Applications of DEC-MDPs in multi-robot systems

Aurélie Beynier, Abdel-Illah Mouaddib

Applications of DEC-MDPs in multi-robot systems

Aurélie Beynier
LIP6 - University Pierre and Marie Curie
104 avenue du Président Kennedy
75016 Paris, France
aurelie.beynier@lip6.fr

Abdel-Illah Mouaddib
GREYC - University of Caen
Bd. Maréchal Juin
14032 Caen Cedex, France
mouaddib@info.unicaen.fr

Abstract

Optimizing the operation of cooperative multi-robot systems that can cooperatively act in large and complex environments has become an important focal area of research. This issue is motivated by many applications involving a set of cooperative robots that have to decide in a decentralized way how to execute a large set of tasks in partially observable and uncertain environments. Such decision problems are encountered while developing exploration rovers, team of patrolling robots, rescue-robot colonies, mine-clearance robots, etc.

In this chapter, we introduce problematics related to the decentralized control of multi-robot systems. We first describe some applicative domains and review the main characteristics of the decision problems the robots must deal with. Then, we review some existing approaches to solve problems of multiagent decentralized control in stochastic environments. We present the Decentralized Markov Decision Processes and discuss their applicability to real-world multi-robot applications. Then, we introduce OC-DEC-MDPs and 2V-DEC-MDPs which have been developed to increase the applicability of DEC-MDPs.

Introduction

Recent robotic researches have demonstrated the feasibility of projects such as space exploration by mobile robots, mine clearance of risky area, search and rescue of civilians in urban disaster environments, etc. In order to increase the performance and abilities of these robots, researchers aim at developing multi-robot systems where the robots could interact. As explained by Estlin et al. [Estlin et al., 1999] about multi-rover exploration of Mars, such teams of robots will be able to collect more data by dividing tasks among the robots. More complex tasks that require several robots to cooperate, could also be executed. Moreover, abilities of the team could be improved by enabling each rover to have special skills. Finally, if one robot fails (robot breakdown or failure of task execution), another robot will be able to repair the damage to the first robot or will complete the unexecuted tasks. Teams of robots can also be used to increase the efficiency of rescue robots, patrolling robots or to develop constellations of satellites [Damiani et al., 2005]. All these applications share common characteristics: they are composed of a set of robots that must autonomously and cooperatively act in uncertain and partially observable environments. Thus, each robot must be able to decide on its own, how to act so as to maximize the global performance of the system. In order these robots to be able to optimize their behaviors, decision making approaches that take into account characteristics of real-world applications (large systems, constraints on task execution, uncertainty and partial observability) have then to be developed.

Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) have proved to be efficient tools for solving problems of single-agent control in stochastic environments [Puterman, 2005, Kaelbling et al., 1998, Zilberstein et al., 2002]. The application of MDPs has therefore been extended to multiagent settings. Thus, Decentralized Markov Decision Processes (DEC-MDPs) have been proposed [Bernstein et al., 2002]. They allow for modeling cooperative and distributed decision problems under uncertainty and partial observability. This chapter will describe how DEC-MDP approaches can contribute to solve multi-robot decision problems.

The chapter will be divided into three main parts. The first part will describe multi-robot real-world applications and we will introduce problematics related to the decentralized control of robot teams. The second part will introduce the DEC-MDP framework and the last part of the chapter will present existing DEC-MDP approaches that are concerned with solving multi-robot decision problems.

Decentralized control in multi-robot systems

This section introduces problematics related to the decentralized control of multi-robot systems. Optimizing the operation of cooperative multi-robot systems that can cooperatively act in large and complex environments has become an important focal area of research. This issue is motivated by many applications involving a set of cooperative robots that have to decide in a decentralized way how to execute a large set of tasks in partially observable and uncertain environments.

Mars exploration scenario

The first problem we consider consists in controlling task execution of a cooperative team of Mars exploration rovers. Once a day, the team receives, from a ground center, a set of tasks to execute (observations, measurements, moves) which is intended to increase science knowledge. As the amount of useful scientific data returned to the ground measures the success of the mission, rovers aim at maximizing science return. This performance measure can be represented by an expected value function. In order to optimize this function, several kinds of constraints must be respected while executing the tasks [Cardon et al., 2001, Bresina et al., 2002, Zilberstein et al., 2002]:

- **temporal constraints**: start times and end times of tasks have to respect temporal constraints. Since robots are solar-powered, most operations must be executed during the day. Moreover, because of illumination constraints, pictures must be taken at sunset or sunrise. On the other hand, some operations must be performed at night (atmospheric measurements).

- **preconditions**: some tasks have setup conditions that must hold before they can be performed. For instance, instruments must be turned on and calibrated in order an agent to perform measurements. If these preconditions do not hold, the agent will fail to perform its task. Preconditions lead to precedence constraints between the tasks and to dependencies between the agents. Let us consider that a robot must take a sample of the ground in order for another robot to analyse it: the second robot cannot start analysis before the first robot has finished to take the sample. The success of the second robot relies therefore on the first robot.

- **resource constraints**: executing a task requires power, storage (storing pictures or measurements) or bandwidth (data communication). These resources must be available to complete a task.

Moreover, robots must handle uncertainty on task execution and partial observability of the environment. Since accuracy and capacity of sensors are limited, each rover partially senses its environment. Because the environment is unknown and the issue of a task may depend on environment parameters (temperature, slope and roughness of the terrain, etc.), durations and resource consumptions of tasks are uncertain. Thus, each task takes differing amounts of time and consumes differing amounts of resources.

Furthermore, robots must deal with limited communications. Mars rovers communicate with operators via a satellite which is often unavailable due to its orbital rotation. Moreover, communications take time and consume resources. Consequently, communications with operators are limited to once a day. During this communication window, the robots send the data they have collected and they receive a new set of tasks to execute. During the rest of the day, the robots cannot communicate with the operators and they must act in an autonomous way. If there is no obstacle between the robots and they are close enough to each other, direct communication is possible. Nonetheless, as rovers cover large area with many obstacles, such direct communication is often impossible. They must therefore be able to perform their tasks without direct information exchange. Then, each rover must be able to autonomously decide how it will act and decision processes have to be decentralized. Finally, space robots have limited computation resources and data storage. Thus, in order to maximize collected data, each rover must be able to efficiently decide (with little computing power and data storage) which task to execute.

Rescue missions

There has been a growing interest in the recent years in disaster management crisis [Morimoto, 2000, RoboCup, 2000]. The RoboCup Rescue Competition has been organized since 2001 to promote research and development in this domain. The scenarios that are considered involve a team of rescue robots that must rescue civilians and prevent buildings from burning, after an earthquake occurs. A team is composed of three kinds of robots: fire brigade robots that must extinguish fires, ambulance robots that must rescue injured people and drive them to hospital, and police robots that can unblock roads. Such skills lead to dependencies between the robots. For instance, ambulance robots and fire robots cannot pass a blockade. Thus, police robots must unblock roads before the other robots can pass.

Rescue robots share many characteristics with planetary rovers. Rescue robots must face uncertainty and partial observability of the environment. Task durations and resource consumptions are uncertain. For instance, extinguishing a fire can take different amounts of time and consumes different amounts of water. Since communication installations often breakdown in such scenarios, it is assumed that communications between the agents are impossible. Moreover, resources are limited: an ambulance can load only one civilian at a time, a fire brigade robot has a limited amount of water, etc. Finally, temporal constraints must be considered. First, a crisis deadline is set. Next, temporal constraints can be deduced from scenarios' characteristics. For instance, a fire robots must have extinguished a fire before the building is entirely destroyed.

Multi-robot exploration and rescue rover missions are closely related to the problem of Decentralized Simultaneous Localization and Mapping (DSLAM) [Kleiner and Sun, 2007, Nettleton et al., 2003]. Decentralized SLAM addresses the problem of cooperatively building a map of an envrionment: a set of agents navigate in an unknown environment and jointly build a map of this environment while simultaneously localising themselves relatively to the map. DSLAM approaches have been applied to the problem of fire searching in an unknown environment [Marjovi et al., 2009].

Multi-robot flocking and platooning

The purpose of robot platooning [Michaud et al., 2006] is to build and to maintain a formation for a group of mobile robots from a starting point to a goal. Because the environments are unknown and the robots have imperfect sensing of the environment, environments are assumed to have unpredictable properties so actions have nondeterministic effects (for example, an agent can skid on a wet ground).Those kind of problems have been studied with flocking approach, where the agents have to maintain a global shape thanks to few simple local basic rules. Flocking rules [Reynolds, 1987] are a set of three very simple rules describing the behaviour of the agents. Those rules are :

1. Cohesion: steer to move toward the average position of local flockmates,
2. Separation: steer to avoid crowding local flockmates,
3. Alignment: steer towards the average heading of local flockmates.

Despite the simplicity of those rules, agents manage to maintain the shape of the group. The main advantage of this approach is that it is fully decentralized, with no communication at all.

The platooning can be seen as a particular form of flocking, where agents try to maintain a line shape and to move toward the platoon's objective (in this line, each agent has the same orientation as the previous agent if it is possible, and the leader heads to the objective. The global shape will then be a straight line or, if agents do not have enough space, a broken straight line). This can be done by giving particular flocking rules to each agent:

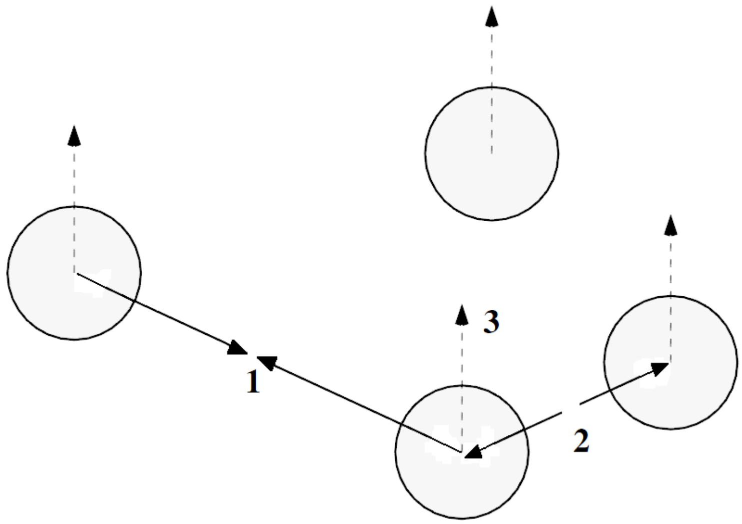

Figure 1: Flocking rules: (1) cohesion, (2) separation, (3) alignment

1. Cohesion: steer to wait for agents behind it,
2. Separation: steer to avoid agents in front of it,
3. Alignment: steer to move toward the near agent in front of it, or toward the objective if no one is in front of it.

The multi-robot teams presented in this section can be easily considered as cooperative multiagent systems. These consist of a set of agents that have to autonomously execute a set of tasks in the same environment so as to maximize a common performance measure[1]. The problems that are considered involve large sets of tasks and agents. For instance, regarding Mars exploration, the set of tasks to execute is sent once a day and may involve about ten robots that have to complete hundreds of tasks. Different kinds of constraints must be considered in order to achieve good performance. These include temporal constraints, precedence constraints, resource constraints, limited or impossible communication, limited computation capacities.

Decentralized Markov Decision Processes

The above mentioned multi-robot applications require a decentralized control approach that enables each robot to decide how to act in a partially observable environments and in a coordinated way with the other robots. Classical multiagent planning approaches are not suitable to handle such decision problems since they are not able to consider uncertainty and partial observability [Shoham and Tennenholtz, 1992, Weld, 1994a, Decker and Lesser, 1993a,

[1]The Robocup rescue competition defines a score based on the number of rescued civilians, the rate of burned buildings, etc.

Decker and Lesser, 1992, Clement and Barrett, 2003]. Some classical planning approaches, such as STRIPS, GRAPHPLAN or PGRAPHPLAN, have been adapted for planning under uncertainty [Blythe, 1999a, Blum and Furst, 1997, Blum and Langford, 1999]. 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 (see Chapter 3). 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 Markov Decision Processes (DEC-MDPs) [Bernstein et al., 2002] have been defined.

Model description

DEC-MDPs provide a mathematical framework to model and solve problems of decentralized control in stochastic environments. So as to modelize partial observability and uncertainty, the DEC-MDP 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 1 *A **Decentralized Markov Decision Process (DEC-MDP)** 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. The state of the system is assumed to be jointly observable [2].*
- *$\mathcal{A} = \langle \mathcal{A}_1, \cdots, \mathcal{A}_n \rangle$ 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'.*

[2] Decentralized Partially Observable Markov Decision Processes (DEC-POMDPs) generalize DEC-MDPs to formalize problems where the state of the system is partially observable [Bernstein et al., 2002].

- *$\mathcal{R}$ 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'.*

Problem solving

Optimally solving a DEC-MDP consists in finding a joint policy which maximizes the expected reward of the system.

Definition 2 *A joint policy π in an n-agent DEC-MDP is a set of individual policies $\langle \pi_1, \ldots, \pi_n \rangle$ where π_i is the individual policy of the agent $\mathcal{A}g_i$. The individual policy π_i of an agent $\mathcal{A}g_i$ is a mapping from each possible state of the agent's information (its state, its observations or its belief state) to an action $a_i \in \mathcal{A}_i$.*

Note that an individual policy π_i takes into account every possible information 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-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-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 to overcome the high complexity of DEC-MDPs. The first set of approaches aims at identifying properties of DEC-MDPs that reduce their complexity. Thus, Goldman and Zilberstein [Goldman and Zilberstein, 2004] have introduced transition independence and observation independence. These properties enable 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-MDPs have focused on finding approximate solutions instead of computing the optimum. [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 improvement of policies: 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 an 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-MDP solutions using one-step Bayesian games that are solved by a heuristic method. Alternatives to Hansen's exact dynamic programming algorithm [Hansen et al., 2004] have also been proposed by Bernstein et al.[Bernstein et al., 2005] and Amato et al. [Amato et al., 2007]. They use memory bounded controllers to limit the required amount of space to solve the problem. Recently, Wu et al. [Wu et al., 2010] have improved the scalability of Amato et al.'s approach [Amato et al., 2007] by avoiding the full backup performed at each step of the policy computation.

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 communicate to inform the other agents of their local state or observation. If communication is free and instantaneous and the system state is jointly observable, the problem is reduced to a Multiagent Markov Decision Process (MMDP) [Boutilier et al., 1999] that is easier to solve. Otherwise, the problem complexity remains unchanged and heuristic methods are described to find near optimal policies.

DEC-MDP approaches for multi-robot systems

Even if DEC-MDP approaches describe a powerful framework to formalize and solve multiagent decision problems, several issues arise while considering problems of decentralized control in real-world multi-robot systems. Bresina et al. [Bresina et al., 2002] point out some difficulties in formalizing robotic planning problems using markovian models. Indeed, this framework considers a simple model of time and actions. All actions are assumed to have the same duration (one time unit) so the agents are assumed to be fully synchronized. Moreover, DEC-MDPs do not take into account temporal and precedence constraints on action execution. The high complexity of optimally solving DEC-MDPs also reduced their applicability since it is difficult to solve problems involving more than two agents.

In this section we introduce two approaches based on DEC-MDPs that have been proposed to reduce the gap between the kinds of problems DEC-MDPs can solve and real world multi-robot applications. These models improve time and action representations and propose efficient approximate algorithms that can solve large problems considering constraints on task execution.

OC-DEC-MDP

The Opportunity Cost Decentralized Markov Decision Process (OC-DEC-MDP) framework [Beynier and Mouaddib, 2005, Beynier and Mouaddib, 2006] has been proposed to modelize and solve problems of decentralized control in multi-robot systems such as the ones presented at the beginning of the chapter. Because of communication limitations and unreliability of information exchange, communication between the agents (i.e. the robots) is assumed to be impossible during task execution. In order for a task to be successfully executed, temporal, precedence and resource constraints must be respected. Temporal constraints define, for each task t_i, a temporal window during which the task should be executed. Precedence constraints partially order the tasks by representing preconditions on task execution such as "task t_j must be finished before t_i can start". Finally, resource constraints guarantee that an agent has enough resources to execute a task.

Mission Definition

Problems of decentralized control in multi-agent systems that are considered in the OC-DEC-MDP framework, are defined as a **mission** $\mathcal{X}$ which stands for 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 the **set of tasks** to execute.

The problem is for the agents $\mathcal{A}g_i \in \mathcal{A}g$ to execute the set of tasks $\mathcal{T}$. The problem of task allocation is out of the scope of this chapter and tasks are supposed to be divided among the agents. Note that task allocation must take into account each agent's skills and must result in a feasible mission. Thus, there must be at least one interval of execution per task which respects temporal and precedence constraints. Hanna and Mouaddib [Hanna and Mouaddib, 2002], and more recently Abdallah and Lesser [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 et al. [Gerkey and Matarić, 2002] and Esben et al. [Esben et al., 2002] using auction principles.

As shown on Figure 2, a mission can be represented by an acyclic graph. This example describes a mission involving three planetary rovers. Edges stand for precedence constraints and nodes represent the tasks.

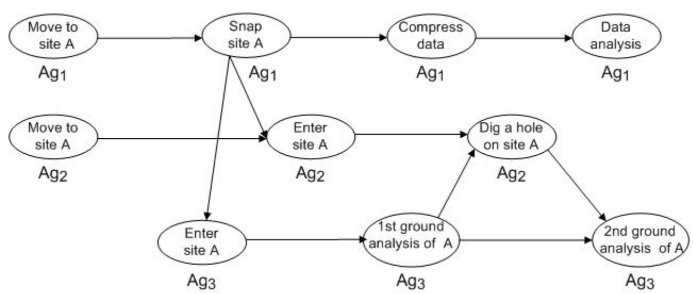

Figure 2: Mission graph

Each **task** $t_i \in \mathcal{T}$ is characterized by :

- an **agen**t $\mathcal{A}g_i$ that has to execute the task.
- **different possible durations** δ^i. $P_t(\delta^i)$ is the probability the execution of t_i takes δ^i time units.

- **different possible resource consumptions** Δ_r^i. $P_r(\Delta_r^i)$ is the probability the execution of t_i consumes Δ_r^i resources.

- **temporal constraints**: each task t_i is assigned a temporal window $TC_i = [EST_i, LET_i]$ during which it should be executed. EST_i is the Earliest Start Time of the task and LET_i is its Latest End Time.

- **precedence constraints**: each task t_i has a set of predecessors $Pred_i$ which defines the tasks to be executed 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 $root$ refers to the first tasks to be executed, i.e. the tasks without predecessors. Coordination constraints similar to our precedence constraints are described in frameworks such as TAEMS (Task Analysis, Environment Modeling and Simulation) [Decker and Lesser, 1993b] which is used to describe task structures of multiagent systems.

- a **reward** $\mathcal{R}_i$ which is the reward the agents obtain when t_i is successfully executed (respecting temporal, precedence and resource constraints).

Given a mission $\mathcal{X}$, the agents' aim consists in maximizing the sum of the cumulative reward they obtain during task execution. Because of the decentralized nature of the decision process and communication limitations, each agent must be able to decide, in a cooperative way, which task to execute and when, without communicating (during task execution) and with respect to constraints.

OC-DEC-MDP model

In order to model large problems, the OC-DEC-MDP model represents the multiagent decision problem as a set of MDPs where each MDP stands for a single agent decision problem. The policy of an agent $\mathcal{A}g_i$ will therefore be deduced from $\mathcal{A}g_i$'s MDP. Since each agent observe all the information it needs to make a local decision, MDPs are defined (not POMDPs) and the framework is referred as OC-DEC-MDPs.

Definition 3 *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 $\mathcal{A}g_i$,*
- *$\mathcal{R}_i : \mathcal{T}_i \to \mathbb{R}$ is the reward function of the agent $\mathcal{A}g_i$.*

Because of interactions between the agents, local MDPs are not independent of each others. Moreover, the components of the MDPs must be defined so as to represent constraints on task execution. The remaining of this section details each component of a local MDP.

States The state space of an agent (i.e. of its MDP) is composed of three kinds of states: success states, partial failure states and failure states.

• Success states: Let us consider an agent $\mathcal{A}g_i$ which has just successfully executed a task t_i during an interval I and let r be the agent's remaining resources. At the end of t_i's execution, the agent $\mathcal{A}g_i$ moves to a success state and must decide its next action. This action depends on the last successfully executed task t_i, its interval I and the remaining resources r. Thus, a success state of $\mathcal{A}g_i$ is defined as a triplet $[t_i, I, r]$.

• Partial failure states: When an agent $\mathcal{A}g_i$ starts to execute a task t_{i+1} at st but fails because the predecessors of t_{i+1} are not finished, it moves to a partial failure state $[t_i, [st, st+1], et(I'), r]$ where t_i stands for $\mathcal{A}g_i$ last successfully executed task, $et(I')$ is the end time of t_i and r is $\mathcal{A}g_i$'s remaining resources after it partially fails.

When an agent starts to execute a task t_{i+1} before the predecessors of t_{i+1} have finished their execution, the agent immediately realizes that the execution of the task partially fails. This means that the agent $\mathcal{A}g_i$, at $st+1$, realizes that it fails. As the agent could retry to execute the task later, this state is called a partial failure state. Thus, if precedence constraints are respected when the agent retries to execute t_{i+1}, the task could be successfully executed.

• Failure states: When an agent $\mathcal{A}g_i$ starts to execute a task t_{i+1} and it lacks resources or it violates temporal constraints, it moves to the failure state $[failure_{t_{i+1}}, *, *]$ associated to t_{i+1}.

Tasks - Actions At each decision step, the agent must decide when to start its next task. The actions to perform thus consist of *"Executing the next task t_{i+1} at time st: $E(t_{i+1}, st)$"*, that is the action to start executing task t_{i+1} at time st where st respects temporal constraints. Actions are probabilistic since the processing time and the resource consumption of the task are uncertain. Precedence and temporal constraints restrict the possible start times of each task. Consequently, there is a finite set of start times for each task and a finite action set.

Transition Function The transition function of an agent $\mathcal{A}g_i$ gives the probability that $\mathcal{A}g_i$ moves from a state s_i to a state s_i when it starts to execute a task t_{i+1} at st. Since the execution of t_{i+1} can lead to three different kinds of states, three kinds of transitions have to be considered: successful transitions, partial failure transitions and failure transitions. Transition probability computation differs for each kind of transition. Let us assume that an agent $\mathcal{A}g_i$ tries to execute a task t_{i+1} at st.

• Successful transitions: The probability that $\mathcal{A}g_i$ successfully executes t_{i+1} relies on: the probability the predecessors of t_{i+1} have finished at st (given by the probabilities on the end times of the predecessors), the probability $\mathcal{A}g_i$ has enough resources to execute the task (given by the probabilities on resource consumptions of t_{i+1}), and the probability $\mathcal{A}g_i$ finishes the execution of the task before its deadline (given by probabilities on the durations of t_{i+1}).

• Partial failure transitions: The probability that $\mathcal{A}g_i$ moves to a partial failure state is the probability that the predecessors of t_{i+1} have not finished at st and $\mathcal{A}g_i$ has enough resources to be aware of its partial failure. The probability that the predecessors have not finished at st is the probability that they will finish later or they will never finish.

• Failure transitions: An agent fails to execute its task if it lacks resources or temporal constraints are violated. The probability that $\mathcal{A}g_i$ lacks of resources is given by the probability the execution of t_{i+1} consumes more resources than available or the agent partially fails and the necessary resources to be aware of it are not sufficient.

If $st > LET_{i+1} - min(\delta^{i+1})$, the agent starts the execution of t_{i+1} before the latest end time of t_{i+1} ($LET_{i+1} - min(\delta^{i+1})$). Temporal constraints are therefore violated and the agent fails.

When $st \leq LET_{i+1} - min(\delta^{i+1})$, the agent may also violate temporal constraints. Indeed, if the duration δ^{i+1} is so long that the deadline is met ($st + \delta^{i+1} > LET_{i+1}$), the agent fails. The probability of violating the deadline therefore relies on duration probabilities.

In order to define transition functions, probabilities on start times and end times of the tasks must be known. The probability an agent starts to execute a task t_{i+1} at st relies on the agent's policy, on its available resources and on the ends times of the predecessors of t_{i+1}. Moreover, the predecessors' end times depend on the policies of their agents. Thus, the agents' policies have to be known to compute probabilities on start times and end times. Assuming an initial set of policies for the agents (one policy per agent), a propagation algorithms has been developed [Beynier and Mouaddib, 2005] to compute such probabilities. This algorithm propagates constraints through the mission graph from the roots to the leaves. Each time a node (i.e. a task) t_i is considered, its temporal probabilities (probabilities on start times and end times) and resource probabilities are computed using temporal and resource probabilities of the predecessors of t_i and using the policy of t_i. Once all the nodes have been considered, transition probabilities can be deduced from temporal probabilities and probabilities on resource consumptions.

Reward function When it successfully executes a task t_{i+1}, the agent $\mathcal{A}g_i$ moves to a success state and obtains the reward associated with t_{i+1}. If the agent partially fails, no reward is obtained. Finally, if the agent permanently fails the execution of t_{i+1}, it is penalized for all the tasks it will not be able to execute due to the failure of t_{i+1}.

Complexity Analysis A joint policy for the agents in an OC-DEC-MDP is a set of individual policies $\langle \pi_1 \cdots \pi_n \rangle$ where π_i is a local policy for an agent $\mathcal{A}g_i$ in the OC-DEC-MDP.

Theorem 1 *Optimally solving an OC-DEC-MDP requires an exponential amount of computation time.*

Proof: Optimally solving an OC-DEC-MDP consists in finding a joint policy that maximizes the global performance [Bernstein et al., 2002]. From the definition of a joint policy for an n-agent OC-DEC-MDP, we can deduce that the number of possible joint policies is exponential in the number of joint states. Evaluating a joint policy can be done in polynomial time through the use of dynamic programming [Goldman and Zilberstein, 2004].

In fact, we use standard policy evaluation algorithms for MDPs since the policy to evaluate is a mapping from joint states to joint actions: the states of the MDPs are the joint states $s = \langle s_1 \cdots s_n \rangle$ where s_i is a local state of $\mathcal{A}g_i$ in the OC-DEC-MDP and the actions of the MDP are the joint actions $a = \langle a_1, \cdots a_n \rangle$ where a_i is an action of $\mathcal{A}g_i$ in the OC-DEC-MDP. Finding an optimal policy for an n-agent OC-DEC-MDP consists in evaluating all the possible local policies and therefore requires an exponential amount of computation time.□

Constraints affect the policy space but have no effect on the worst case complexity. They reduce the state space and the action space. Thus, the policy space can be reduced. Nonetheless, the number of policies remains exponential. Consequently, dealing with constraints does not result in lower complexity.

Due to the high complexity of OC-DEC-MDPs, it is untractable to optimally solve large size of problems. It is thus better to turn towards an approximate planning approach that can solve large size of problems and computes a solution that is closed to the optimum. Indeed, developing an optimal algorithm would limit the size of problems that can be solved in practice and real-world multi-rover applications could not be considered.

Decision Problem

During task execution, each agent has a local view of the system and does not know the other agents' states nor actions. If the execution of a task t_i starts before its predecessors 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 "wasted" 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 and the agent fails permanently executing the task increases.

The problem is to find a local policy for each agent that maximizes the sum of the rewards of all the agents. Thus, the agents must trade off the probability of partially failing and consuming resources to no avail against the consequences of delaying the execution of a task. Indeed, to maximize the sum of the expected rewards, each agent must consider the consequences of a delay on itself and on its successors.

Opportunity Cost and Expected Value For purposes of coordinating the agents, the notion of Opportunity Cost has been introduced by Beynier and Mouaddib [Beynier and Mouaddib, 2005, Beynier and Mouaddib, 2006]. It is borrowed from economics and refers to hidden indirect costs associated with a decision. In the OC-DEC-MDP framework, Opportunity Cost measures the indirect effect of an agent's decision on the other agents. More specifically, the Opportunity Cost is the loss of expected value resulting from delaying the execution of the other agents' 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.

Consequently, the policy of an agent $\mathcal{A}g_i$ in a state s_i is computed using two equations. The first one is a standard Bellman equation that computes the expected utility of an agent $\mathcal{A}g_i$ and considers the tasks $\mathcal{A}g_i$ still has to execute:

$$V(s_i) = \overbrace{R(s_i)}^{\text{Immediate Gain}} + \overbrace{max_{E(t_{i+1},st_{i+1}),st_{i+1}\geq t}(V(E(t_{i+1},st_{i+1}),s_i))}^{\text{Expected Utility}} \quad (1)$$

where $s_i =< t_i, [st_i, et_i], r_{t_i} >$ (and $et_i = t$) or $s_i =< t_i, [t-1, t], et_i, r_{t_i} >$. If s_i is a success state ($s_i =< t_i, [st_i, et_i], r_{t_i} >$), the agent obtains a reward for successfully executing task t_i and $R(s_i) = \mathcal{R}(t_i)$. Otherwise, $R(s_i) = 0$.

$V(E(t_{i+1}, st_{i+1}), s_i))$ denotes the expected utility of the agent while executing $E(t_{i+1}, st_{i+1})$ from state s_i. Since the execution of the action can lead to different types of transitions, $V(E(t_{i+1}, st_{i+1}), s_i))$ is defined as:

$$\begin{aligned} V(E(t_{i+1},st_{i+1}),s_i) =& V_{suc}(E(t_{i+1},st_{i+1}),s_i) + V_{PCV}(E(t_{i+1},st_{i+1}),s_i) \\ &+ V_{fail}(E(t_{i+1},st_{i+1}),s_i) \end{aligned}$$

where $V_{suc}(E(t_{i+1}, st_{i+1}), s_i)$ is the expected value of the agent when t_{i+1} is successfully executed at st_{i+1}, V_{PCV} is the expected value of the agent when the execution of t_{i+1} starts at st_{i+1} and partially fails, and V_{fail} is the expected value if the agent start executing t_{i+1} at st_{i+1} and it lacks resources or temporal constraints are violated.

The second equation computes the best foregone action using a modified Bellman equation in which an Expected Opportunity Cost (EOC) is introduced. It allows the agent to select the best action to execute in a state s_i, considering its expected utility and the EOC induced on the other agents:

$$\pi_i(s_i) = argmax_{E(t_{i+1},st_{i+1}),st_{i+1}\geq et_i}\Big(\overbrace{V(E(t_{i+1},st_{i+1}),s_i)}^{\text{Expected Utility}} - \overbrace{EOC(t_{i+1},st_{i+1})}^{\text{Expected Opportunity Cost}}\Big) \quad (2)$$

where:

- $argmax$ denotes the operator which returns the action $E(t_{i+1}, st_{i+1})$ which maximizes the trade-off between the expected utility V of the agent and the expected opportunity cost provoked on the other agents.
- $EOC(t_{i+1}, st_{i+1})$ is the expected opportunity cost the execution of t_{i+1} will induce if it starts at st_{i+1}.

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

- The expected value, computed using a standard Bellman equation (Equation 1). It takes into account the expected value of executing the agent's remaining task.
- The expected opportunity cost provoked on the other agents.

The EOC induced on the other agents when t_{i+1} starts at st is defined as follows:

$$EOC(t_{i+1}, st) = P_{suc} \cdot \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} EOC_{\mathcal{A}g_j, t_{i+1}}(et_{i+1}) \tag{3}$$

$$+P_{fail} \sum_{\mathcal{A}g_j \in \mathcal{A}g, j \neq i} EOC_{\mathcal{A}g_j, t_{i+1}}(fail) + P_{PCV} \cdot EOC(t_{i+1}, st')$$

where et_{i+1} is a possible end time of t_{i+1}, $EOC_{\mathcal{A}g_j, t_{i+1}}(et_{i+1})$ is the EOC induced on the agent $\mathcal{A}g_j$ when t_{i+1} ends at et_{i+1}. It is computed using Equation 4. $EOC(t_{i+1}, st')$ is the OC when the execution of t_{i+1} partially fails and the agents re-tries to execute the task at st' (the next start time of the task). P_{suc} stands for the probability to successfully execute the task, P_{PCV} is the probability to fail partially because the predecessors have not finished. P_{fail} is the probability to fail permanently.

EOC values can be deduced by considering the delay provoked on the successors. The Expected Opportunity Cost described in Equation 3 is given by :

$$EOC_{\mathcal{A}g_j, t_{i+1}}(et_{t_{i+1}}) = \sum_{r_{t_j}} P_{ra}^{t_j}(r_{t_j}) \cdot OC_{t_j}(\Delta t, r_{t_j}) \tag{4}$$

where t_j is the nearest task that will be executed by $\mathcal{A}g_j$ (the distance between two tasks t_i and t_j is given by the number of nodes that belongs to the shortest path between t_i and t_j in the mission graph). $P_{ra}^{t_j}(r_{t_j})$ is the probability that $\mathcal{A}g_j$ has r_{t_j} resources when it starts to execute t_j. Δt is the delay induced on t_j when t_{i+1} ends at $et_{t_{i+1}}$. This delay is computed by propagating temporal constraints between t_{i+1} and t_j. $OC_{t_j}(\Delta t, r_{t_j})$ is the Opportunity Cost provoked on t_j when it is delayed by Δt. It stands for a difference in expected value computed as follows:

$$OC_{t_j}(\Delta t, r_{t_j}) = V_{t_j}^{0, r_{t_j}} - V_{t_j}^{\Delta t, r_{t_j}} \tag{5}$$

where $V_{t_j}^{0, r_{t_j}}$ is the expected value of $\mathcal{A}g_j$ if the execution of t_j is not delayed and the agent has r_{t_j} resources when it starts to execute t_j. $V_{t_j}^{\Delta t, r_{t_j}}$ is the expected value of the agent $\mathcal{A}g_j$ when the execution of t_j is delayed by Δt and the agent has r_{t_j} resources when it starts to execute t_j.

If the execution of t_{i+1} fails, t_j could not be executed because of violation of precedence constraints. Then, $EOC_{\mathcal{A}g_j, t_{i+1}}(fail)$ is given by:

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

Policy computation Given a state s_i of an agent $\mathcal{A}g_i$, Equation 2 allows for the agent to decide its policy from s_i. Beynier and Mouaddib [Beynier and Mouaddib, 2006] have proposed an iterative revision algorithm which applies this decision method to each state of each agent and computes an approximate solution to the multiagent decision problem. The algorithm consists in iteratively improving an initial policy set. At each iteration step, the agents improve their initial local policy at the same time. Given the initial policies that have been used to compute temporal and resource probabilities, each agent tries to improve its own policy. At each iteration step, each agent $\mathcal{A}g_i$ traverses the task graph in the reverse topological order and, revises the execution policy of each task (node), using Equation 2. While revising the policy of t_i, $\mathcal{A}g_i$ considers all the states s_i from which t_i can be executed (states associated to the previous task t_{i-1} of $\mathcal{A}g_i$). The expected value of s_i is then computed and its policy is deduced. This process is repeated until no changes are made. An equilibrium is then reached.

Experiments

Experiments have been developed to prove the scalability, the efficency and the applicability of OC-DEC-MDPs. Experiments show that large problems can be solved using the OC-DEC-MDP framework. Indeed, missions of hundreds of tasks and more than twenty agents can be considered. The performances obtained at each iteration step have also been studied by running mission executions. Experiments illustrate that the performance of the agents increases with the number of iterations. By iterating the process, the likelihood the agents fail because of partial failure resource consumption and because of lack of resources, decreases. The resulting policy is safer than policies of previous iterations and the gain of the agents is steady over executions. A near optimal policy is obtained at the end of the first iteration. Second iteration leads to small improvements but it diminishes the number of partial failures.

Finally, the OC-DEC-MDP framework has been applied to real-world scenarios using Koala robots. Scenarios derived from Mars rover missions were considered. Figure 3 represents a scenario involving two robots that have to explore a set of 8 interesting places. The first robot (robot $\mathcal{A}g_1$) can take pictures and the second one (robot $\mathcal{A}g_2$) can take and analyse ground samples. Robot $\mathcal{A}g_1$ must take picture of sites A, B, D, E, F, H and J, and robot $\mathcal{A}g_2$ must analyse sites C, D, F, H and I. Sites are ordered so as to minimize travelling resource consumptions. Furthermore, precedence constraints have to be taken into account. As taking samples of the ground may change the topology of the site, pictures of a site must be taken before the other robot starts to analyse it. Moreover, robot $\mathcal{A}g_1$ must have left a site before robot $\mathcal{A}g_2$ can start to analyse it. Thus, robot $\mathcal{A}g_1$ must have taken a picture of site D before robot $\mathcal{A}g_1$ enters this site. Temporal constraints have also to be considered: visiting earliest start times and latest end times are associated with each site.

The mission was represented using a mission graph (Figure 4). Then, the corresponding OC-DEC-MDP was automatically built and solved by the iterative algorithm. Finally, resulting policies were implemented on Koala robots. During task execution, robots only have to execute their policies which map each state to an action. Thus, initial ambitions about the limitation of computational resources needed to make a decision have been fulfilled. Coordination performs well even if robots cannot communicate. Temporal and precedence constraints are respected. As shown on Figure 5 for the crossing point D, while deciding when to start its action, the first robot takes into account the fact that the other robot waits for him (thanks to the OC). The decision of the second robot is based on the probability that robot $\mathcal{A}g_1$ has left the site, the cost of a partial failure, and the robot's own expected value. Thus, robot 1 enters site D, completes its task (Picture 2) and leaves the site (Picture 3). Then, robot $\mathcal{A}g_2$

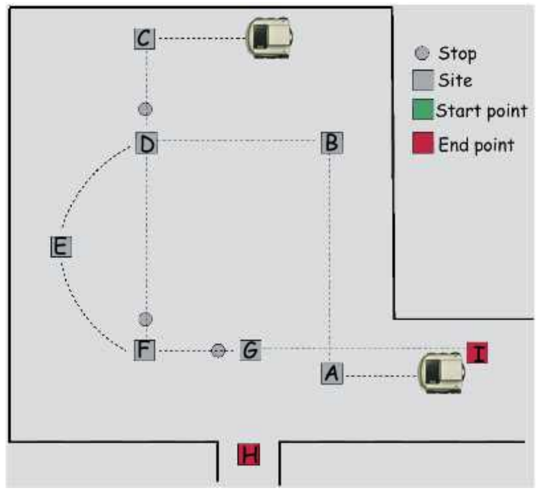

Figure 3: Two-robot exploration scenario

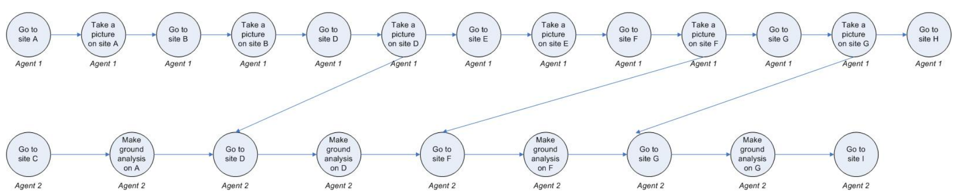

Figure 4: Mission graph of the two-robot exploration scenario

tries to enter the site (Picture 4). As robot 1 does not know the other robot actions, it may try to enter the site and fails because the other robot has not finished to take the picture. The second robot realizes that it fails when it tries to enter the site. If precedence constraints are not respected the robot returns to its last position. If temporal constraints are respected, the robot enters the site (Picture 4). These experiments show that the OC-DEC-MDP approach can be used by physical robots which are thus able to successfully and cooperatively complete their mission.

2V-DEC-MDP for flocking and platooning

In [Mouaddib et al., 2007], the Vector-Valued Decentralized Markov Decision Process (2V-DEC-MDP) framework has been proposed to coordinate locally the actions of a group of agents. It is based on MDP with an online coordination part. Assuming without loss of generality that all agents are identical, a 2V-DEC-MDP is a set of 2V-MDP, one per agent. A 2V-MDP is composed by an off-line part, an MDP, and an on-line part to adapt its actions with the other agents.

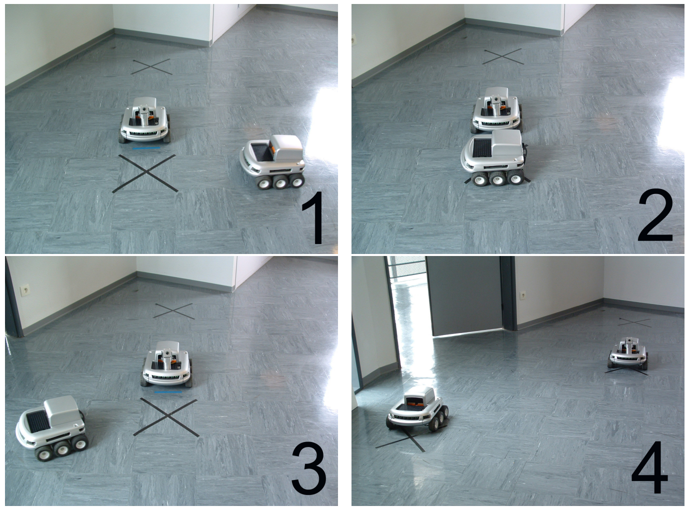

Figure 5: Execution of the mission (crossing point of site D)

The MDP is a tuple $\langle S, A, P, R\rangle$, with:

- S a set of states,
- A a set of action,
- $P : S \times A \times S \to [0; 1]$, the transition function,
- $R : S \times A \times S \to \mathbb{R}$, the reward function which expresses both positive reward for goal states and negative reward for hazardous states.

For the optimality criteria, an expected reward is defined on a finite horizon T. The optimal value function V^* of a state is defined by:

$$V^*(s) = \max_{a\in A}(R(s, a) + \sum_{s'\in S} P(s, a, s') \cdot V^*(s')), \forall s \in S$$

A policy is a function $\pi : S \to A$, the optimal policy is a policy π^*, such that:

$$\pi^*(s) = argmax_a(R(s, a) + \sum_{s'\in S} P(s, a, s') \cdot V^*(s')), \forall s \in S$$

The neighborhood for an agent i is defined as the set of states of (detected) agents who can interact with i. Until now, it is assumed that all the agents near enough (according to a fixed maximum distance d) could be detected and their states could be known. Taking into account partial observability will be the subject of some future works. If the neighborhood is too big, it can be restricted to a subset (more the neighborhood will be big and more the policy will be good but more the computation of this policy will take time).

The on-line part of a 2V-MDP is built with the computation of local social impact, according to local observations. The functions for computing the value of the impact on the group are:

- ER for the individual reward (the value of the optimal policy of the MDP),
- JER for the group interest,
- JEP for the negative impact on the group.

Using those functions, the agents will use a $LexDiff$ operator to choose the policy (i.e. the best action) to apply.

$LexDiff$ builds a vector $v = (ER(\pi_i), JER(\pi_i), JEP(\pi_i))$ for every π_i and normalize each values vector $v_i = (v_i^1, v_i^2, v_i^3)$ to a utilities vector $v_u = (v_u^1, v_u^2, v_u^3)$. $LexDiff$ then permutes those utilities vectors so that each vector (v^1, v^2, v^3) be such that $v^1 \geq v^2 \geq v^3$. The best vector is then founded by a lexicographic order: for two vectors $v_a = (v_a^1, v_a^2, v_a^3)$ and $v_b = (v_b^1, v_b^2, v_b^3)$, we choose v_a if $v_a^1 > v_b^1$ and v_b if $v_a^1 < v_b^1$. If $v_a^1 = v_b^1$, we compare v_a^2 and v_b^2, and so on.

Thanks to this design, the DEC-MDP is expressed as a set of 2V-MDP, allowing the coordination problem to be tractable. In [Boussard et al., 2008], ER JER and JEP have been defined for platoon emergence, but this work does not try to keep the shape of the platoon.

2V-DEC-MDP-Based approach for flocking

2V-DEC-MDP has been used to formalize the problem, by translating the three criteria into three formulae (each formula having one or more equations) which will parameterize each 2V-MDP. Three functions have been defined: ER as the alignment criterion, JER as the cohesion criterion and JEP as the separation criterion.

Notations

- s_i^j is the state j of agent i (the environment being reduced to a discrete set of possible positions, a state is one position of this set and one orientation),
- $\overrightarrow{s} = (s_1, \ldots, s_N)$ is the joint state vector,
- $face(s)$ gives all the agents the are closer to the objective than s,
- $distance(s^1, s^2)$ gives the number of actions needed to go from s^1 to s^2,
- $angle(s^1, s^2)$ gives the angle between the orientation of s^1 and the one of s^2:

$$angle(s^1, s^2) = \frac{\|orientation_{s^1} - orientation_{s^2}\|}{angle_{max}}$$

- $back(s)$ gives the next place available behind s (if s^1, the location just behind s according to the orientation of s, is available, s^1 is returned. If it is not available, $back(s^1)$) is returned.

So now, using those definitions, the formulae for ER, JER and JEP can be written in the platooning context:

Alignment

$$ER(s,a) = \sum_{s' \in S} p(s,a,s') ER_i, \quad i = 1,2,3$$

Depending on the situation, ER_i are defined by:

$$\begin{aligned}
ER_1 &= V^*(s') \\
ER_2 &= -\min_{s_j \in face(s')} (distance(s', s_{b1}) + \frac{angle(s', s_{b1})}{angle_{max}}) \\
ER_3 &= -(distance(s', s_{b2}) + \frac{angle(s', s_{b2})}{angle_{max}})
\end{aligned}$$

where $s_{b1} = back(s_j)$, $s_{b2} = back(leader)$ and $V^*(s)$ a function of the expected distance between s and the objective of the platoon. $distance(s^1, s^2)$ gives the cost of going from a state s^1 to a state s^2 and $angle(s^1, s^2)$ gives the cost of rotating from the orientation of s^1 to the one of s^2. Thus, it has been added to those equations two costs: the cost of going from a state s^1 to a state s^2, wich means the cost of reaching the position of s^2 AND rotating to the good orientation. The angle is divided by the maximum angle to be sure that the cost of the distance will always be bigger than the cost of the angle, so the agent will not choose to stay on a distant place for saving the cost of a rotation. In ER_2 and ER_3, $back(target)$ is used instead of $target$, because the agent wants to go behind its target.

An agent does not have the same objectives whether it is on a leader position or inside a platoon. Indeed, a leader will move in the direction of its objective, while a non-leader agent will follow the one in front of it. Hence, an agent have to choose which equation to follow before resolving its 2V-MDP.

So, if the agent is a leader, or if it is out of range of any platoon, it chooses ER_1. If it is inside a platoon but it knows that the leader is behind it, it chooses ER_3. Finally, if it is inside a platoon and have no leader behind it, it chooses ER_2.

Separation

$$JEP(s,a) = \sum_{s' \in S} [p(s,a,s') \cdot \sum_{s_j \in D} (\sum_{a_j^k, k=1}^{|A_j|} p(s_j, a_j^k, s') \cdot C)]$$

Where D is the set of states of detected agents in neighborhood and C a constant equal to the cost of a collision between two agents.

Cohesion

$$JER(s,a) = \sum_{s' \in S} (p(s,a,s') \cdot K(s'))$$

Where $K(s)$ is the function which estimate the gain of a given situation for the group. $K(s)$ gives a reward if at least one agent is behind s.

After choosing an equation for the ER criteria, the agent has to fix the weight of ER, JER and JEP. For a leader, it is set w_{JEP} to 0 since the criterion is with no sense for it and, typically, w_{ER} to 0.49 and w_{JER} to 0.51. For a non-leader, $w_{JEP} = 0.35$, $w_{ER} = 0.32$ and $w_{JER} = 0.33$ (except if a leader is detected behind the agent, in which case $w_{JER} = w_{JEP} = 0$, and $w_{ER} = 1$). Finally, for any agent, $w_{JER} = 0$ as soon as it is near to the objective of the platoon. Experimentations proved that values of those weights do not change anything on the behavior of the agents. The only important thing is the order of those weights: the most important criteria has to have the biggest weight, the second criteria has to have the second weight, etc., so values for those weights could be chosen arbitrary.

Experiments with real robots

After testing the approach on a simulator, tests on real robots (Koalas) have been developped. Those robots know the "map" of the environment they are evolving in and have local visibility, so they know the position and orientation of the agents around them. A 2V-DEC-MDP, parameterized as described before, is running on them. An example with 3 robots is shown on Figure 6. Robots are placed on a same line, and an objective in front of them is given (the door on the right side). Figure 7, Figure 8 and Figure 9 are captions of those tests.

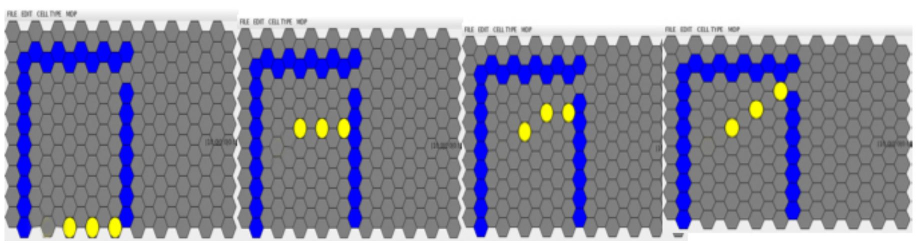

Figure 6: Scenario of multi-robot platooning

Figure 7: Initial situation

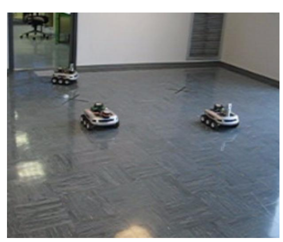

Figure 8: After few moves

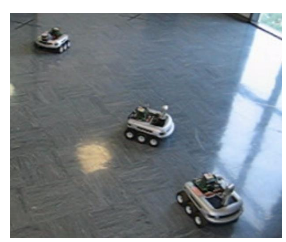

Figure 9: Platoon is formed

When the test starts, the closest robot to the objective chooses the $ER1$ function and goes toward its objective. Because of the JER function, it waits for the other agents. In the same time, the two other agents follow the first one: according to the $ER2$

function, one of them chooses to follow the first agent, while the other one chooses to take the third place.

The platoon then emerges from those interactions: we can see the robots in their initial position in Figure 7, and their position after a few moves in Figure 8. Then, in Figure 9, we can see the fully shaped platoon.

Many other initial configurations were considered and we can see that, for each configuration, robots fully form a platoon after some moves.

Conclusion

Decentralized decision making is an appropriate approach for multi-robot applications since they are able to support uncertainty, partial observability and decentralized control. Even if Decentralized Markov Decision Processes suffer from a high complexity, the structure of multi-robot decision problems such as constraints on task execution (exploration mission) or locality of interactions (platooning) can be exploited to reduce the complexity. This chapter presented two approaches based on DEC-MDPs that have been proved to solve efficiently multi-robot cooperative problems. These approaches allow us to derive individual cooperative policies for the robots such that a global utility is maximized. The coordination in those approaches is considered during the computation of the policies by evaluating the effect of a local decision on the other robots. In the opposite to that, classical multiagent planning techniques address the problem of coordination in two steps: computing plans and then coordinating them. The second step requires in general a costly communication between the robots that limits their applicability in real-world applications (communication not always available, costly and time consuming). Another drawback of classical approaches is when the execution deviates from the expected behavior and thus re-planification and re-coordination are required that can reduce the performance of the system during the execution. Another contribution of decentralized decision models is to better formalize the flocking techniques by improving their robustness, supporting the uncertainty and assessing the quality of the global behavior.

Markov Decision Processes have also been successfully used to solve decentralized decision problems in non Artificial Intelligence domains. For instance, decision problems of search and storage in peer-to-peer server networks have been solved using a set of Interactive Markov Decision Processes [Beynier and Mouaddib, 2009].

Future works in multi-robot domain should concern the extension of the DEC-MDPs to deal with problems involving human and robot interactions such as mixed initiative techniques [Weld, 1994b, Sidner and Lee, 2005, Freedy et al., 2008]. These systems can operate mostly autonomously, but may need supervision or help in particular situations. Examples include mobile robots or intelligent vehicles navigating in a narrow corridor or heavy traffic, or avoiding risky areas that could cause costly failures. Similarly, robots performing complex surgical operations may require supervision and intervention of the specialist. In these applications, a supervision unit, often a human operator, can take over control when the situation is too complex for the autonomous system [Crandall and Goddrich, 2005]. While the supervision unit (e.g., a driver, a surgeon, or a control center operator) may be able to perform each task by manually controlling the system, this would normally result in a time-consuming, costly operation. The problem is therefore to develop a general framework for supervision unit - autonomous unit teaming, to optimize performance and reduce the supervision unit work-load, costs, fatigue-driven errors and risks [Green et al., 2008].

References

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

[Amato et al., 2007] Amato, C., D.S., B., and Zilberstein, S. (2007). Optimizing memory-bounded controllers for decentralized pomdps. In *Proceedings of the Twenty Third Conference on Uncertainty in Artificial Intelligence*.

[Becker et al., 2003] Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. (2003). Transition-independent decentralized markov decision processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 41–48, Melbourne, Australia.

[Bernstein et al., 2005] Bernstein, D., Hansen, E.A., and Zilberstein, S. (2005). Bounded policy iteration for decentralized pomdps. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland.

[Bernstein et al., 2002] Bernstein, D., Zilberstein, S., and Immerman, N. (2002). The complexity of decentralized control of mdps. In *Mathematics of Operations Research*, pages 27(4):819–840.

[Bernstein et al., 2001] Bernstein, D., Zilberstein, S., Washington, R., and 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.

[Beynier and Mouaddib, 2009] Beynier, A. and Mouaddib, A. (2009). Decentralized decision making process for document server networks. In *Proceeding of the International Conference on Game Theory and Networks*.

[Beynier and Mouaddib, 2005] Beynier, A. and Mouaddib, A.I. (2005). A polynomial algorithm for decentralized markov decision processes with temporal constraints. *Proceedings of the fourth Interantional Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 963–969.

[Beynier and Mouaddib, 2006] Beynier, A. and Mouaddib, A.I. (2006). An iterative algorithm for solving constrained decentralized markov decision processes. In *The Twenty-First National Conference on Artificial Intelligence (AAAI-06)*.

[Blum and Furst, 1997] Blum, A. and Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300.

[Blum and Langford, 1999] Blum, A. and Langford, J. (1999). Probabilistic planning in the graphplan framework. In *Proceedings of ECP'99*.

[Blythe, 1999a] Blythe, J. (1999a). Decision-theoretic planning. *AI Magazine*.

[Blythe, 1999b] Blythe, J. (1999b). *Planning under uncertainty in Dynamic domains*. Phd thesis, Carnegie Mellon University.

[Boussard et al., 2008] Boussard, M., Bouzid, M., and Mouaddib, A. (2008). Vector valued markov decision process for robot platooning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI 2008)*.

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

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

[Cardon et al., 2001] Cardon, S., Mouaddib, A., Zilberstein, S., and Washington, R. (2001). Adaptive control of acyclic progressive processing task structures. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI-2001*, pages 701–706.

[Chadès et al., 2002] Chadès, I., Scherrer, B., and 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*.

[Clement and Barrett, 2003] Clement, B. and Barrett, A. (2003). Continual coordination through shared activities. In *International Joint Conference on Autonomius Agents and MulAgent Systems, AAMAS*.

[Crandall and Goddrich, 2005] Crandall, P. and Goddrich, M. (2005). Validating human-robot interaction schemes in multitasking environments. *IEEE transaction Systems, Man, and Cybernetics, Part A : Systems and Humans*, 35(4):438–449.

[Damiani et al., 2005] Damiani, S., Verfaillie, G., and Charmeau, M. (2005). An earth watching satellite constellation: how to manage a team of watching agents with limited communications. In *Proceedings of the fourth International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2005)*, pages 455–462.

[Decker and Lesser, 1992] Decker, K. and Lesser, V. (1992). Generalizing the partial global planning algorithm. *Journal on Intelligent Cooperative Information Systems*, 1(2):319–346.

[Decker and Lesser, 1993a] Decker, K. and Lesser, V. (1993a). Quanititative modeling of complex computational task environnement. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224.

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

[Emery-Montemerlo et al., 2004] Emery-Montemerlo, R., Gordon, G., Schneider, J., and 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 et al., 2002] Esben, H. O., Maja, J. M., and Gaurav, S. S. (2002). Multi-robot task allocation in the light of uncertainty. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3002–3007.

[Estlin et al., 1999] Estlin, T., Tobias, A., Rabideau, G., Castana, R., Chien, S., and Mjolsness, E. (1999). An integrated system for multi-rover scientific exploration. In *The Sixteenth National Conference on Artificial Intelligence*.

[Freedy et al., 2008] Freedy, A. ., Sert, O. ., Freedy, E. ., Weltman, G. ., Mcdonough, J. ., Tambe, M., and Gupta, T. (2008). Multiagent adjustable autonomy framework (maaf) for multirobot, multihuman teams. In *International symposium on collaborative technologies (CTS) 2008*.

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

[Goldman and Zilberstein, 2003] Goldman, C. and Zilberstein, S. (2003). Optimizing information exchange in cooperative multiagent systems. In *International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 137–144.

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

[Green et al., 2008] Green, A. S., Billinghurst, M., Chen, W., and Chase, J. (2008). Human-robot collaboration: A literature review and augmented reality approach in design. *International Journal of Advanced Robotic Systems*, 5(1):1–16.

[Hanna and Mouaddib, 2002] Hanna, H. and Mouaddib, A. (2002). Task selection as decision making in multiagent system. In *International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 616–623.

[Hansen et al., 2004] Hansen, E.A., Bernstein, D., and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*.

[Howard, 1960] Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press.

[Kaelbling et al., 1998] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134.

[Kleiner and Sun, 2007] Kleiner, A. and Sun, D. (2007). Decentralized slam for pedestrians without direct communication. In *Proceedings of IROS 2007*, pages 1461–1466.

[Marjovi et al., 2009] Marjovi, A., Nunes, J., Marques, L., and de Almeida, A. (2009). Multi-robot fire searching in unknown environment. In *Intelligent Robots and Systems (IROS 2009)*, pages 1929–1934.

[Michaud et al., 2006] Michaud, F., Lepage, P., Frenette, P., Letourneau, D., and Gaubert, N. (2006). Coordinated maneuvering of automated vehicles in platoons. *ITS*, 7(4):437–447.

[Morimoto, 2000] Morimoto, T. (2000). *How to develop a RoboCupRescue agent*. RoboCupRescue Technical Committee.

[Mouaddib et al., 2007] Mouaddib, A., Boussard, M., and Bouzid, M. (2007). Towards multiobjective multiagent planning. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*.

[Nair et al., 2005] Nair, R., Pradeep, V., Milind, T., and 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 et al., 2003] Nair, R., Tambe, M., Yokoo, M., Marsella, S., and Pynadath, D.V. (2003). Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 705–711.

[Nettleton et al., 2003] Nettleton, E., Thru, S., Durrant-Whyte, H., and Sukkarieh, S. (2003). Decentralized slam with low-bandwith communication for teams of vehicles. In *Proceedings of the International Conference on Field and Service Robotics*.

[Peshkin et al., 2000] Peshkin, L., Kim, K., Meuleu, N., and Kaelbling, L. (2000). Learning to cooperate via policy search. In *Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 307–314.

[Puterman, 2005] Puterman, M. L. (2005). *Markov Decision processes : discrete stochastic dynamic programming*. Wiley-Interscience, New York.

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

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34.

[RoboCup, 2000] RoboCup (2000). *RoboCup-Rescue simulator manual*.

[Roy et al., 2000] Roy, N., Pineau, J., and 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.

[Shoham and Tennenholtz, 1992] Shoham, Y. and Tennenholtz, M. (1992). On social laws for artifical agent sociaties: off-line design. *Articial Intelligence*, 73(1-2):231–252.

[Sidner and Lee, 2005] Sidner, C. and Lee, C. (2005). Robots as laboratory hosts. *Interactions*, 12(2):24–26.

[Varakantham et al., 2007] Varakantham, P., Marecki, J., Yabu, y., Milind, T., and 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)*.

[Weld, 1994a] Weld, D. (1994a). An introduction to least-commitment planning. *Articial Intelligence Magazine*, 15(4):27–61.

[Weld, 1994b] Weld, D. (1994b). Robots that work in collaboration with people. *AAAI Fall Symposium on the intersection of Robotics and Cognitive sciences*.

[Wu et al., 2010] Wu, F., Zilberstein, S., and Chen, X. (2010). Point-based policy generation for decentralized pomdps. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

[Xuan et al., 2001] Xuan, P., Lesser, V., and Zilberstein, S. (2001). Communication decisions in multiagent cooperation : Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 616–623, Montreal. ACM Press.

[Zilberstein et al., 2002] Zilberstein, S., Washington, R., Bernstein, D., and Mouaddib, A.I. (2002). Decision-theoretic control of planteray rovers. *M. Beetz et al. (Eds.), Plan-Based control of Robotic Agents*, pages 2466:270–289.