COLLECTIVE DECISION UNDER PARTIAL OBSERVABILITY
A Dynamic Local Interaction Model

Arnaud Canu and Abdel-Illah Mouaddib
Université de Caen Basse-Normandie, UMR 6072 GREYC, F-14032 Caen, France
CNRS, UMR 6072 GREYC, F-14032 Caen, France

Keywords: Markovian decision process, Game theory and applications, Multiagent decision making, Co-operation.

Abstract: This paper introduces DyLIM[a], a new model to describe partially observable multiagent decision making problems under uncertainty. DyLIM deals with local interactions amongst the agents, and build the collective behavior from individual ones. Usually, such problems are described using collaborative stochastic games, but this model makes the strong assumption that agents are interacting all the time with all the other agents. With DyLIM, we relax this assumption to be more appropriate to real-life applications, by considering that agents interact sometimes with some agents. We are then able to describe the multiagent problem as a set of individual problems (sometimes interdependent), which allow us to break the combinatorial complexity. We introduce two solving algorithms for this model and we evaluate them on a set of dedicated benchmarks. Then, we show how our approach derive near optimal policies, for problems involving a large number of agents.

[a]This work is supported by the DGA (Direction Générale de l'Armement, France).

1 INTRODUCTION

Decision making under uncertainty is an important aspect of Artificial Intelligence. Its extension to multiagent settings is even more important to deal with real-world applications such as multirobot systems or sensor networks for example. Stochastic games are useful to describe such problems, especially the specific case of cooperative agents, represented by DEC-POMDPs. However, this model is very hard to solve because of the significant number of different situations each agent can face. Furthermore, this model is based on the strong assumption that each agent is interacting with all the other agents, at anytime. For this reason, its a NEXP-Complete problem[1].

Recently, other models were introduced, relaxing this assumption. They are based on the idea that an agent only interacts sometimes, with some other agents (local interactions). However, they all suffer from limitations in terms of applicability. Some approaches use a *static interaction model*, meaning that the agent is always interacting with the same agents. Other approaches use a *dynamic interaction model* but limit the possible interactions (using coordination locales with task allocation, or needing free communication and full local observability).

Our work is motivated by the practical problem of a group of autonomous vehicles evolving in an environment where they cannot communicate. The agents are able to observe their neighbors, so they take decisions based on local interactions: the existing models are unable to formalize such problems. In this paper, we introduce a new model (Dylim, the Dynamic Local Interaction Model) to describe problems involving local interactions with a dynamic interaction model, with partial observability and no communications.

We describe our approach to compute a near-optimal policy for a given agent, using our model. First, we give an algorithm which extract the decision making problem of the agent, from the multiagent problem. Then, we give a second algorithm able to solve this decision making problem. Finally, we present the performance of these techniques, with experimental results using the existing dedicated benchmarks. For each benchmark, we compare the quality of our policies with the underlying MMDP, we show the efficiency of our approach and how we can scale up to large problems with good computation times.

[1]NEXP is the set of decision problems that can be solved by a non-deterministic Turing machine using time $O(2^{p(n)})$ for some polynomial $p(n)$, and unlimited space.

2 BACKGROUND

This study concerns the scalability of DEC-POMDPs (which are cooperative stochastic games) by using a different point of view on how to solve them and overcome the curse of dimensionality. For this purpose, we introduce DEC-POMDPs as the classic model for partially observable multiagent decision making. Then, we introduce POMDPs (Cassandra et al., 1994) and Q-MDPs as models for monoagent problems with partial observability and MMDPs (Boutilier, 1996) as a model for multiagent problems model with full observability: we will use those two frameworks.

2.1 Stochastic Games

A stochastic game (Shapley, 1953) is defined by a tuple $\langle I,S,A,R,T\rangle$ with (1) I the number of players in the game, (2) S a set of states in which the game can be (each state being similar to a classical game with I players), (3) $A = A_1 \times A_2 \times \ldots \times A_I$ a set of joint actions ($A_i = \left\{a_i^1,\ldots,a_i^{|A_i|}\right\}$ the actions player i can do), (4) $R = \{R_1,R_2,\ldots,R_I\}$ the set of `reward functions` ($R_i : S\times A \to \mathbb{R}$ the reward function of player i which gives, in each state, the positive or negative reward associated to each joint action) and (5) $T : S\times A\times S \to [0,1]$ the joint `transition function` ($T(s,a,s')$ the probability for the I players applying an action a in a state s to move to a state s').

Players can know the actual state, or only know a probability distribution over a subset of possible states. At each step, each player i chooses an action from its set A_i, based on the actual state s and on its `policy` π_i. The game then moves to a new state s'. The joint policy is given by $\pi = (\pi_1,\ldots,\pi_I)$.

2.2 Decentralized Partially Observable Markovian Decision Processes

DEC-POMDP (Bernstein et al., 2000) is a model formalizing cooperative multiagent decision-making in partially observable environments. A DEC-POMDP is a specific stochastic game, described with a tuple $\langle I,S,A,T,R,\Omega,O\rangle$, where each player is called an "agent", $S = \{s^1,\ldots,s^k\}$ is a set of k joint states ($\forall i, s_j^i$ is the individual state of agent j) and $A = \{a^1,\ldots,a^l\}$ and T are similar to a stochastic game. There is only one reward function $R : S\times A\times S \to \mathbb{R}$ (we could write $R = R_1 = R_2 = \ldots$) and we add $\Omega = \{o^1,\ldots,o^j\}$ a set of joint observations the agents can receive about the environment and $O : S\times A\times S\times\Omega \to [0;1]$ an `observation function` giving the probability to receive an observation $o\in\Omega$ after a transition $s \to_a s'$.

The main difference, with a "standard" stochastic game, is that we only have one reward function R for all the agents, instead of one per agent. The agents are then cooperative, because they are optimizing the same reward function. Solving a DEC-POMDP means computing a joint policy π which gives, at any moment, the joint action $a\in A$ the agents will have to apply. We call `history` the sequence $(a^0,o^0,a^1,o^1,\ldots)$ of actions and observations done by the agents from the beginning of the execution. If we write H the set of all possible histories, a policy will be a function $\pi : H \to A$. Moreover, if we have a usable criterion to evaluate a given policy, we can compute an optimal policy π^*. Solving a DEC-POMDP is done by computing the optimal joint policy: the time complexity is NEXP-complete (Bernstein et al., 2000), which is very hard. Until now, existing algorithms only work on problems with limited agents, reducing their applicability for real world problems. The aim of our framework is to build a new model, able to solve larger problems, by breaking the complexity of a DEC-POMDP.

2.3 POMDP and Q-MDP

POMDPs are used to describe partially observable monoagent planning problems under uncertainty: we can see a POMDP as a specific DEC-POMDP, where $I = 1$. A POMDP is a tuple $\langle S,A,T,R,\Omega,O\rangle$ such that those elements are similar to a DEC-POMDP, in the monoagent settings. POMDPs have an interesting property: a history can be replaced by a `belief-state` b which is a probability distribution over S with $b(s)$ the probability to be in s. Belief-states are very hard to compute for DEC-POMDPs.

Solving a POMDP is a hard problem, so Q-MDPs (Littman et al., 1995) were introduced in order to simplify this task. The basis of Q-MDPs is to solve the underlying MDP and to compute a Q-value function $Q^{MDP} : S\times A \to \mathbb{R}$. Then we can write $\forall b,a : Q(b,a) = \sum_{s\in S} b(s).Q^{MDP}(s,a)$. It is then possible to extract from Q a policy π for the POMDP. However, such an approach is not exact, because we overestimate the value of each state (with the wrong hypothesis that we are able to act optimally with a full observability) and we underestimate the value of the epistemic actions (actions modifying the belief only, and not the environment).

2.4 Fully Observable Problems: Multiagent MDPs

A fully observable problem is such that an agent alone can know its state at any moment, so we don't need to use observations. MMDPs (Boutilier, 1996) are able to describe a multiagent fully observable problem with a tuple $\langle I,S,A,T,R\rangle$ (those elements are similar to a DEC-POMDP). Solving an MMDP means computing a joint policy $\pi : S \rightarrow A$. In order to compute this policy, we use the classical Bellman operator: for each state, we compute an optimal value function $V^*(s)$. Using it, we compute $\pi^*(s) = \text{argmax}_{a\in A}\sum_{s'\in S} T(s,a,s').[R(s,a,s') + \gamma V^*(s')]$ with $0 \leq \gamma < 1$. It is usually possible to write $\pi = \{\pi_1,\dots,\pi_I\}$ with π_i the policy of the agent i (from individual states to individual actions). The time needed to compute such a joint policy grows polynomially with $|S|$ the number of states, but $|S|$ grows exponentially with the number of agents.

3 RELATED WORKS

Recently, a lot of work has been developed to overcome the curse of dimensionality and scale partially observable multiagent problems to more than 2 agents, using interaction-based models. However, these models bring a new difficulty: if we make the assumption that agents interact only sometimes, with some other agents (local interactions), then we need to detect when these interactions occur. Some approaches use tasks allocation (Varakantham et al., 2009), a subclass of multiagent decision making. In the general case, the most promising works make strong assumption to detect these interactions, and thus suffer from limitations in terms of applicability.

3.1 Static Interaction Models

The ND-POMDP model (Nair et al., 2005; Kumar and Zilberstein, 2009) was introduced to describe problems with local interactions. Using this model, one can solve large problems, but is limited by two strong assumptions. First, ND-POMDPs address problems with a static interaction structure, meaning that an agent is always interacting with the same set of neighbors. Second, this model can only deal with dependencies over rewards, but not over transitions.

More recently, good results were achieved using Factored-DEC-POMDPs (Oliehoek et al., 2008). It is able to solve problems with more than two agents while keeping dependencies between agents. However, those studies are based on the same kind of problems as ND-POMDPs, with static interactions.

3.2 Task-based Interaction Models

Distributed POMDPs can be used to describe multiagent problems, using *coordination locales* (Varakantham et al., 2009). Using such an approach, the agents are mainly independent, but share a set of tasks which can be done or not, and transition and reward functions depend not only on the state of the agent, but also on the set of tasks. Moreover, an agent does not observe anything about the other agents, but receive observations about tasks (and about its own state).

Using such a model, it is possible to deal with a lot of agents, with the solving process being highly decentralized. However, this model describes problems with a task allocation component. Such problems are only a subclass of multiagent decision making, and a lot of other problems need more complex interactions.

3.3 Dynamic Interaction Models

The IDMG model (Spaan and Melo, 2008) was introduced to describe problems with local interactions, like ND-POMDPs, but with a dynamic interaction model. Using IDMGs, one can describe large problems with dependencies between agents, and is no more limited to static interactions: each agent interacts with an evolving set of agents. Moreover, using this model, Spaan et al. were able to compute near-optimal policies on a set of dedicated benchmarks.

However, this model introduces new strong assumptions. First, each agent has to know its own state (it is a full local observability). Second, an agent can use unlimited and free communication with the agents it is interacting with. So, the sub-problem of a given interaction becomes fully observable. Then, the problems addressed by this solver are not similar to DEC-POMDPs but are a good threshold between DEC-POMDPs and MMDPs. Our approach is similar to IDMG, extended to partial observability.

4 THE MODEL: DYLIM

Our goal, with the DyLIM, is to describe multiagent decision making problems with no assumption (excepted the idea that wa can give a finit set of all the possible interactions between two agents). We deal with dependencies over transitions, rewards and observations. Moreover, we deal with partial observability, over the state of the agent and over the other agents. Finally, we use a dynamic interaction model without needing any communication between agents. In this section, we introduce our model and how to use it, to describe a multiagent decision making problem.

4.1 Local Interactions

DyLIM is built on a simple idea: it is not always necessary to consider all the agents as fully dependent of each other. For example, in a group of robots evolving in a given environment, each has to move to a given point while observing the neighbors so they don't collide. However, the agent does not need to remember the positions of the agents outside the neighborhood. So, in a given state, an agent is only interacting with a subset of the I agents involved in the problem.

Such a model deals with multiple kind of applications. We will focus on examples derived from those described in (Spaan and Melo, 2008). We consider the ISR problem where I agents evolve into a map (fig.1). In this problem, two agents start at a random position and have to reach a given target (the two 'X' in fig.1). The agents must not be in the same place at the same moment. An agent receives observations about walls surrounding it and about the relative position of the other agent, but observation accuracy decreases with the distance. We will use the ISR problem as an example in the following sections.

4.2 Main Idea

We can split a multiagent problem into two parts. The first one describes the individual problem, used for a single agent "ignoring" the other agents (with fully independent transition, reward and observation functions). In ISR, it describes how the agent evolves in the map, gets observations about the walls and receives a reward when it reaches its target. The second part of the problem describes its coordination aspect: how the agent influences the agents with which it is in interaction (we call those agents `interacting agents`). In ISR, it describes how the agent gets observations about its neighbors, how they evolve and how a negative reward is received when colliding.

This second part is described by relations between the agent and its interacting agents, observations the agent can receive about those relations and joint functions (rewards and transitions) describing how those relations will evolve. Finally, the problem is a couple $Pb = (< individual >, < coordination >)$. We only have to solve those two problems, and to extract a solution for the global problem, so we can give for each state s a vector-value function $V(s) = (V^{ind}(s), V^{coo}(s))$. Figure 2 describes this principle.

4.3 Individual Part

This part is a classical POMDP. We represent the planning problem of a single agent with a tuple $\langle S, A, T, R, \Omega, O\rangle$ which will be solved with any POMDP solver. This part is exact for states where the agent has no interaction, but could induce mistakes on transitions, rewards or observations for states with interactions. Then, we use the "coordination" part.

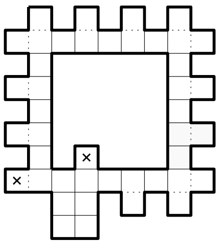

Figure 1: The ISR problem.

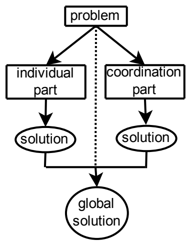

Figure 2: General idea.

4.4 Coordination Part

We show in this section how the coordination problem of an agent i with all its interacting agents is described with a tuple $\langle SR, \Omega R, OR, C\rangle$.

4.4.1 *SR*: Relations

SR is a set of `relations`, describing how an agent ag_k can be interacting with agent ag_i.

Definition 1 (Relation). *A relation $\mathcal{R}^l$ describes a property l between a state of ag_i and one of its interacting agents ag_j: $\mathcal{R}^l = \{(s_i, s_j) | l(s_i, s_j) = True\}$*

We write $SR = \{\mathcal{R}^1, \ldots, \mathcal{R}^{|SR|}\}$ the set of all possible relations and we have: $\forall \mathcal{R}^l, \mathcal{R}^k \in SR, \mathcal{R}^l \cap \mathcal{R}^k = \emptyset$. In the ISR example, one relation will be `Front`, meaning an agent j is in front of agent i. When an agent ag_i, in a state s_i, perceives an interacting agent ag_j, it can build the couple (s_i, s_j) and find the associated relation $\mathcal{R}$. Then, we say this relation is the `relative state` rs_j of agent ag_j.

According to this definition, a joint relative state rs is such that $rs = (rs_1, \ldots, rs_k)$ with rs_j the relative state associated to agent j. In the ISR problem, we imagine that an agent i (in state $s_i = (x2, y3)$) detects an agent j in front of him ($s_j = (x2, y4)$). We write $rs_j = front$ the relative state of ag_j, because we have $front(s_i, s_j) = True$. If ag_i detects another agent k on its left, we write $rs = (front, left)$. Figure 3 is an example of such a joint relative state: the joint relative state for the middle agent is $(far - w; near - sw; far - ne)$. The fourth neighbor is too far to be observed, and thus is not a part of the interaction.

4.4.2 ΩR and OR: Observing Relative States

We described in sec. 4.3 how the individual component is represented with a POMDP, so we have an ob-

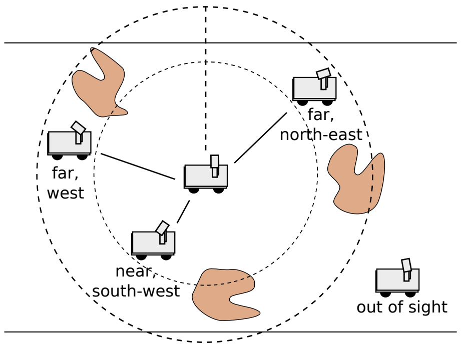

Figure 3: Example of a joint relative state.

servation set Ω and an observation function O used for the individual states $s_i \in S$. Likewise, we have an observation set ΩR and an observation function OR used for the joint relative states. OR gives a probability to receive an observation $o \in \Omega R$, once agents acted and moved to a new joint state: it is used to estimate the relative state of each interacting agent. In ISR, ΩR describes if a neighbor is observed, in which direction (front, right, left, behind) and how far.

We have $\Omega R = \{o^1, \ldots, o^{|\Omega R|}\}$ a set of observations an agent can receive about its interacting agents and $OR : \Omega R \times \bigcup_{i=0}^{m} SR^{(i)} \to [0,1]$ the `joint observation function`. Here, $\bigcup_{i=0}^{m} SR^{(i)}$ is the set of each possible joint relative state involving 0 to m interacting agents. This joint observation function returns a probability, according to an observation and a transition to a new joint relative state.

4.4.3 *C*: Relation Clusters

In real world problems, it is often possible to extract several sub-problems. If we consider a mobile robots example and we add a door in the environment, then we have two sub-problems: navigate in open spaces, and cross doors. It is then possible to work on those two problems independently, in order to break the combinatorial complexity. Now, let us bring this idea to our framework. In a given situation, interacting agents are represented by a joint relative state. So, for a given sub-problem, we have a set of joint relative states. We call this set a `relation cluster` (RC). In $\langle SR, \Omega R, OR, C\rangle$, those RCs are represented by the set $C = \{RC^1, ..., RC^{|C|}\}$. Those clusters are not computed: it is an input of the problem like states, actions, etc. We can see C as a representation of "how an agent interacts with other agents". This is an important aspect of this approach: for this reason, the next section is dedicated to its formal description.

4.5 The "Relation Cluster" Concept

We can see a relation cluster as a situation, involving an agent ag_i interacting with other agents in a specific sub-problem associated to a set of joint relative states. First, we give a formal definition. Second, we explain how to choose those clusters, while designing the problem description.

4.5.1 Definition

An agent i builds a cluster $RC^n = (S^n, T^n, R^n)$ where $S^n = \{rs^1, ..., rs^{|S^n|}\}$ is a set of joint relative states (such that $\forall rs^j \in S^n, rs^j = (rs^j_1, ..., rs^j_{|rs^j|})$, with $rs^j_k \in SR$ the relation describing how the agent k interacts with the agent i), $T^n : S^n \times A \times S^n \to [0,1]$ is a joint transition function associated to the cluster n (with A the same set of actions as in the individual problem) and $R^n : S^n \times A \times S^n \to \mathbb{R}$ is a joint reward function.

T^n and R^n are used to describe dependencies. Each joint relative state can involve dependencies over transitions, rewards or both. In DEC-POMDPs, T and R are given for all joint states $s \in S$ while in our approach, they are only given for expressing dependencies. In ISR, we could imagine that an agent would not be able to cross a corridor if another agent is already crossing it. So, if ag_i is in a corridor and considers that the joint relative state is $(Front)$, we have a dependency over transitions. But, if ag_1 considers that the joint relative state is $(Behind)$, we do not have any dependencies, because ag_1 will be able to cross the corridor without colliding with the other agent. Still in this example, there are dependencies over rewards for each joint relative state, because an agent will receive a negative reward if it collides.

In order to compute a transition probability and a reward for a given (rs, a, rs'), we will use T^n if the associated transition implies dependencies (and R^n if the reward implies dependencies). Otherwise, we will use the individual functions T and R, coming from the individual POMDP, to build the joint transitions and rewards. Those joint functions only depend on the action of ag_i. More details are given in sec.5 about how to build those functions.

4.5.2 How To Build Clusters

We described C as a set of k relation clusters $RC^n = (S^n, T^n, R^n)$. Those $S^1, \ldots, S^k$ are partitions of the set of all possible joint relative states. With m the maximum number of interacting agents involved at the same time, we have $\forall rs \in \bigcup_{i=0}^{m} SR^{(i)}, \exists RC^j | rs \in S^j$ and $\forall i, \forall j, S^i \cap S^j = \emptyset$.

The goal is to split the set of joint relative states into n clusters. First of all, we build an "empty" cluster, associated to situations where agent i has no interaction with any other agent. Such a cluster will be $RC = (\emptyset, \emptyset, \emptyset)$, which means we have no joint relative state, no joint transition and no joint reward. In such a cluster, the agent can follow an individual and independent policy with no need of coordination. The next step consists in identifying sub-problems in order to classify remaining joint relative states. Good clusters will be such that there are many transitions between the joint relative states of a given cluster, but only a few transitions between two different clusters (weakly coupled clusters and strongly coupled relative states in the same cluster). In ISR, we could build a cluster associated to the corridor crossing sub-problem, and solve it as an independent problem without introducing too much approximations.

4.6 Sub-optimality of the Approach

Using this model, each agent will solve its own individual problem, taking into account the existence of the neighborhood. Such an approach is sub-optimal, compared to standard approaches which compute optimal joint policies, but gave good results as shown in our experiments (section 6.2). However, is it really a good idea, to always seek the optimal joint policy? It is proved that DEC-POMDPs are not approximable (finding epsilon-approximations of the joint policy is NEXP-hard (Rabinovich et al., 2003)). Then, could not we just compute "good enough" policies, in order to scale up to real world problems?

Our work is based on this idea, to quickly compute a good policy, and avoid the huge amount of computation steps necessary to find the optimal one, while our is good enough. Then, a difficult problem is to determine if a policy is good enough. Our model gives everything the agent needs to take its decisions: we described not only how the agent evolves in its environment and receives rewards, but also how the other agents (the interacting ones) impact these rewards. The next section introduces a set of algorithms, able to compute a policy using this model.

5 ALGORITHMS

We developed two algorithms able to solve a problem described with a DyLIM. First, we describe how to find an upper bound for the combinatorial complexity. Second, we give some details about how we build the interaction problem. Third, we introduce our algorithms solving the problems expressed with DyLIM.

5.1 Approximate Joint Relative States

The number of possible joint relative states grows exponentially with the number of involved agents: with M the number of possible relations and I the number of agents, we have (in the worst case) M^I different joint relative states (see §6.1 for a more detailed complexity analysis). In order to bound this combinatorial explosion, we apply the same behavior as a human evolving in a crowd. In such a situation, the human only considers a subset of the people surrounding him. For example, he tries to avoid colliding with people in front of him only.

We apply this idea in our algorithms with two assumptions. First, we consider that one relation can involve several agents. For example, if the agent has three agents in front of it, and two on its left, we consider that the induced joint relative state is $(front, left)$ and not $(front, front, front, \dots)$. Second, we have a preference order between relations and we consider a maximum of N relations at the same time. For example, with $N = 2$ and the order $front > left > behind$, the joint relative state $(front, behind, left)$ would be reduced to $(front, left)$. Because of those assumptions, we are able to bound the combinatorial explosion at N (see §6.1). If N is large enough, we compute good policies (for example, in a navigation problem, $N = 4$ is enough to consider any immediate danger).

5.2 Building the Interaction Problem

The individual part of the problem is fully described with the tuple $\langle S, A, T, R, \Omega, O \rangle$, while the tuple $\langle SR, \Omega R, OR, C \rangle$ used to describe the interaction part needs a preliminary preprocessing before being used. We already have the set of joint relative states ($S^1, \dots, S^n$ from each relation cluster $RC^n \in C$) and the observation function (ΩR and OR). In order to completely define the model, we define the joint transition and reward functions, using C: we formalize each relation cluster as a *nearly independent* MMDP.

5.2.1 Computing Transitions and Rewards

We consider an MMDP for each relation cluster $RC^n = (S^n, T^n, R^n)$ with $S^{MMDP} = S^n$ and $A^{MMDP} = A$. For a given MMDP, we compute $T(rs, a, rs')$ for each tuple (rs, a, rs') using algorithm 1, the idea for computing $R(rs, a, rs')$ being the same. In this algorithm, we compute a transition between two joint relative states, knowing that a joint relative state $rs = (front, left)$ could be associated to the joint state $s^1 = ([x3, y0], [x3, y1], [x2, y0])$, or $s^2 = ([x2, y5], [x2, y6], [x1, y5])$ etc. We call those joint

Algorithm 1: Computing a transition.

```
Input: $rs$, $a$, $rs'$; $RC^i = (S^i, T^i, R^i)$ a relation cluster
Result: the transition probability $T(rs, a, rs')$
if $T^i(rs, a, rs')$ is defined then return $T^i(rs, a, rs')$
$T^i(rs, a, rs') \leftarrow 0$;
foreach $max(0, |rs - rs'|) \leq \texttt{out} \leq min(|rs|, I - |rs'|)$
do
    foreach subset G from rs with out agents do
        foreach part rs" of rs' such that
        $|rs"| = |rs| - out$ do
            // probability for the agents
               of $G$ to leave the
               interaction:
            $p^{out} \leftarrow \sum_{RC^j \neq RC^i} P_G(RC^j | RC^i, a)$;
            // probability for the $|rs| - out$
               agents not in $G$ to move to
               the state $rs"$:
            $p^{stay} \leftarrow P_{ag \notin G}(rs" | rs, a)$;
            // probability for $|rs'| - |rs"|$
               agents from the outside to
               move to $rs' - rs"$:
            $p^{in} \leftarrow P_{out}((rs' - rs") | out, a)$;
            // the global probability:
            $P^{(G,s")}(rs, a, rs') \leftarrow p^{out} \times p^{stay} \times p^{in}$;
        $P^G(rs, a, rs') \leftarrow \sum_{rs"} P^{(G,rs")}(rs, a, rs')$;
    $T^i(rs, a, rs') \leftarrow T^i(rs, a, rs') + \sum_G P^G(rs, a, rs')$;
return $T^i(rs, a, rs')$;
```

states `instances`. We do not use all the instances of *rs* but only a representative set. Indeed, a given joint relative state can be associated to a lot of instances, but it is not always interesting to use all of them (most of the time, we only need to consider *k* of the *n* available instances in order to compute transitions and associated rewards, so we can avoid a lot of useless computation steps).

This algorithm is based on *rs* and rs' the set of interacting agents. Because of that, we suppose that several agents could leave the interaction after moving from *rs* to rs', and some others could join the interaction. In the algorithm, p^{out}, p^{stay} and p^{in} are used to compute these probabilities:

- p^{stay}, the probability for the agents not in G to stay in interaction, can be an input of the problem or computed from the POMDP. Its not an exact probability (which would be impossible without knowing the action taken by the other agents), but a *reachability* for all the joint relative states,
- p^{out} is the probability for the agents in G to move to a state where they are not interacting anymore,
- p^{in} is the probability, for $k = |rs'| - |rs"|$ agents not interacting with the agent, to move to the joint relative state $(rs' - rs")$. This probability is computed assuming a uniform distribution of the $I - |rs|$ remaining agents over unknown states.

Figure 4 is an example of such a transition.

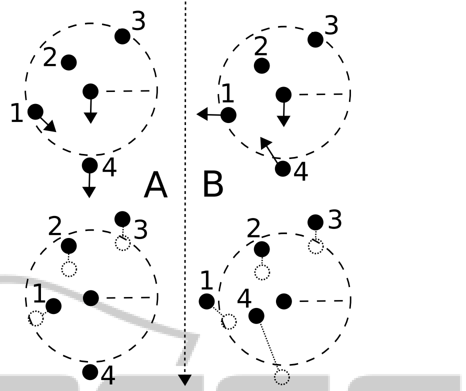

Figure 4: Transition between two joint relative states.

In example A, the agent is interacting with 3 other agents (top-let part of the figure). The agent moves to the south, agents 2 and 3 do not move, agent 1 moves to the south-east and agent 4 moves to the south. The bottom-left part of the figure is the result of those moves: the joint relative state is now approximately (top,left), agents 3 and 4 being too far to be considered. Here, p^{out} is the transition probability for agent 3, p^{stay} is the probability for agents 1 and 2 and $p^{in} = 1$ (no agent came from the outside of the interaction). Probability for transition A is $p^{stay} \times p^{out} \times p^{in}$. In example B, the initial situation is the same (top-right part of the figure), but agent 1 moves to the west and agent 4 moves to the north-west. The result is the same as example A, the joint relative state being (top,left), but p^{out} is now the probability for agents 1 and 3, p^{stay} the probability for agent 2 and p^{in} the probability for agent 4. Then, the global transition probability is the sum of examples A and B. We can see, in this example, how a transition between two joint relative state is computed as a sum of each possible case. We compute transition and reward functions for each MMDP: we have to compute those functions between two different MMDPs.

5.2.2 Transitions between Relation Clusters

To compute the transition probability from a relation cluster to another one, we compute transitions between their corresponding MMDPs (and the associated rewards). So, for a given $MMDP^i = \langle S^i, A^i, T^i, R^i \rangle$ and for each $MMDP^j = \langle S^j, A^j, T^j, R^j \rangle$ with $j \neq i$, we apply algorithm 2.

In line 4 of this algorithm, we add an abstract state *j*. Being in this state means "the agent is no longer

Algorithm 2: Transitions between two MMDPs.

```
Input: $MMDP^i$, $MMDP^j$
Result: updated $MMDP^i$
// $j$=abstract state representing $MMDP^j$:
$S^i \leftarrow S^i \cup \{j\}$;
$A^i \leftarrow A^i \cup A^{POMDP}$;
foreach $rs \in S^i$ and $a \in A^i(rs)$ do
    // we compute $T(rs,a,rs')$ and $R(rs,a,rs')$
       with algorithm 1:
    $T^i(rs,a,j) \leftarrow \sum_{rs' \in S^j} T(rs,a,rs')$;
    $r \leftarrow \frac{\sum_{rs' \in S^j} [T(rs,a,rs') \times R(rs,a,rs')]}{\sum_{rs' \in S^j} T(rs,a,rs')}$;
    $R^i(rs,a,j) \leftarrow r$;
foreach $rs \in S^i$ and $a \in A^{POMDP}$ do
    $R(j,a,rs) \leftarrow 0$;
    if $rs \neq j$ then $T(j,a,rs) \leftarrow 0$;
    else $T(j,a,rs) \leftarrow 1$;
return $MMDP^i$;
```

in the relation cluster i, but in the relation cluster j". Then we compute the associated transitions, between j and each of the other states, describing how the agent can move toward another MMDP. During the solving process, we will change the value of this abstract state, so it reflects the interest for the agent to move toward the associated relation cluster.

5.3 The Solving Method

In this section, we present two methods to solve both the individual and the interaction parts. Then, once Pb^{ind} and Pb^{coo} are solved, we can give for each state s a vector-value function $V(s) = (V^{ind}(s), V^{coo}(s))$.

5.3.1 Computing a Relational Belief State

We can easily compute a *relational belief state* b^r over the set of joint relative states, like we do in a POMDP. Such a belief state is needed in our two methods. With $S(rs) = \{s \in S | \forall rs_j \in rs, (s,-) \in rs_j\}$, $b^{ind}(rs) = \sum_{s \in S(rs)} b^{ind}(s)$ and knowing a the action taken by the agent, o the observation received about the neighbors and b^{ind} the belief state about the individual problem, the equation to update a relational belief state b^r_{t-1} to b^r_t is the following: $b^r_t(rs') =$

$$\frac{OR(rs',o) \sum_{rs} b^{ind}(rs) \cdot b^r_{t-1}(rs) \cdot T^n(rs,a,rs')}{\sum_{rs''} OR(rs'',o) \sum_{rs} b^{ind}(rs) \cdot b^r_{t-1}(rs) \cdot T^n(rs,a,rs'')}$$

5.3.2 POMDP+Q-MMDP

A first method consists in solving independently the individual problem and the interaction one. We use an existing POMDP solver, implementing the SARSOP (Kurniawati et al., 2008) algorithm, to solve the individual problem. Once this problem is solved, the agent has a Q-value function $Q^{ind}(b^{ind},a)$ giving an expected value for an action a in each possible individual belief-state b^{ind} about its states. We solve the interaction problem with an algorithm inspired from Value Iteration. This process is described in Algorithm 3, with E the set of abstract states representing a transition between two MMDPs.

Algorithm 3: Solving nearly independent MMDPs.

```
Input: $M$ a set of MMDPs, $\varepsilon$ a bound
$E \leftarrow \{abstract\ states\}$;
foreach $MMDP^i \in M$ and $s \in S^i$ do
    if $s \in E$ then $V[i](s) \leftarrow 0$ else $V[i](s) \leftarrow R^i(s)$;
repeat
    // step 1 (doing Value Iteration):
    foreach $MMDP^i \in M$ do
        $V'[i] \leftarrow VI(MMDP^i)$;
        foreach $s \in E$ do $V'[i](s) \leftarrow 0$;
    // step 2 (propagating values):
    foreach $MMDP^i \in M$ do
        foreach $MMDP^j \in M$ such that $j \neq i$ do
            // $E[i]$ representing $MMDP^i$:
            $V'[j](E[i]) \leftarrow \max_{s \in S^i}(V'[i](s))$;
    // step 3 (computing values change):
    $\Delta \leftarrow 0$;
    foreach $MMDP^i \in M$ do
        $\delta \leftarrow 0$;
        foreach $s \in S^i$ do
            $d \leftarrow |V[i](s) - V'[i](s)|$;
            if $d > \delta$ then $\delta \leftarrow d$;
        if $\delta > \Delta$ then $\Delta \leftarrow \delta$;
    $V \leftarrow V'$;
until $\Delta < \varepsilon$;
return $V$;
```

Then we finally have, for each MMDP, a Q-Value function $Q^{MMDP}(s,a)$ giving a value for each couple (state,action). Moreover, those functions consider the possibility to move from an MMDP to another, so the agent can seek the best relation cluster according to its current state. However, this function works in fully observable settings, so we have to build a new function (using this one), working in partially observable settings. With $MMDP(s)$ the MMDP i such that $s \in S^i$, we have $Q^{coo}(b^r,a) = \sum_{s \in b^r} b^r(s).Q^{MMDP(s)}(s,a)$.

However, such an approach implies the same problems as those associated to Q-MDPs (see

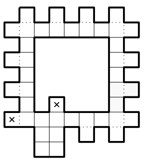
Figure 5: ISR.

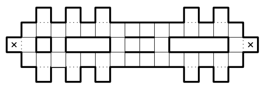
Figure 6: MIT.

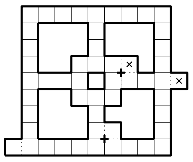
Figure 7: PENTAGON.

sec. 2.3), such as sub-optimality. During the problem execution, we keep a belief-state over individual states, and another one over joint relative states. Using those beliefs, we can compute at each timestep an individual value V^{ind} and an interaction value V^{coo} for each action. Then, we choose the action offering the best tradeoff between those two values. This method gives good results on the benchmarks, but it could be improved: the impact of the interaction over the individual decisions is only computed at horizon 1.

5.3.3 Augmented-POMDP

In this second approach, we compute an *Augmented-POMDP* $\langle S^{aug}, A^{aug}, T^{aug}, R^{aug}, \Omega^{aug}, O^{aug} \rangle$. We use this augmented POMDP to describe the individual problem of the agent augmented with informations about the other agents. We write (1) $S^{aug} = S \times (\bigcup_{i=0}^{|C|} S^i)$, (2) $\Omega^{aug} = \Omega \times \Omega R$ and (3) $A^{aug} = A$. The transition function $T^{aug}((s,rs),a,(s',rs'))$ is given by $T(s,a,s') \times T^{MMDP}(rs,a,rs')$ (the observation and reward functions O^{aug} and R^{aug} are defined in the same way). T^{aug} is the product of two independent functions: we use this product because DyLIM constructs the individual and the coordination problems as two independent parts. Then, we solve this POMDP using SARSOP. This approach gives to the interactions a long-term impact over the individual decisions. We had nearly-optimal results using this method.

6 PERFORMANCES ANALYSIS

In this section, we analyze the performances of our approach. First, we give the complexity of our algorithms. Second, we give some experimental results, solving a set of dedicated benchmarks.

6.1 Complexity of the Algorithms

There are two computation steps in our approach. First, we compute the individual problem of the agent and second, we solve this problem. The first step implies computing k transitions and rewards: in the worst case, all the agents are always interacting and we have $k = |S|^I \cdot |A| \cdot |S|^I$. Computing a transition (or a reward) means computing p^{out}, p^{in} and p^{stay} which is done in $O(X \cdot I)$ with X the maximum number of representative instance for a given joint relative state. X can be fixed constant and low (sec. 5.2.1), so the global complexity is in $O(|S|^{2I} \cdot |A| \cdot I)$. We described in sec. 5.1 how the maximum number of relations considered at the same time can be bound by N. Then, with $M = \min(I,N)$, the complexity for the first step is in $O(|S|^{2M} \cdot |A| \cdot M)$. This is exponential while the number of agents is less than the bound N. With more than N agents, the complexity stays constant. The second step consists in solving the augmented POMDP associated to the individual problem. The complexity is known to be P-SPACE complete. If this augmented POMDP is too large to be solved, then we apply the POMDP+MMDP approach, solving a little POMDP plus an MMDP (P complete).

6.2 Experimental Results

We chose, to evaluate our approach, to use a set of problems recognized as dedicated benchmarks in the community, coming from (Spaan and Melo, 2008).

6.2.1 Quality of the Behavior

We used instances of the problem described in Sec. 4.2. In each of them (fig. 5 to 7), the state is described by a couple (x,y) and a direction (N,S,E,W). A reward of +10 is assigned when an agent reaches a target, after what this agent can't move anymore. If two agents collide, a negative reward of -100 is assigned (the negative reward is not assigned one more time if the agents stay collided, but only if they collide again after separation). Each agent can move forward, turn left or turn right with no cost, but each time an agent moves forward it might derive with a probability of 0.05 for each side. Finally, an agent does not know its position but receives observations about the surrounding walls, and about its neighborhood: for each direction (front, corner-front-right, right, corner-right-behind, ...), the agent detects if there are neighbors or not (we don't know how many) and how far is the closest. Our results are presented in table 1.

For each instance, we ran 1000 simulations (over 30 timesteps) and we solved each of them with individuals POMDPs (agents acting independently: lower

Table 1: Average Discounted Rewards (ADR).

Instance	Ind. POMDPs	DyLIM (POMDP +MMDP)	DyLIM (Augm. POMDPs)	MMDP
ISR	-14.8	8.49	11.28	12.84
MIT	-14.5	9.18	11.57	12.96
PEN	-14.94	8.39	10.72	13.11

bound), with our two algorithms and with the underlying MMDP (each agent is given a full observability: *optimal* bound). With s_t the joint state during the timestep t, we computed these values using the equation: $ADR = \sum_{t=0}^{30} 0.95^t \times R(s_t)$.

Using our first algorithm (POMDP+MMDP), we have average results (better than individual POMDPs), but not satisfying enough: the agents manage to avoid collision, however once an agent is on a target, the other agents stay near without moving to the next target. Using the Augmented POMDP, the agents have a long-term view of their interactions, so they are able to reach each target. Then, we have nearly-optimal results (in comparison to MMDP). In each of those instances, the policy was computed in less than 2 minutes (including the time needed to read the input, compute the interaction problem, build the Augmented-POMDP and solve it using SARSOP).

6.2.2 Scalability

We developed experiments about scalability, with more than two agents and the results are very encouraging. Computation times for those experiments are presented in table 2 (in each instance, the quality was as good as in the previous examples).

Table 2: Computation times (’ = min., ” = sec., H = Hour).

N	I (number of agents)						
	2	3	4	5	6	7	100
1	3”	3”	3”	3”	3”	3”	3”
2	3”	27”	27”	27”	27”	27”	27”
3	3”	27”	7’	7’	7’	7’	7’
4	3”	27”	7’	3H	3H	3H	3H
5	3”	27”	7’	3H	-	-	-

This table gives the computation times according to the number of agents I and the bound N over the number of interacting agents considered at the same time. Intractable instances have a “-” value. We can see that N efficiently bounds the computation time. Moreover, we are able to compute policies with $N = 4$, which is a large enough bound for those instances.

7 CONCLUSIONS

This paper introduced DyLIM, a model used to describe interaction-based DEC-POMDP-like problems, and two solving algorithms. This model allows us to describe problems where an agent interacts only sometimes, with a small set of agents, which is a more realistic approach than classical models. Using this model, one can describe problems involving dynamic interactions (contrary to models such as ND-POMDPs), without strong assumptions (contrary to IDMG for example). We also described, using our model and its structure, how to break the combinatorial complexity and compute good policies in a bounded time. We were finally able to scale up to large problems, involving up to 100 agents.

REFERENCES

Bernstein, D., Zilberstein, S., and Immerman, N. (2000). The complexity of decentralized control of markov decision processes. In *Proc. of UAI*.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *TARK*.

Cassandra, A., Kaelbling, L., and Littman, M. (1994). Acting optimally in partially observable stochastic domains. In *Proc. of AAAI*.

Kumar, A. and Zilberstein, S. (2009). Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proc. of AAMAS*.

Kurniawati, H., Hsu, D., and Lee, W. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*.

Littman, M., Cassandra, A., and Pack Kaelbling, L. (1995). Learning policies for partially observable environments: Scaling up. In *Machine Learning*, pages 362–370.

Nair, R., Varakantham, P., Tambe, M., and Yokoo, M. (2005). Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *Proc. of AAAI*.

Oliehoek, F., Spaan, M., Whiteson, S., and Vlassis, N. (2008). Exploiting locality of interaction in factored Dec-POMDPs. In *Proc. of AAMAS*.

Rabinovich, Z., Goldman, C., and Rosenschein, J. (2003). The complexity of multiagent systems: The price of silence. In *Proc. of AAMAS*.

Shapley, L. (1953). Stochastic games. In *National Academy of Sciences*.

Spaan, M. and Melo, F. (2008). Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *Proc. of AAMAS*.

Varakantham, P., Kwak, J., Taylor, M., Marecki, J., Scerri, P., and Tambe, M. (2009). Exploiting coordination locales in distributed POMDPs via social model shaping. In *Proc. of ICAPS*.