{t_i}} - V_{t_i}^{\Delta t, r_{t_i}}$ and deduce EOC values
14: **end for**
15: **end for**
16: **for all** agent $\mathcal{A}g_j$ that does not execute t_{i+1} **do**
17: Update the EOC values of $\mathcal{A}g_j$'s nearest task from t_{i+1}
18: **end for**
19: **end for**
20: **end for**

$O(1)$ since they consist in summation over possible transitions. In the worst case, there are $|ST_i| \times \#r_i$ values to compute for each task. Thus, the complexity of computing OC values is $O(|ST(t_i)| \times \#r_i)$ and

$$|ST(t_i)| \times \#r_i < |\mathcal{S}(t_i)|$$

Moreover, EOC values must be updated. In the worst case there are $|ST| \times |\mathcal{A}g|$ values to update and $|ST| \times |\mathcal{A}g| < |\mathcal{S}(t_i)| \times |ST|$

Thus, the complexity of revising a task t_i is $O(|\mathcal{S}(t_i)| \times |ST|)$. Lines 3 to 18 are executed for each task and

$$\sum_{t_i \in \mathcal{T}} |\mathcal{S}(t_i)| = |\mathcal{SU}|$$

The time complexity of the algorithm is therefore $O(|ST| \times |\mathcal{SU}|)$. □

Note that if there are several possible next tasks from a state s_i (the agent must choose between several tasks), the algorithm remains polynomial. Nonetheless, the number of actions that can be executed from a state s_i is $|ST| \times |Next|$ where $|Next|$ is the number of possible next tasks to consider. Note that non-selected actions are delayed. The consequences of delaying this task are taken into account by the expected utility V' of the agent and the expected opportunity cost provoked on the other agent (EOC computation is similar to the computation described in this paper).

4.2.2 Decentralized revision algorithm

Decentralized revision algorithm (Algorithm 2) allows the agents to simultaneously evaluate their own local MDPs. Thus, each agent derives a new local policy from its initial policy by applying the decision mechanism described in Equation 8. Unlike centralized revision algorithm which considers only one task at the same time, decentralized algorithm allows for several agents to simultaneously revise the execution policies of their tasks.

While revising his policy, each agent only considers the tasks he has to execute. Given a mission graph, a graph of tasks can be defined for each agent $\mathcal{A}g_i$. This graph of tasks orders the task the agent $\mathcal{A}g_i$ has to execute. Then, the agent passes through his graph of tasks from the leaves to the roots and revises the policy of each task.

Because of decentralization of the decision process, the agents have to communicate expected opportunity cost values. When an agent evaluates the policy of a task t_{i+1}, he needs to know the expected opportunity cost he will provoke on the other agents. He must therefore have received EOC values from its successors. If these values have not been received, the agent waits until delivery. We assume that there is no loss of messages. As soon as EOC values are known, the agent can compute its expected value and the policy of t_{i+1}. Notice that even if decentralized execution of the algorithm requires off-line communication, the agents never communicate during the execution of the mission.

As soon as an agent finishes revising the policy of a task t_{i+1}, he computes the OC values associated with t_{i+1} using Equation 9 and EOC values of t_{i+1} are deduced. Given the EOC values he has received and the OC values of t_{i+1}, he then updates the expected opportunity cost provoked on the other agents by t_{i+1}. For each agent $\mathcal{A}g_j$ and each delay $\Delta t_{t_{i+1}}$, he therefore computes $EOC_{t_{i+1}}(\Delta t_{t_{i+1}}, t_j)$ using Equations 11 and 12 and t_{i+1}'s new policy (t_j is the nearest task that will be executed by $\mathcal{A}g_j$). Finally, he sends these updated EOC values to the predecessors of t_{i+1} which will use them while computing the EOC defined by Equation **??**. Communicating updated expected opportunity cost values instead of opportunity cost values guarantees the accuracy of opportunity cost regarding revised policies.

Algorithm 2 Decentralized Revision Algorithm

Require: $\mathcal{A}g_i$'s local MDP and the initial policy π_i o $\mathcal{A}g_i$
Ensure: A new policy π_i of the agent $\mathcal{A}g_i$
for all level L_n from the leaves to the roots of the agent $\mathcal{A}g_i$'s graph of tasks **do**
  for all task t_{i+1} in level L_n **do**
    while the agent does not have received the EOC values he needs **do**
      wait
    end while
    Compute V for the failure state: $[failure(t_{i+1}), *, *]$
    for all partial failure states $[t_i, [st, st+1], et(I')_{t_i}, r_{t_i}]$ associated with t_i **do**
      Compute V and π for $[t_i, [st, st+1], et(I')_{t_i}, r_{t_i}]$
    end for
    for all success state $[t_i, [st, st+\delta_c^i], r_{t_i}]$ associated with t_i **do**
      Compute V and π for $[t_i, [st, st+\delta_c^i], r_{t_i}]$
    end for
    for all start time st of t_{i+1} from $UB_{t_{i+1}}$ to $LB_{t_{i+1}}$ **do**
      for all resource rate r_{t_i} available after a successful execution of t_i **do**
        Compute $V_{t_i}^{\Delta t, r_{t_i}}$ where $\Delta t = st - LB_{t_i}$
        Compute $OC(\Delta t, r_{t_i}) = V_{t_i}^{0, r_{t_i}} - V_{t_i}^{\Delta t, r_{t_i}}$ and deduce EOC values
        Send EOC to t_{i+1}'s predecessors
      end for
    end for
    for all task t_j executed by another agent and for which EOC values has been received **do**
      Update the EOC values and send them to t_{i+1}'s predecessors
    end for
  end for
end for

Theorem 3 *The complexity of the decentralized revision algorithm is polynomial time in* $|\mathcal{A}| \times |\mathcal{SU}| + \#_{OC} \times K \times TM$ *where* K *is the size of a message and* TM *is the time needed to communicate one unit of information.* $|\mathcal{A}|$ *is the maximum number of actions that can be executed from a state* s_i. $|\mathcal{A}| = |ST|$ *where* $|ST|$ *is the maximum number of possible start times for a task.*

Proof: In the worst case, none of the states can be evaluated at the same time and the algorithm has the same time complexity as the centralized algorithm which evaluates all the agents' states one by one. Only one state is therefore evaluated at a time. The time needed to pass trough the state space of each local MDP and to value each state is $|ST| \times |\mathcal{SU}|$.

The complexity of sending an OC value relies on the size of the message (a message consists in communicating a "double" value) and the time needed to send one unit of the message. In the worst case there are $\#_{OC}$ values of opportunity cost to communicate where:

$$\#_{OC} = \sum_{t_i \in \mathcal{T}} |\mathcal{A}g| \times |ST|$$

Indeed, each time an agent revises the policy of a task t_{i+1}, it must broadcast the updated expected opportunity cost values. While updating the expected opportunity cost values, the nearest task of each agent is considered and there are $|ST|$ values to update for each agent.

Thus, the time complexity of the algorithm is $O(|ST| \times |\mathcal{SU}| + \#_{OC} \times K \times TM)$. □

Complexity analysis of both versions of the revision algorithm suggests that large problems could be solved. Indeed, we propose a polynomial algorithm whereas other existing approaches are in best case exponential (Nair et al., 2003; Becker et al., 2004b). As detailed in section 5.1, our approach allows for scaling up to problems with hundreds of tasks whereas existing approaches are limited to small problems involving at best about ten tasks.

*Performance*The revision algorithm consists in improving an initial set of policies and results in an approximate solution. However, under some identified assumptions, the algorithm returns an optimal solution. Let EST-policy be the policy which consists in executing each task as soon as possible (Earliest possible Start Time). Let LST-policy be the policy which consists in executing each task as late as possible (Latest possible Start Time). For purpose of good understanding, we do not detail the proofs of the following claims. Although, the authors are willing to provide mathematical details of these proofs.

Claim 1 *Under unlimited resources, EST-policy is an optimal policy.*

Proof: Under unlimited resources, decisions are not influenced by available resources. Thus, the agents do not tend to avoid partial failures and do not have to delay the execution of their tasks. The agents do not have to care about resource failure consumption. Nevertheless, they have to to respect temporal constraints. In order to limit failures due to violation of temporal constraints, the agents must start their task as soon as possible. Under unlimited resources, the agents therefore select the earliest possible start time of each task and EST-policy is an optimal policy.□

Claim 2 *If there is no constraint on tasks' end times (*$LET \simeq +\infty$*), LST-policy is an optimal policy.*

Proof: If there is no constraint on tasks' end times ($LET \simeq +\infty$), an agent cannot fail because he finishes the execution of his task after the deadline. Delaying the execution of a task does not increase the probability of deadline met. Moreover, this decreases the probability of partially failing. So as to maximize their utility, the agents have to start the execution of their task as late as possible. LST-policy is therefore an optimal policy.□

Claim 3 *If there is no constraint on tasks' end times ($LET \simeq +\infty$) and resources are unlimited, all policies are optimal policies.*

Proof: If there is no constraint on tasks' end times ($LET \simeq +\infty$) and resources are unlimited, resources and deadlines do not influence the agents' expected utility. Whenever the agents start the execution of their task, their expected utility remains unchanged. Thus, the agents can start the execution of their task whenever they want.□

Claim 4 *Under unlimited resources, the revision algorithm computes an optimal policy.*

Proof: See Appendix A

Claim 5 *If there is no constraint on tasks' end times ($LET \simeq +\infty$), the revision algorithm computes an optimal policy.*

Proof: See Appendix B

Claim 6 *If there is no constraint on tasks' end times ($LET \simeq +\infty$) and resources are unlimited, the revision algorithm computes an optimal policy.*

Proof: If there is no constraint on tasks' end times ($LET \simeq +\infty$) and resources are unlimited, all policies are optimal policies. We deduce that computed policies are always optimal. □

Claim 7 *Let $\mathcal{X}$ be a mission involving two agents $\mathcal{A}g_i$ and $\mathcal{A}g_j$. If the initial policy of $\mathcal{A}g_i$ is EST-policy and all precedence constraints come from $\mathcal{A}g_i$ to $\mathcal{A}g_j$ then, the revision algorithm computes an optimal policy.*

Proof: See Appendix C

4.3 Iterative algorithm

Both versions of the revision algorithm presented in the previous section consist in improving an initial policy set. When the algorithm stops, each task has been considered once and a new policy is available for each agent. In order to obtain better solutions, we suggest re-executing the revision algorithm considering that the initial policy set is the set of policies we have just computed. By iterating the revision algorithm, each task is therefore considered several times. Let t_i be a task executed by $\mathcal{A}g_i$ and let t_j be a predecessor of t_i. At first iteration step, an initial policy is assumed for the execution of t_i and t_j. t_i's policy is revised first, assuming t_j's initial policy. Once the policy of t_i has been revised, t_j is considered and its policy is also revised. Thanks to updated expected opportunity cost, t_j's new policy is computed assuming the new policy of t_i. Nonetheless, given the new policy of t_j, a better policy may be found for t_i. The iteration process allows for revising t_i's policy given t_j's new policy.

The outcome policies of iteration N-1 are the initial policies of iteration N. Obviously, the transition function depends upon the initial policy of the current iteration and must be updated at each iteration step by propagating temporal and resource constraints through the mission graph (see section 3.2). Once the new transition function is known, each agent re-executes the revision algorithm to obtain new local policies. This process is repeated until no changes are made. Note that states, actions and reward functions remain unchanged at each iteration.

Algorithm 3 Centralized Iterative Algorithm

Require: the OC-DEC-MDP, an initial set of policies $\pi =< \pi_1, \cdots \pi_n >$
Ensure: a new set of policies π
1: **repeat**
2: nbChanges = 0
3: Compute new transition functions from π
4: $\pi' \leftarrow$ Revise local policies $\pi_i \in \pi$ using the centralized revision algorithm
5: **for all** $\mathcal{A}g_k \in \mathcal{A}g$ **do**
6: nbChanges += number of changes in π'
7: **end for**
8: $\pi \leftarrow \pi'$
9: **until** nbChanges == 0

Two versions of the iterative algorithm have been developed. They are based on the two versions (centralized and decentralized) of the revision algorithm. The centralized iterative algorithm (Algorithm 3) consists in iteratively executing the centralized revision algorithm until no policy changes are made.

Theorem 4 *The complexity of the centralized iterative algorithm is polynomial time in* $\mathcal{IN} \times |\mathcal{SU}| \times |\mathcal{A}|$ *where* $\mathcal{IN}$ *is the number of iterations.* $|\mathcal{A}|$ *is the maximum number of actions that can be executed from a state* s_i. $|\mathcal{A}| = |ST|$ *where* $|ST|$ *is the maximum number of possible start times for a task.*

Proof: At each iteration step, the transition function is computed and the revision algorithm is executed. The complexity of the centralized revision algorithm is polynomial time in $|\mathcal{SU}| \times |ST|$. The transition function is updated before each iteration step (line 4) by propagating temporal constraints through the mission graph whose complexity is less than $O(|\mathcal{SU}|)$. Then, the overall complexity of the centralized iterative algorithm is $O(\mathcal{IN} \times |\mathcal{SU}| \times |ST|)$. □

While decentralizing the revision process (Algorithm 4), each agent iteratively improves its own policy until no policy changes are made by any agents. In order to update its transition function, each agent has to know all the policy changes the other agents have made. Policy changes must therefore be broadcasted at the end of each iteration. Nonetheless, little information exchange is required compared to other approaches such as Subjective MDPs developed by Chadès et al. (2002) or the Joint Equilibrium based Search for Policies (JESP) (Nair et al., 2003). Indeed, these approaches only revise a part of the agents' policies at each iteration step thus leading to more frequent policy exchange.

Theorem 5 *The complexity of the centralized iterative algorithm is polynomial time in* $\mathcal{IN} \times (|\mathcal{A}| \times |\mathcal{SU}| + (\#_{OC} + |\mathcal{SU}|) \times K \times TM)$ *where* $\mathcal{IN}$ *is the number of iterations.* $|\mathcal{A}|$ *is the maximum number of actions that can be executed from a state* s_i. $|\mathcal{A}| = |ST|$ *where* $|ST|$ *is the maximum number of possible start times for a task.*

Proof: At each iteration step, the transition function is computed and the revision algorithm is executed. The complexity of the centralized revision algorithm is $O(|ST| \times |\mathcal{SU}| + \#_{OC} \times K \times TM)$. At the end of each iteration step, each agent must send his policy changes to the other agents. There are in the worst case $|\mathcal{SU}|$ values to communicate. The transition function is then updated by propagating temporal constraints through the mission graph whose complexity is less than $O(|\mathcal{SU}|)$. Then, the overall complexity of one iteration is $O(|ST| \times |\mathcal{SU}| + (\#_{OC} + |\mathcal{SU}|) \times K \times TM)$ and the complexity of the centralized iterative algorithm is $O(\mathcal{IN} \times (|ST| \times |\mathcal{SU}| + (\#_{OC} + |\mathcal{SU}|) \times K \times TM))$. □

Algorithm 4 Decentralized Iterative Algorithm

Require: the agent $\mathcal{A}g_i$'s MDP, an initial set of policies $\pi =< \pi_1, \cdots \pi_n >$
Ensure: a new policy π_i for $\mathcal{A}g_i$
1: **repeat**
2: Compute the transition function of $\mathcal{A}g_i$ from π
3: $\pi_i' \leftarrow$ Revise the local policy using the decentralized revision algorithm
4: **for all** $\mathcal{A}g_j \in \mathcal{A}g, j \neq i$ **do**
5: Send π_i' to $\mathcal{A}g_j$
6: **end for**
7: nbChanges = 0
8: **for all** $\mathcal{A}g_k \in \mathcal{A}g$ **do**
9: nbChanges += number of changes in π_k
10: $\pi_k \leftarrow \pi_k'$
11: **end for**
12: **until** nbChanges == 0

*Convergence*Temporal complexity analysis proves that it is mainly influenced by the number of iterations. We end this section discussing convergence of the iterative process. We describe the influence of centralization, decentralization and opportunity cost computation on convergence guarantee.

First, it is proved that Equation 8 allows for choosing the action that maximizes the agents' joint utility.

Claim 8 *If expected opportunity cost accurately measures the influence of an action on the other agents' expected utility then, selecting the action to execute from a state s_i using Equation 8 maximizes the system' utility.*

Proof: See Appendix D

This claim assumes that an accurate measure of expected opportunity cost is used to revise policies. Indeed, the expected opportunity cost must correctly and accurately measure the influence of an action on the other agents' expected utility. While executing the centralized iterative algorithm, the execution policy of only one task is revised at the same time. Moreover, each time a task is revised, expected opportunity cost values are updated considering the new policy of the task. We thus guarantee that the expected opportunity cost used in Equation 8 always remains accurate and correct and it can be deduced that centralized iterative algorithm converges.

Claim 9 *If expected opportunity cost accurately measures the influence of an action on the other agents' expected utility then, the centralized iterative algorithm converges.*

Proof: Let task t_i be executed by agent $\mathcal{A}g_i$ and let π be the joint policy of the agents. While revising the execution policy of t_i (in a centralized way), the execution policy of all the other tasks remains unchanged. Let π_{-i} be the policy of the agents $\mathcal{A}g_j$ where $j \neq i$. For each state s_i from which t_i can be executed, Claim 8 proves that the revised policy of these states maximizes the joint expected utility given the policy π_{-i} of the other agents. Let π_i^{old} be the policy of the agent $\mathcal{A}g_i$ before revising the execution of t_i and let π_i^{new} be the policy of the agent $\mathcal{A}g_i$ after revising the execution of t_i. From Claim 8, we deduce:

$$\text{If } \pi^{old} \neq \pi^{new} \text{ then, } V^{\pi_{-i} \times \pi^{old}}(s_i) < V^{\pi_{-i} \times \pi^{new}}(s_i) \ \ \forall s_i$$

where $V^{\pi_{-i} \times \pi^{old}}$ is the expected gain of the agents while executing the joint policy $\pi_{-i} \times \pi^{old}$.

Thus, each time the execution policy of a task t_i is modified, the agent's joint utility increases. As the agents' utility is upper bounded (the upper bound is the sum of the rewards of the tasks), we deduce that the algorithm converges.□

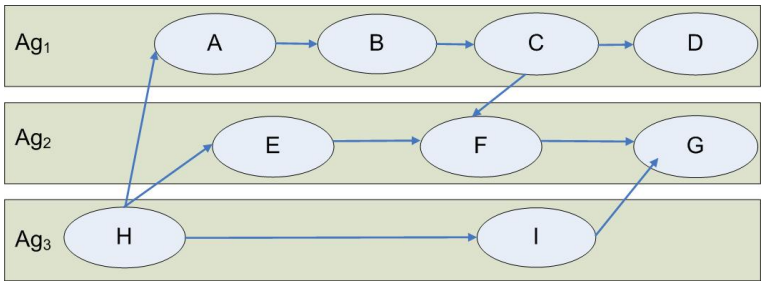

Fig. 6 A mission graph involving rescue agents

While executing the decentralized iterative algorithm, the execution policies of several tasks are revised at the same time. We cannot therefore guarantee the accuracy of expected opportunity cost. Figure 6 simplifies the mission graph presented on Figure 1. While executing the decentralized algorithm on this mission, tasks F and I are revised at the same time. The expected opportunity cost provoked by agent $\mathcal{A}g_3$ on $\mathcal{A}g_2$ is influenced by $\mathcal{A}g_2$'s available resources before the execution of G. These rely on the resources consumed to execute F and on the execution policy of F. As tasks F and I are revised at the same time, the accurate amount of resources available before the execution of G is not known by agent $\mathcal{A}g_3$. Thus, while revising the execution of I, the expected opportunity cost used in Equation 8 is not accurately known and we cannot guarantee that Equation 8 selects the action that maximizes the agents' joint utility. That's why convergence is not guaranteed.

*Performance*In order to remedy the high complexity of optimally solving DEC-MDPs, we have developed approximate algorithms that improve an initial set of policies. Experiments dealing with the quality of the resulting policies are presented in the next section. Nonetheless, properties of the computed solutions can be identified. We first introduce Bayesian games to define the kind of equilibrium achieved by the centralized iterative algorithm.

Our work deals with partially observable domains. Indeed, we consider multiagent systems where each agent does not exactly know the other agents' states nor actions. Bayesian games modelize decision problems in which information about the other players is incomplete. Each agent (or player) has private information that is relevant to the decision making process and that influences the expected utility of the system (Emery-Montemerlo et al., 2004). The private information held by each agent is called "type". In our approach, the type of an agent $\mathcal{A}g_i$ stands for his state. Each agent knows his state (or type) but he does not know the other agents's states. Computing the strategy for an agent then consists in finding a strategy that maximizes the agent's expected utility conditioned by the probability distribution over the other agents' states. Let u_i^{π} be the expected utility of the agent $\mathcal{A}g_i$ when strategy (policy) π is applied by the agents. A Bayesian Nash equilibrium is then such as:

$$\forall i \;\; u_i^{\pi}(s_i) \geq u_i^{\pi_{-i} \times \pi_i'}(s_i) \;\; \forall s_i \in \mathcal{S}_i \;\; \forall \pi_i'$$

where π_{-i} is the policy if the agents $\mathcal{A}g_j$ such as $j \neq i$ and:

$$u_i^{\pi}(s_i) = \sum_{s \in \mathcal{S} | s = \langle \cdots, s_i, \cdots \rangle} p(s|s_i) u_i^{\pi}(s)$$

where $p(s|s_i)$ is the probability that the system is in state s when $\mathcal{A}g_i$ is in s_i.

Claim 10 *Let $\mathcal{X}$ be a mission involving two agents $\mathcal{A}g_i$ and $\mathcal{A}g_j$. If all precedence constraints come from $\mathcal{A}g_i$ to $\mathcal{A}g_j$ then, the iterative algorithm computes an optimal policy.*

Proof: If $\mathcal{A}g_i$'s initial policy is EST-policy, it has been previously proved that the revision algorithm computes an optimal policy. While executing the iterative process, any initial policy leads to an optimal solution. Whatever $\mathcal{A}g_i$'s policy is initially considered, $\mathcal{A}g_i$'s policy resulting from the first iteration is EST-policy. Then, second iteration consists in executing the revision algorithm assuming $\mathcal{A}g_i$'s policy is EST-policy. This results in an optimal policy (see Claim 7).□

Like our iterative revision algorithm, co-alternative algorithms for solving DEC-POMDP iteratively improve the agents' policies until no more improvement is possible. Nonetheless, these algorithms, such as the Joint Equilibrium based Search for Policies (JESP) and the Dynamic Programming Joint Equilibrium based Search for Policies (DP-JESP) **?**, improve only one local policy at a time while our algorithm allows the agents' policies to be improved at the same time. Moreover, they do not take into account temporal and precedence contraints. Finally, improvement is centralized and the algorithms can solve only small sizes of problems: DP-JESP Nair et al. (2003), cannot be run for problems over horizon of 7. In order to increase the efficiency of these algorithms, a decentralized version of JESP, that exploits the locality of interactions has also been described Nair et al. (2005). This algorithm, called LID-JESP, allows for solving larger problems bu still remains limited to small results (4-agent problems that can be solved up to horizon 5 by JESP are solve up to horizon 6 by LID-JESP).

Our revision algorithms take advantage of precedence constraints to order policy revision. The policy of a task t_i is revised once the successor task of t_i has been considered. Similarly, the Global Optimal Algorithm (GOA) Nair et al. (2005) exploits the structure of the agents'interactions to order policy revision. GOA solve problems where interactions can be formalized by a tree structure. This algorithm allows each agent to compute his optimal policy given the policies of hiss children in the tree-structure. Although exploiting the structure of interactions speeds up the problem resolution, the optimal resolution limit the size of problems (4-agent problems that can be solved up to horizon 6 by LID-JESP are solve up to horizon 3 by GOA). As mentioned previously, SPIDER is an approximate algorithm that solves problems where interactions are formalized as a tree structure. Unlike our approach, SPIDER Varakantham et al. (2007) improves the policy of only one agent at a time. Although SPIDER can provide bounds on quality solutions and allows for considering larger set of agents (5 agents), experimental results are limited to problems up to horizon 4.

5 Experimental results and analysis

In order our approach to be used to solve realistic problems, it must be able to consider large problems and to find good approximate solutions. The following experiments first test the scalability and efficiency of our approach and then, describe its performance.

5.1 Scalability

Previous complexity analysis show that the efficiency of our algorithms depends on the state space and action space sizes of the local MDPs. Thus, first experiments deal with the influence of problems' parameters on the number of states and actions.

5.1.1 State space size

An upper bound on each agent's state space size can be computed considering the number of tasks to execute, the number of durations and resource consumptions per task, temporal and resource constraints.

Let $\#_{succ}(\mathcal{A}g_i)$ be the worst case number of success states of an agent $\mathcal{A}g_i$:

$$\#_{succ}(\mathcal{A}g_i) = \sum_{t_i \in \mathcal{T}_i} \#I(t_i) \times \#r_i$$

where $\#I(t_i)$ is the number of possible execution intervals of task t_i. Most of the time, the number of states of $\mathcal{A}g_i$ is less than $\#_{succ}(\mathcal{A}g_i)$ since not all resource rates are possible for each execution interval.

Similarly, the worst case number of partial failure states of an agent $\mathcal{A}g_i$ is:

$$\#_{EP}(\mathcal{A}g_i) = \sum_{t_i \in \mathcal{T}_i} |ET(t_{i-1})| \times \#r_i \times |ST(t_i)|$$

where $|ET(t_{i-1})|$ is the number of possible end times for the task t_{i-1} which is executed by $\mathcal{A}g_i$ before it starts t_i.

As there is one failure state associated with each task, the state space size $\mathcal{S}_i$ of agent $\mathcal{A}g_i$ is:

$$\begin{aligned} |\mathcal{S}_i| &= \#_{succ}(\mathcal{A}g_i) + \#_{EP}(\mathcal{A}g_i) + |\mathcal{T}_i| \\ &= \sum_{t_i \in \mathcal{T}_i} \Big(\#I(t_i) \times \#r_i + |ET(t_{i-1})| \times \#r_i \times |ST(t_i)| + 1 \Big) \end{aligned}$$

The state space size of each agent $\mathcal{A}g_i$ therefore relies on: the number of tasks $|\mathcal{T}_i|$ executed by $\mathcal{A}g_i$, the number of execution intervals per task, the number of possible resource rates per task, the number of start times and end times per task. Moreover, the number of intervals per task is strongly related to the number of start times and end times per task; and in the worst case $\#I(t_i) = |ST(t_i)| \times |ET(t_i)|$

Number of tasks The number of tasks each agent has to execute mainly influences the agents' state spaces. The more tasks an agent has to execute, the larger his state space is.

The number of tasks per agent relies on the number of agents involved in the mission. Given a set of tasks, the state space changes as the number of agents changes. Figures 7 and 8 describe the agents' state space sizes considering several missions where we increase the number of agents. We consider missions involving 20, 50, 100, 150 and 200 tasks. A peak in the number of states can be observed when starting to increase the number of agents. While increasing the number of agents that have to execute a set of tasks, the number of precedence constraints between two different agents increases. Thus, there are more dependencies between the agents and more partial failure states have to be considered. Figure 9 illustrates changes in the number of each kind of states while increasing the number of agents involved in a mission composed of 50 tasks. On the other hand, the number of success states decreases as there are less tuple $[t_i, I, r_{t_i}]$ to consider for each task t_i.

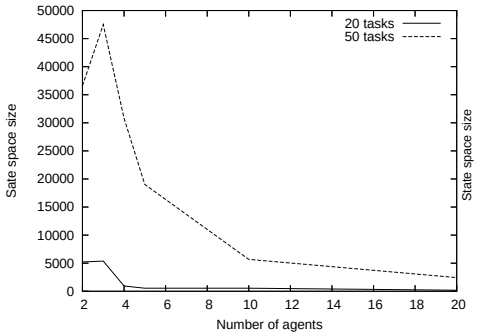

Fig. 7 Influence of the number of agents on the state space size

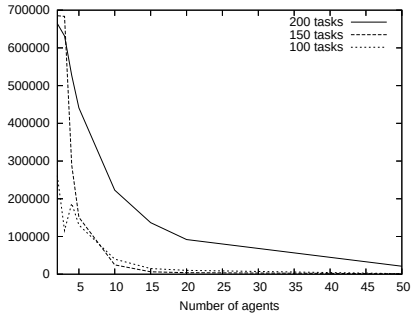

Fig. 8 Influence of the number of agents on the state space size

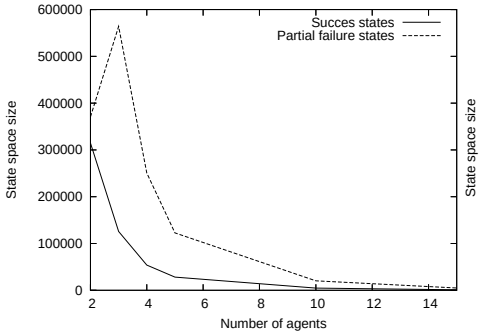

Fig. 9 Influence of the number of agents on the different kinds of states

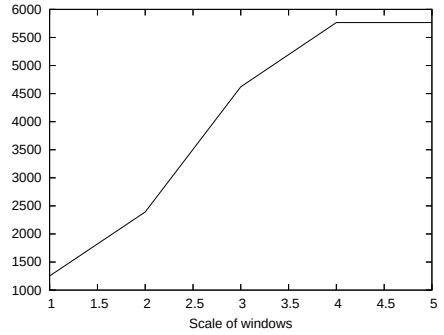

Fig. 10 Influence of temporal windows on the state space size

Number of intervals per task The number of execution intervals associated with a task t_i relies on the number of possible start times and end times of t_i which depends on: precedence constraints associated with t_i, temporal constraints associated with t_i, possible durations associated with t_i.

When we increase the size of the temporal windows [EST, LET], the state space size grows. Indeed, temporal constraints are less tight, and new execution plans (involving new states) can be considered. Figure 10 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 as large as the initial size. On Figure 10, the state space size rises and then, levels off at 4. Indeed, temporal windows become more and more large, and temporal constraints get more and more relaxed. While the size of temporal windows is multiply by 4, temporal windows do not constrain the execution of the agent any more. All the possible

execution plans (and possible states) are considered and the maximum of the state space size is reached. Keeping on relaxing the constraints does not increase the state space.

Precedence constraints also influence the number of intervals per task. In fact, they restrict or increase the number of possible start times and intervals of each task. Adding precedence constraints to a task t_i increase the number of t_i's predecessors. According to the end times of t_i's new predecessors, temporal propagation may result in new start times for t_i or it may remove some possible start times of t_i. Figure 11 illustrates changes in the number of states per agent while increasing the number of precedence constraints. One can observe that the number of states finally levels off. In fact, for each task t_i, all the possible start times have been considered and restricted at most. Thus, adding a precedence constraint does not add or remove any start time. The number of intervals therefore remains unchanged.

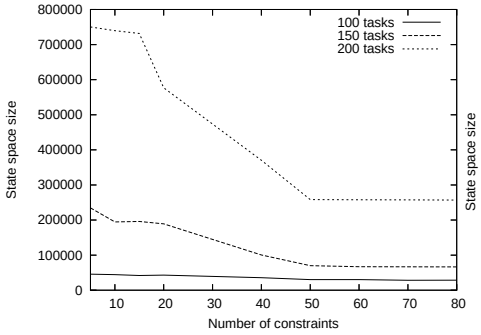

Fig. 11 Influence of precedence constraints on the state space size

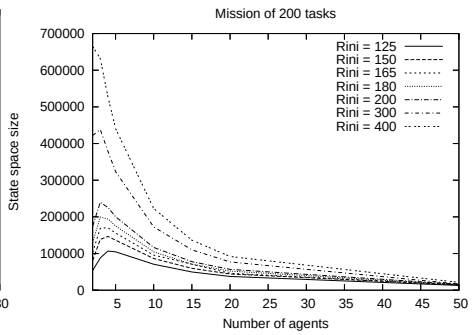

Fig. 12 Influence of initial resources on the state space size

Finally, execution durations influence the number of intervals per task. The more durations we consider for each task, the more intervals we obtain. Nonetheless, the number of intervals is restricted by temporal constraints which define lower and upper bounds on intervals bounds.

Number of resource rates per task The peak we have previously observed while increasing the number of agents also relies on resources. Let R_{ini} be the amount of resources which is initially available to each agent. When we increase the number of agents, each agent $\mathcal{A}g_i$ has less tasks to execute and initial resources become wider given the set of tasks $\mathcal{A}g_i$ must execute. Lacks of resources become scarce and the number of possible resource rates which are greater than zero and that have to be considered, increases. Thus, when we increase initial resources, agents do not lack of them and there are more possible resource consumptions to consider for each task. That's why we observe a peak. As shown on Figure 12, under tight initial resource rates ($R_{ini} \in [165, 300[$), we observe a peak in the number of states. Let R_{ini} be 165. If the number of agents that must execute the mission increases, the agents have the same initial resource rate ($R_{ini} = 165$) to execute less tasks. Then, resources become wider and more positive resource rates have to be considered. Therefore, the number of states

increases and there is a peak. When resources are enough large, increasing the number of agents (i.e. decreasing the number of tasks per agents) does not add new possible resource rates and there is no peak. Indeed, all the possible resource rates are computed even for small numbers of agents. If we increase the number of agents no other resource rate is computed and there is no peak in the number of states. Moreover, each agent has less tasks to execute and the state space size diminishes.

5.1.2 Action space size

The action set of each agent $\mathcal{A}g_i$ relies on the number of tasks which are executed by $\mathcal{A}g_i$ and the number of possible start times for each task. Temporal and precedence constraints influence the number of possible start times per task. In fact, temporal constraints restrict the number of possible start times of each task. Moreover, precedence constraints increase or decrease the number of possible start times. Note that adding new agents to a mission increases dependencies between the agents. This increases the number of precedence constraints between different agents thus, influencing the number of possible start times of each task. On the other hand, adding new agents to a mission decreases the number of tasks per agent. Table 1 describes the influence of the number of agents on the number of actions the agents have to consider.

Scalability experiments show that many parameters influence the state and action spaces of each agent. It is therefore difficult to *a priori* evaluate the number of states and actions that have to be considered by the revision algorithm. Despite the wide range of parameters to consider, a rough estimate of state space sizes can be given. Considering a mission of 200 tasks and 3 agents, MDPs composed of about 250 000 states are obtained. Increasing initial resources leads to MDPs composed of 700 000 states. Nonetheless, adding temporal constraints and agents to the mission allows for limiting the MDPs' state spaces. Thus, problems involving 800 tasks, 20 agents and 700 constraints have led to MDPs composed of about 57 200 states.

5.2 Efficiency

Based on scalability experiments, we have tested our algorithms on different problems. We considered a benchmark composed of several sizes of missions (20, 50, 10, 150 and 200 tasks, from 2 to 50 agents) where we varied the number of precedence constraints, temporal windows and initial resources. Running time of the centralized revision algorithm has then been studied. Tables 1 and 2 describe the influence of the numbers of actions and states on the running time of the centralized revision algorithm[1]. Solving a problem involving 30 000 states usually takes less than 15 seconds. Revising about 150 000 states takes less than 5 minutes and revising a million of states takes between one and two hours.

Experiments have also be run using the decentralized revision algorithm. The time gained using decentralization is mainly influenced by the number of tasks that can be revised at the same time and by the time needed to communicate opportunity cost values. On small problems, running times of centralized and decentralized algorithms are quite the same. In fact, since there are few agents, few tasks can be revised at the same time. If communication takes a long time, the decentralized algorithm runs slower than the centralized algorithm since communication is very time consuming. When large problems are considered (more

[1] Algorithms have been executed on a computer equipped with Pentium III, 700 MHz.

Number of tasks	Number of agents	State space size	Number actions	Running time (s.)
50	2	73 202	222 696	138
50	3	142 477	153 336	95
50	4	123 189	208 301	154
50	5	95 172	326 907	292
50	10	56 873	702 553	312
50	15	48 697	548 435	350

Table 1 Running time of centralized revision algorithm

Number of tasks	Number of agents	Running time
20	2	8s.
20	10	8s.
50	2	138s.
50	10	312 s.
100	2	15 min
100	10	20 min
200	2	1h30

Table 2 Running time of centralized revision algorithm

than 50 tasks), the decentralized algorithm allows for gaining time if the problem involves many agents and they can revise their tasks at the same time. Gain of several minutes (from 2 to 3 minutes on our experiments) have been recorded for problems involving a hundred tasks and ten agents.

These experiments prove that large missions can be solved using the revision algorithm. We have then studied the efficiency of the iterative algorithm. We have therefore been interested in the number of iterations needed to converge. One iteration of the iterative algorithm consists in executing the revision algorithm and updating the transition function. Nonetheless, the time needed to update transition functions is negligible compared to the running time of the revision process. For instance, updating the transition functions of a problem composed of 20 tasks and 3 agents takes about 5 milliseconds whereas the revision algorithm takes about 4 seconds. The running time of one iteration is therefore mainly influenced by the revision algorithm's running time.

Experiments dealing with the number of iterations have highlighted the influence of initial resources on the number of iterations. Figure 13 relates the number of iteration steps to the initial resource rate considering that EST-policy is the initial policy. With large or unlimited resources, only one iteration step is needed to converge. As the initial resource rate decreases, the number of iteration steps increases since it reaches a maximum which corresponds to a critical resource rate. If the agents initially have less resources than this critical rate, they will not be able to execute all the tasks. Whatever their policy, the latest tasks cannot be executed because of a lack of resources. Then, all the possible policies of these tasks are equivalent and there is no strictly better policy than the initial EST-policy. Figure 14 describes the relationship between the initial resource rate and the number of policy changes of each iteration. If resources are large or unlimited, there is no change. Indeed, we proved that the initial policy is an optimal policy. As the initial resource rate decreases, more and more changes are needed to obtain the solution. If initial resources

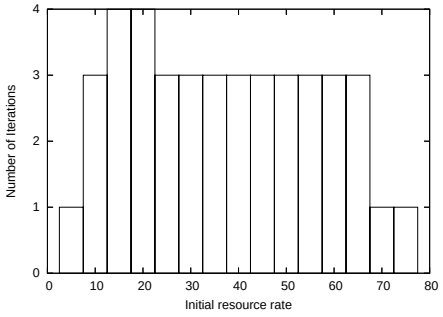

Fig. 13 Influence of resources

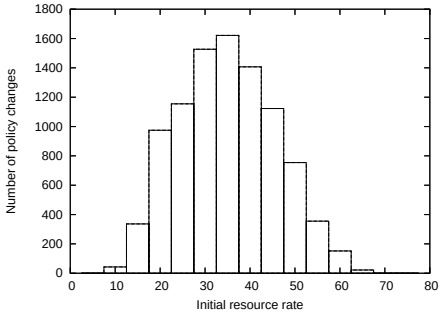

Fig. 14 Influence of resources

are low, the policy of the latest task remains unchanged and few changes are necessary. The number of policy changes per iteration has also been studied. Most changes are made during the earliest iterations. Then, the number of policy changes per iteration diminishes until it becomes null and convergence is reached.

While experimenting the number of iterations to converge, the iterative algorithm have been run for planning each mission of our benchmark. Most of the time, it takes less than four iterations for the centralized and decentralized algorithms to converge. Although convergence of the decentralized algorithm is not guaranteed, it never diverges while considering our benchmark. Divergence seems therefore to be unusual.

5.3 Performance

Finally, the performances of our solutions have been studied by running mission executions. We have compared the performances obtained at each iteration step. The performances of our approach have also been compared with other existing works. The following experiments have been developed using the previously described benchmark.

We have studied how the quality of the solutions evolves at each iteration step. We have thus run the policies that are computed ate each iteration and we have studied the gain of the agents (sum of the rewards obtained by the agents). Experiments demonstrate that the performances of the agents increase with the number of iterations. Moreover, it can be shown that first iteration achieves largest improvements. Subsequent iterations reduce the number of partial failures and often lead to maximum gain. By iterating the process, the likelihood the agents fail because of lack of resources decreases. The resulting policy is safer than policies of previous iteration steps. Figure 15 plots the number of partial failures of the agents over 1000 executions. Note that experiments described on Figure 15 are developed assuming that first iteration's initial policy is EST-policy. At the first iteration, revision of policies always consists in delaying the execution of the tasks. That's why the number of partial failures at "iteration 1" is greater than the number of partial failures at "iteration 0". A near optimal policy is obtained at the end of the first iteration. Then, the second iteration leads to small improvements but it diminishes the number of partial failures. When resources are tight, more iterations are needed to converge. Now, more benefits are gained

from re-iterating. As soon as the solution produces the maximum gain, re-iterating reduces the number of partial failures.

Because of the high complexity of the problems that we consider, only small problems can be optimally solved in practice. While considering problems involving more than two agents and ten tasks, the optimal solution cannot be computed and our solutions cannot be compared with the optimum. Let OC-policy be the solution computed by the revision algorithm. In order to test the performance of our approach, OC-policies have been compared with the performances of three heuristic policies which have been proved to be optimal under some assumptions. The first one is EST-policy, the second policy is LST-policy and the third policy (P-policy) selects the most likely start time of each task. In fact, for each start time st of a task t_i, it computes the probability that the predecessors of t_i end at t_i and it selects the start time with the highest probability. Performances of these policies have been compared considering the number of partial failures of the agents and the gain they obtain (the sum of the rewards obtained by the agents) over 1000 executions.

Experiments show that OC-policy outperforms heuristic policies. Figure 16 exemplifies the agents' performance considering a scenario involving 2 agents and 20 tasks. The number of partial failures of the agents is quiet small while executing OC-policy. Indeed, over 1000 executions, the agents move 113 times to a partial failure and they obtain the maximum gain.

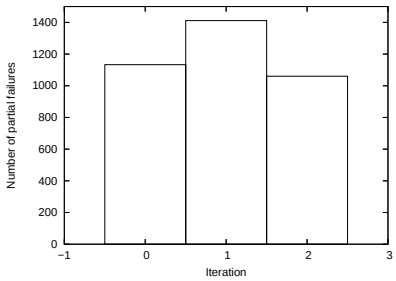

Fig. 15 Number of partial failures per iteration

	EST-pol.	LST-pol.	P-pol.	OC-pol.
Partial failures	519	0	114	113
Gain	8274	9600	9888	10059

Fig. 16 Heuristic methods vs OC-DEC-MDP

Partial failures arise from miscoordination between the agents: an agent starts to execute a task t_i before the predecessors of t_i have finished their execution. Since partial failures are time and resource consuming, the agent must avoid such failures. If we consider the number of partial failures, the worst case arises for EST-policy. Indeed, each agent chooses to start the execution of its next task as soon as possible, even if the probability to fail is high. In the worst case, the agent will try to execute its task at each possible start time. On the other hand, LST-policy never leads to a partial failure: the agent only tries once to execute its task, if he fails he could not retry later because there is no more possible start time (permanent failure).

The more initial resources the agents have, the closer EST-policy is to OC-policy. If initial resources are high, the cost of a partial failure is negligible and the OC-policy tries to execute the task at each possible start time (like EST-policy). Then, these policies are the same. Under unlimited resources, these policies are optimal.

The difference between the gain of OC-policy and the gain of P-policy, relies on temporal constraints and on the probability distributions on durations. If P-policy is safe (the probability to move to a permanent failure is low) and the expected opportunity cost of such a policy is low, P-policy is close to OC-policy.

The performance of our approach have also been compared with the optimum on small size of problems. The Coverage Set Algorithm (CSA) developed by Becker et al. (2004b) is the only algorithm that is able to solve DEC-MDPs with complex constraints and several durations for each task. Despite the wide range of problems solved by CSA (Becker et al., 2003, 2004a), only small problems can be solved in practice. While considering the kind of missions we deal with, CSA can only solve problems involving two agents and one way precedence constraints, (precedence constraints between the agents always come from the first agent to the second one). Such problems have been proved to be optimally solved by our approach (Claims 7 and 10). Thus, our approach performs as well as CSA on the problem CSA and our algorithms both can solve.

It is difficult to compare the performance of our approach on larger sizes of problems. We have in fact developed the first approach that can deal with large problems and several kinds of constraints. Even approximate approaches only solve small problems. Moreover these methods do not deal with constraints and several durations so, we were not able to compare our results even on small size of problems. Despite the lack of comparison, experiments show that most of the time, the iterative algorithm allows the agents to obtain the maximum reward and the Bayesian Nash equilibrium achieved by the centralized iterative algorithm seems often to be very closed to optimal.

6 Conclusion

The framework of DEC-MDPs has been proposed to solve decision problems in cooperative multiagent systems. Nonetheless, DEC-MDPs assume a simple model of time and actions and they suffer from a high complexity. It is therefore difficult to formalize and solve large multiagent decision problems with complex constraints, like multi-robot decision problems. The framework of ED-DEC-MDPs (Becker et al., 2004a) has been the only attempt to increase the expressiveness of DEC-MDPs considering contraints on task execution. Nevertheless, ED-DEC-MDPs suffer from large state spaces that are exponential in the number of dependencies and make them intractable for large problems. Due to the high complexity of optimally solving DEC-MDPs, recent works have focused on developing approximate approaches. Nonetheless, effective methods to solve large DEC-MDPs are still lacking. Indeed, approximate approaches are able to solve larger problems than optimal approaches but they remain limited to small sizes of problems (about 2 agents and 10 tasks). In order to increase the applicability of DEC-MDP based approaches, our purpose has been to propose a model that can deal with more complex time and action representations; and to develop algorithms that efficiently solve large problems with respect to constraints on task execution.

We proposed a new model, OC-DEC-MDP, that allows for representation of temporal, precedence and resource constraints. In order to deal with large missions, the multiagent decision problem has been broken into a set of MDPs which represent the agent's decision problems. Full definition of MDPs requires the system's transition function to be decomposed. Because of dependencies between the agents such decomposition is not easy. We have therefore considered an initial set of policies and individual transitions have been computed assuming that each agent follows its initial policy.

Once the problem was formalized, we tackled policy computation. Given the high complexity of finding an optimal solution, we turned to an approximate approach. We then tried to improve the initial policy set which has been used to define individual transition functions. Opportunity cost has been introduced in order to coordinate the agents. We have defined expected opportunity cost to better estimate the influence of an action on the other agents. Thus, each decision of an agent $\mathcal{A}g_i$ results from a trade-off between the expected utility of $\mathcal{A}g_i$ and the opportunity cost provoked on the other agents. It has been proved that, if the expected opportunity cost accurately measures the influence of an action on the other agents then, each agent chooses the action that maximizes the expected gain of the system.

We then developed a revision algorithm that applies this decision trade-off to improve the initial policy set. This revision process has been iterated to obtain higher quality solutions and to allow a Bayesian Nash equilibrium to be reached. Finally, we analysed the complexity of our approach and the quality of solutions. Thus, we have pointed out some properties that guarantee optimal policy computation. Complexity analysis proved that our approach is polynomial time in the number of states and actions whereas other approaches are exponential. Moreover, experiments have shown that constraints on task execution limit the state and action spaces. Our algorithms are therefore able to solve large problems composed of hundreds of tasks and more than ten agents.

Experimental results, complexity and quality analysis have shown that our approach fulfils initial ambitions. Indeed, we have developed efficient algorithms that can deal with large missions and compute good quality solutions respecting several kinds of constraints. Future work will first aim at increasing the expressiveness of OC-DEC-MDPs. Thus, we plan to consider a wide variety of problems. The agents would therefore be able to choose between several possible tasks. We also plan to relax the assumption about the order of the tasks of each agent to allow the agents to choose whether execute t_i or t'_i first. We could also extend the range of constraints formalized by our approach and the expressiveness of the mission. We also plan to improve the performance of the system. Marecki and Tambe (2007) have proposed an heuristic solution to speed up OC-DEC-MDP resolution and to achieve better solution qualities. Nevertheless, they do not take into account resource consumptions nor partial failures. In order to increase the quality of the policies computed by approach, we plan to allow the agents to communicate during task execution. Indeed, sometimes communication during task execution is possible. In fact, it is restricted by temporal windows. Then, on-line information exchange could improve decision making and could lead to higher performance (Nair et al., 2004). The agents should therefore trade-off communication cost and relevance of communicated information (Becker et al., 2005; Goldman and Zilberstein, 2004).

References

Abdallah, S., Lesser, V., 2005. Modeling Task Allocation Using a Decision Theoretic Model. In: Proceedings of Fourth International Joint Conference on Autonomous Agents and Multiagent Systems. ACM Press, Utrecht, Netherlands, pp. 719–726.

Amato, C., Carlin, A., Zilberstein, S., 2007a. Bounded dynamic programming for decetralized pomdps. In: AAMAS 2007 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains.

Amato, C., D.S., B., Zilberstein, S., 2007b. Optimizing memory-bounded controllers for decentralized pomdps. In: Proceedings of the Twenty Third Conference on Uncertainty in Artificial Intelligence.

Becker, R., Lesser, V., Zilberstein, S., 2004a. Decentralized Markov Decision Processes with Event-Driven Interactions. In: The Third International Joint Conference on Autonomous Agents and Multi Agent Systems. Vol. 1. IEEE Computer Society, NYC, pp. 302–309.

Becker, R., Lesser, V., Zilberstein, S., September 2005. Analyzing Myopic Approaches for Multi-Agent Communication. In: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 05). IEEE Computer Society, Compiegne, France, pp. 550–557.

Becker, R., Zilberstein, S., Lesser, V., Goldman, C., July 2003. Transition-independent decentralized markov decision processes. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems. Melbourne, Australia, pp. 41–48.

Becker, R., Zilberstein, S., Lesser, V., Goldman, C., December 2004b. Solving transition independent decentralized markov decision processes. Journal of Artificial Intelligence Research 22, 423–455.

Bernstein, D., Hansen, E.A., Zilberstein, S., 2005. Bounded policy iteration for decentralized pomdps. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence. Edinburgh, Scotland.

Bernstein, D., Zilberstein, S., Immerman, N., 2002. The complexity of decentralized control of mdps. In: Mathematics of Operations Research. pp. 27(4):819–840.

Bernstein, D., Zilberstein, S., Washington, R., Bresina, J., 2001. Planetary rover control as a markov decision process. In: The 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space. Montreal, Canada.

Blythe, J., 1999a. Decision-theoretic planning. AI Magazine.

Blythe, J., 1999b. Planning under uncertainty in dynamic domains. Phd thesis, Carnegie Mellon University.

Boutilier, C., Brafman, R., Geib, C., 1997. Prioritized goal decomposition of Markov decision processes: Towards a synthesis of classical and decision theoretic planning. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann, San Francisco, pp. 1156–1163.

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

Bresina, J., Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D., Washington, R., 2002. Planning under continuous time and resource uncertainty: A challenge for ai. In: UAI.

Chadès, I., Scherrer, B., Charpillet, F., 2002. A heuristic approach for solving decentralized-POMDP: Assessment on the pursuit problem. In: Proceedings of the Sixteenth ACM Symposium on Applied Computing.

Dean, T., Lin, S., 1995. Decomposition techniques for planning in stochastic domains. In: IJCAI-95.

Decker, K., Lesser, V., 1993. Quantitative modeling of complex environments. International Journal of Intelligent Systems in Accounting Finance and Management 2 (4), 215–234.

Emery-Montemerlo, R., Gordon, G., Schneider, J., Thrun, S., 2004. Approximate solutions for partially observable stochastic games with common payoffs. In: Proceedings of the Third Joint Conference on Autnomous Agents and Multi Agent Systems.

Esben, H. O., Maja, J. M., Gaurav, S. S., 2002. Multi-robot task allocation in the light of uncertainty. In: Proceedings of IEEE International Conference on Robotics and Automation. pp. 3002–3007.

Gerkey, B. P., Matarić, M. J., 2002. Sold!: Auction methods for multi-robot coordination. IEEE Transactions on Robotics and Automation 18 (5), 758–768.

Goldman, C., Zilberstein, S., 2003. Optimizing information exchange in cooperative multiagent systems. In: International Joint Conference on Autonomous Agents and Multi Agent Systems. pp. 137–144.

Goldman, C., Zilberstein, S., 2004. Decentralized control of cooperative systems: Categorization and complexity analysis. Journal of Artificial Intelligence Research 22, 143–174.

Hanna, H., Mouaddib, A., 2002. Task selection as decision making in multiagent system. In: International Joint Conference on Autonomous Agents and Multi Agent Systems. pp. 616–623.

Hansen, E.A., Bernstein, D., Zilberstein, S., 2004. Dynamic programming for partially observable stochastic games. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence.

Howard, R. A., 1960. Dynamic Programming and Markov Processes. MIT Press.

Koller, D., Milch, B., 2003. Multi-agent influence diagrams for representing and solving games. Games and Economic Behavior, 45(1): 181–221.

Marecki, J., Tambe, M., 2007. On opportunistic techniques for solving decentralized mdps with temporal constraints. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS).

Meuleau, N., Hauskrecht, M., Kim, K.-E., Peshkin, L., Kaelbling, L., Dean, T., Boutilier, C., 1998. Solving very large weakly coupled markov decision processes. In: AAAI/IAAI. pp. 165–172.

Morimoto, T., 2000. How to develop a RoboCupRescue agent. RoboCupRescue Technical Committee.

Nair, R., Pradeep, V., Milind, T., Makoto, Y., 2005. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05).

Nair, R., Roth, M., Yokoo, M., Tambe, M., 2004. Communication for improving policy computation in distributed pomdps. In: Proceedings of the Third International Joint Conference on Agents and Multiagent Systems (AAMAS-04). pp. 1098–1105.

Nair, R., Tambe, M., Yokoo, M., Marsella, S., Pynadath, D.V., 2003. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In: Proceedings of the International Joint Conference on Artificial Intelligence. pp. 705–711.

Peshkin, L., Kim, K., Meuleu, N., Kaelbling, L., 2000. Learning to cooperate via policy search. In: Sixteenth Conference on Uncertainty in Artificial Intelligence. pp. 307–314.

Poupart, P., Boutilier, C., Patrascu, R., Schuurmans, D., 2002. Piecewise linear value function approximation for factored mdps. In: Eighteenth National Conference on Artificial Intelligence. Edmonton.

Puterman, M. L., 2005. Markov Decision processes : discrete stochastic dynamic programming. Wiley-Interscience, New York.

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

Roy, N., Pineau, J., Thrun, S., 2000. Spoken dialogue management using probabilistic reasoning. In: Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000). Hong Kong.

Seuken, S., Zilberstein, S., 2007. Memory-bounded dynamic programming for dec-pomdps. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI). pp. 2009–2015.

Singh, S., Cohn, D., 1998. How to dynamically merge markov decision processes. In: Advances in Neural Information Processing Systems. Vol. 10. The MIT Press.

Szer, D., Charpillet, F., Zilberstein, S., 2005. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence.

The RoboCup Rescue Technical Committee, 2000. RoboCup-Rescue simulator manual.

Varakantham, P., Marecki, J., Yabu, y., Milind, T., Makoto, Y., 2007. Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies. In: Proceedings of the International Joint Conference on Agents and Multiagent Systems (AAMAS-07).

Wieser, F., 1889. Valeur naturelle (Der natrliche Wert).

Xuan, P., Lesser, V., Zilberstein, S., January 2001. Communication decisions in multiagent cooperation : Model and experiments. In: Proceedings of the Fifth International Conference on Autonomous Agents. ACM Press, Montreal, pp. 616–623.

A Proof of Claim 4

Under unlimited resources, the revision algorithm computes an optimal policy.

Proof: Under unlimited resources, decisions are not influenced by available resources. Thus, expected utility and opportunity cost values do not depend on resources. Moreover, partial failures do not penalize the agents. Indeed, as resources are unlimited, partial failures' resource consumptions do not increase the probability of failing because of insufficient resources.

When an agent $\mathcal{A}g_i$ delays the execution of its task t_{i+1}, he increases the probability of failing because of deadline met. Although the agents reduces the probability of partially failing, the probability of failing because of insufficient ressources remains unchanged. It can therefore be deduced that the agent's expected utility decreases when he delays the execution of his tasks. Moreover the opportunity cost increases as the delay of a task increases. Let st and st' be two possible start times ($st < st'$) for a task t_{i+1} executed by agent $\mathcal{A}g_i$ and let V'_{st} be $\mathcal{A}g_i$'s expected utility when he starts to execute t_{i+1} at st. Then,

$$V'_{st} \geq V'_{st'} \text{ and } OC(t_{i+1}, st) \leq OC(t_{i+1}, st')$$

As the policy of each task t_{i+1} is computed using Equation 8, it can be deduced that the policy computed by the revision algorithm consists in starting each task as soon as possible (EST-policy). In fact, for each state from which a task t_{i+1} can be executed, the start time which maximizes Equation 8 is the earliest start time of t_{i+1}.Under unlimited resources, it has been proved that EST-policy is an optimal policy (see Claim 1). Then, it can be deduced that, under unlimited resources, the revision algorithm computes an optimal policy. □

B Proof of Claim 5

If there is no constraint on tasks' end times ($LET \simeq +\infty$), the revision algorithm computes an optimal policy.

Proof: If there is no constraint on tasks' end times, the agents do not fail because of deadline met. When an agent $\mathcal{A}g_i$ delays the execution of its task t_{i+1}, the probability of partially failing decreases. Moreover, the probability of failing because of insufficient resources decreases. On the other hand, the probability of failing because of deadline met does not increase. It can therefore be deduced that the agent's expected value increases when the agent delays the execution of his tasks. As the highest expected value is obtained for the largest delay, opportunity cost is equal to zero whatever the delay. Let st and st' be two possible start times ($st < st'$) for a task t_{i+1} executed by agent $\mathcal{A}g_i$ and let V'_{st} be $\mathcal{A}g_i$'s expected utility when he starts to execute t_{i+1} at st. Then,

$$V'_{st} \leq V'_{st'} \text{ and } OC(t_{i+1}, st) = OC(t_{i+1}, st') = 0$$

As the policy of each task t_{i+1} is computed using Equation 8, we can deduce that the policy computed by the revision algorithm consists in starting each task as late as possible (LST-policy). In fact, for each state from which a task t_{i+1} can be executed, the start time which maximizes Equation 8 is the latest start time of t_{i+1}. If there is no constraint on tasks' end times, it has been proved that LST-policy is an optimal policy (see Claim 2). Then, it can be deduced that the revision algorithm computes an optimal policy when there is no constraint on tasks' end times. □

C Proof of Claim 7

Let $\mathcal{X}$ be a mission involving two agents $\mathcal{A}g_i$ and $\mathcal{A}g_j$. If the initial policy of $\mathcal{A}g_i$ is EST-policy and all precedence constraints come from $\mathcal{A}g_i$ to $\mathcal{A}g_j$ then, the revision algorithm computes an optimal policy.

Proof: If there is no precedence constraints from $\mathcal{A}g_j$ to $\mathcal{A}g_i$, agent $\mathcal{A}g_i$ never partially fails. Because $\mathcal{A}g_i$ does not have to wait for other agents, he can start to execute his tasks as soon as possible. Thus, he minimizes the probability of meeting the deadline and the delay provoked on the other agent $\mathcal{A}g_j$. $\mathcal{A}g_i$'s optimal policy is therefore EST-policy.

Let st and st' be two possible start times ($st < st'$) for a task t_{i+1} executed by agent $\mathcal{A}g_i$ and let V'_{st} be $\mathcal{A}g_i$'s expected utility when he starts to execute t_{i+1} at st. When $\mathcal{A}g_i$ delays the execution of its task t_{i+1}, his expected utility decreases and the opportunity cost provoked on $\mathcal{A}g_j$ increases. Thus,

$$V'_{st} \geq V'_{st'} \text{ and } OC(t_{i+1}, st) \leq OC(t_{i+1}, st')$$

Computing $\mathcal{A}g_i$'s policy using Equation 8 leads to EST-policy. If the initial policy of $\mathcal{A}g_i$ is EST-policy, this remains unchanged while executing the revision algorithm. $\mathcal{A}g_j$'s policy is therefore computed assuming $\mathcal{A}g_i$'s optimal policy. As there is no precedence constraints from $\mathcal{A}g_j$ to another agent, modifying $\mathcal{A}g_j$'s policy does not influence any agent and the opportunity cost provoked by $\mathcal{A}g_j$ is equal to zero. If the initial policy of $\mathcal{A}g_i$ is EST-policy the OC-DEC-MDP resolution consists in computing the policy that maximizes $\mathcal{A}g_j$'s expected value given $\mathcal{A}g_i$'s optimal policy. As the opportunity cost provoked by $\mathcal{A}g_j$ is equal to zero, Equation 8 maximizes $\mathcal{A}g_j$'s expected utility and allows for computing $\mathcal{A}g_j$'s optimal policy. Because $\mathcal{A}g_j$ does not influence any agent, his optimal policy is individually optimal and jointly optimal. The revision algorithm therefore computes the optimal policy of each agent. □

D Proof of Claim 8

The opportunity cost provoked on $\mathcal{A}g_j$ when $\mathcal{A}g_i$ fails executing t_{i+1} is defined by:

$$EOC_{\mathcal{A}g_j,t_{i+1}}(fail) = OC_{t_j}(fail) = \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) V^0_{t_j,r_{t_j}} - V([failure_{t_j}, *, *])$$

Given transition probabilities' computation and the definition of the expected opportunity cost provoked by a partial failure, Equation 10 can be re-written as follows:

$$\begin{aligned}
OC(t_{i+1}, st) = & \overbrace{P^r_{enough}(Pred(t_{i+1})).\prod_{a \in P_{red}(t_{i+1})-t_i} \sum_{s \leq st} P^a_{\mathcal{ET}}(s|et(I')_{t_i})}^{\text{Probability of success}} \\
& \overbrace{. \sum_{\Delta r | r \geq \Delta_r^{i+1}} \sum_{\delta_c^{i+1} | st+\delta_c^{i+1} \leq LET} P_r(\Delta_r^{i+1}).P_c(\delta_c^{i+1})}^{\text{Probability of success}} . \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} EOC_{\mathcal{A}g_j,t_{i+1}}(et_{i+1}) \\
& + \overbrace{(P_{LR} + P_{TT} + P_{DM})}^{\text{Failure probability}} \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} (\sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) V^0_{t_j,r_{t_j}} - V([failure_{t_j}, *, *])) \\
& + P_{PPC-fail} \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} (\sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) V^0_{t_j,r_{t_j}} - V([failure_{t_j}, *, *])) \\
& + \sum_{et'(t_{i+1}) > et(t_{i+1})} P_{PPC-suc}(et'(t_{i+1})) \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} EOC_{\mathcal{A}g_j,t_{i+1}}(et'(t_{i+1}))
\end{aligned}$$

where $P_{PPC-fail}$ is the probability that the execution of t_{i+1} fails after one partial failure or more. $P_{PPC-suc}(et'(t_{i+1}))$ is the probability that the execution of t_{i+1} succeeds after one partial failure or more and the task ends at $et'(t_{i+1})$. Thus,

$$P_{PPC} = \sum_{et'(t_{i+1}) > et(t_{i+1})} P_{PPC-suc}(et'(t_{i+1})) + P_{PPC-fail}$$

Moreover,

$$EOC_{\mathcal{A}g_j,t_{i+1}}(et_{t_{i+1}}) = max_{t_k \in Suc(t_{i+1})} \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) OC_j(et_{t_{i+1}} - LB_k, r_{t_j})$$

For purpose of good understanding, it is assumed that $t_j = t_k$. If $t_j \neq t_k$, a similar proof can be done by recursively applying Equations 11 and 12.

We deduce that:

$$\begin{aligned}
EOC_{\mathcal{A}g_j,t_{i+1}}(et_{t_{i+1}}) &= \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) OC_j(et_{t_i} - LB_k, r_{t_j}) \\
&= \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) (V_{t_j}^{0,r_{t_j}} - V_{t_j}^{\Delta t, r_{t_j}}) \\
&= \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) V_{t_j}^{0,r_{t_j}} - \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) V_{t_j}^{\Delta t, r_{t_j}} \\
&= \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) \cdot V_{t_j}^{0,r_{t_j}} - \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) V_{t_j}^{\Delta t, r_{t_j}}
\end{aligned}$$

We thus obtain:

$$EOC_{\mathcal{A}g_j,t_{i+1}}(et_{t_{i+1}}) = \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) V_{t_j}^{0,r_{t_j}} - \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) V_{t_j}^{\Delta t, r_{t_j}}$$

where $\Delta t = et_{t_{i+1}} - LB_j$

Then,

$$EOC(t_{i+1}, st) = \overbrace{P^r_{enough}(Pred(t_{i+1})).\prod_{a\in P_{red}(t_{i+1})-t_i}\sum_{s\leq st} P^a_{\mathcal{ET}}(s|et(I')_{t_i})}^{\text{Probability of success}}$$
$$\overbrace{.\sum_{\Delta r|r\geq\Delta^{i+1}_r}\sum_{\delta^{i+1}_c|st+\delta^{i+1}_c\leq LET} P_r(\Delta^{i+1}_r).P_c(\delta^{i+1}_c)}^{\text{Probability of success}}.$$
$$\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}(\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^{0,r_{t_j}}_{t_j} - \sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^{\Delta t,r_{t_j}}_{t_j})$$
$$+\overbrace{(P_{LR}+P_{TT}+P_{DM})}^{\text{Failure Probability}}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^0_{t_j,r_{t_j}}$$
$$-\overbrace{(P_{LR}+P_{TT}+P_{DM})}^{\text{Failure Probability}}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i} V([failure_{t_j},*,*])$$
$$+P_{PPC-fail}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}(V^0_{t_j} - V([failure_{t_j},*,*]))$$
$$+\sum_{et'(t_{i+1})>et(t_{i+1})} P_{PPC-suc}(et'(t_{i+1}))\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^{0,r_{t_j}}_{t_j}$$
$$-\sum_{et'(t_{i+1})>et(t_{i+1})} P_{PPC-suc}(et'(t_{i+1}))\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^{\Delta t,r_{t_j}}_{t_j}$$

$$EOC(t_{i+1}, st) = \overbrace{P^r_{enough}(Pred(t_{i+1})).\prod_{a\in P_{red}(t_{i+1})-t_i}\sum_{s\leq st} P^a_{\mathcal{ET}}(s|et(I')_{t_i})}^{\text{Probability of success}}$$
$$\overbrace{.\sum_{\Delta r|r\geq\Delta^{i+1}_r}\sum_{\delta^{i+1}_c|st+\delta^{i+1}_c\leq LET} P_r(\Delta^{i+1}_r).P_c(\delta^{i+1}_c)}^{\text{Probability of success}}.\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^{0,r_{t_j}}_{t_j}$$
$$+\overbrace{(P_{LR}+P_{TT}+P_{DM})}^{\text{Failure Probability}}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^0_{t_j,r_{t_j}}$$
$$+\sum_{et'(t_{i+1})>et(t_{i+1})} P_{PPC-suc}(et'(t_{i+1}))\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^{0,r_{t_j}}_{t_j}$$
$$+P_{PPC-fail}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^0_{t_j,r_{t_j}}$$
$$-\overbrace{P^r_{enough}(Pred(t_{i+1})).\prod_{a\in P_{red}(t_{i+1})-t_i}\sum_{s\leq st} P^a_{\mathcal{ET}}(s|et(I')_{t_i})}^{\text{Probability of success}}$$
$$\overbrace{.\sum_{\Delta r|r\geq\Delta^{i+1}_r}\sum_{\delta^{i+1}_c|st+\delta^{i+1}_c\leq LET} P_r(\Delta^{i+1}_r).P_c(\delta^{i+1}_c)}^{\text{Probability of success}}.\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^{\Delta t,r_{t_j}}_{t_j})$$
$$-\overbrace{(P_{LR}+P_{TT}+P_{DM})}^{\text{Failure Probability}}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i} V([failure_{t_j},*,*])$$
$$-P_{PPC-fail}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i} V([failure_{t_j},*,*])$$
$$-\sum_{et'(t_{i+1})>et(t_{i+1})} P_{PPC-suc}(et'(t_{i+1}))\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}} P^{t_j}_{ra}(r_{t_j})V^{\Delta t,r_{t_j}}_{t_j}$$

As the transition system is complete:

$$\overbrace{P^r_{enough}(Pred(t_{i+1})).\prod_{a\in P_{red}(t_{i+1})-t_i}\sum_{s\leq st}P^a_{\mathcal{ET}}(s|et(I')_{t_i}).\sum_{\Delta r|r\geq\Delta^{i+1}_r}\sum_{\delta^{i+1}_c|st+\delta^{i+1}_c\leq LET}P_r(\Delta^{i+1}_r).P_c(\delta^{i+1}_c)}^{\text{Success probability}}$$
$$+P_{LR}+P_{TT}+P_{DM}+P_{PPC}=1$$

$EOC(t_{i+1},st)$ can therefore be simplified as:

$$\begin{aligned}EOC(t_{i+1},st)=&\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}}P^{t_j}_{ra}(r_{t_j})V^{0,r_{t_j}}_{t_j}\\&-\overbrace{P^r_{enough}(Pred(t_{i+1})).\prod_{a\in P_{red}(t_{i+1})-t_i}\sum_{s\leq st}P^a_{\mathcal{ET}}(s|et(I')_{t_i})}^{\text{Success probability}}\\&\overbrace{.\sum_{\Delta r|r\geq\Delta^{i+1}_r}\sum_{\delta^{i+1}_c|st+\delta^{i+1}_c\leq LET}P_r(\Delta^{i+1}_r).P_c(\delta^{i+1}_c)}^{\text{Success probability}}.\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}}P^{t_j}_{ra}(r_{t_j})V^{\Delta t,r_{t_j}}_{t_j})\\&-\overbrace{(P_{LR}+P_{TT}+P_{DM})}^{\text{Failure probability}}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}V([failure_{t_j},*,*])\\&-P_{PPC-fail}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}V([failure_{t_j},*,*])\\&-\sum_{et'(t_{i+1})>et(t_{i+1})}P_{PPC-suc}(et'(t_{i+1}))\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}}P^{t_j}_{ra}(r_{t_j})V^{\Delta t,r_{t_j}}_{t_j}\end{aligned}$$

Whatever st, probabilities on resource rates r_{t_j} remain unchanged. So, $\sum_{r_{t_j}}P^{t_j}_{ra}(r_{t_j})V^{0,r_{t_j}}_{t_j}$ remains unchanged whatever t_{i+1}'s start time.

If the expected opportunity cost is accurately and correctly computed, it can be re-written as the difference between a constant (C) and the expected utility of the other agents:

$$OC(t_{i+1},st)=C-\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}V'_{\mathcal{A}g_j}$$

where $V'_{\mathcal{A}g_j}$ is the expected utility of $\mathcal{A}g_j$.

Indeed, if the expected opportunity cost is accurately and correctly computed:

$$\begin{aligned}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}V'_{\mathcal{A}g_j}=&\overbrace{P^r_{enough}(Pred(t_{i+1})).\prod_{a\in P_{red}(t_{i+1})-t_i}\sum_{s\leq st}P^a_{\mathcal{ET}}(s|et(I')_{t_i})}^{\text{Probability of success}}\\&\overbrace{.\sum_{\Delta r|r\geq\Delta^{i+1}_r}\sum_{\delta^{i+1}_c|st+\delta^{i+1}_c\leq LET}P_r(\Delta^{i+1}_r).P_c(\delta^{i+1}_c)}^{\text{Probability of success}}.\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}}P^{t_j}_{ra}(r_{t_j})V^{\Delta t,r_{t_j}}_{t_j})\\&+\overbrace{(P_{LR}+P_{TT}+P_{DM})}^{\text{Failure probability}}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}V([failure_{t_j},*,*])\\&+P_{PPC-fail}\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}V([failure_{t_j},*,*])\\&+\sum_{et'(t_{i+1})>et(t_{i+1})}P_{PPC-suc}(et'(t_{i+1}))\sum_{\mathcal{A}g_j\in\mathcal{A}g,j\neq i}\sum_{r_{t_j}}P^{t_j}_{ra}(r_{t_j})V^{\Delta t,r_{t_j}}_{t_j}\end{aligned}$$

Equation 8 therefore consists in maximizing the following terms:

$$\pi_i(s_i) = arg\_max_{E(t_{i+1},st),st \geq et(I')_{t_i}}(\overbrace{V'_{\mathcal{A}g_i}}^{\text{Exepcted Utility of } \mathcal{A}g_i} - \overbrace{C}^{\text{Constant}} + \overbrace{\sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} V'_{\mathcal{A}g_j}}^{\text{Expected Utility of the other agents}})$$

The selected action from s_i therefore maximizes the joint expected utility. Indeed, if the expected opportunity cost is correctly and accurately estimated when an agent revises its policy, the revised policy maximizes the system's expected utility.□