

**AAMAS**  
**2013**

**Saint Paul**  
**Minnesota**  
**USA**

**6th - 10th May 2013**

twelfth  
international  
conference  
on  
autonomous  
agents and  
multiagent  
systems

## **W14 - The 8<sup>th</sup> Workshop Multiagent Sequential Decision Making Under Uncertainty (MSDM 2013)**

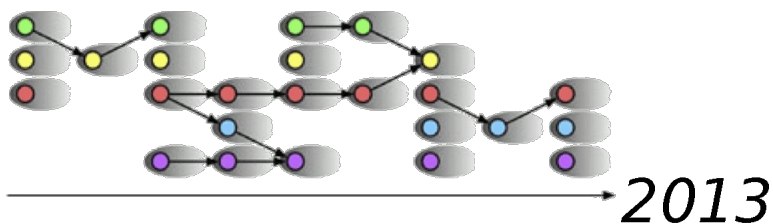
**Prashant Doshi (University of Georgia)**  
**Stefan Witwicki (INESC-ID, Instituto Superior Técnico)**  
**Jun-young Kwak (University of Southern California)**  
**Frans A. Oliehoek (Maastricht University)**  
**Brenda Ng (Lawrence Livermore National Laboratory)**

**May 7**

Proceedings of the  
Eighth Annual Workshop  
on  
**Multiagent Sequential Decision Making Under Uncertainty**  
(MSDM-2013)

**Held in St. Paul, Minnesota, USA  
in conjunction with AAMAS**

**May 7, 2013**



## Organizing Committee

Prashant Doshi	University of Georgia
Stefan Witwicki	INESC-ID, Instituto Superior Técnico
Jun-young Kwak	University of Southern California
Frans Oliehoek	Maastricht University
Brenda Ng	Lawrence Livermore National Laboratory

## Program Committee

Martin Allen	University of Wisconsin - La Crosse
Christopher Amato	MIT
Bikramjit Banerjee	University of Southern Mississippi
Raphen Becker	Google
Daniel Bernstein	Fiksu, Inc.
Aurélien Beynier	University Pierre and Marie Curie (Paris 6)
Alan Carlin	University of Massachusetts
Georgios Chalkiadakis	Technical University of Crete
François Charpillet	INRIA-Loria
Ed Durfee	University of Michigan
Alberto Finzi	Università di Napoli
Piotr Gmytrasiewicz	University of Illinois Chicago
Claudia Goldman	GM Advanced Technical Center Israel
Akshat Kumar	IBM Research, India
Michail Lagoudakis	Technical University of Crete
Francisco Melo	Instituto Superior Técnico/INESC-ID
Hala Mostafa	BN Technologies
Abdel-Ilhah Mouaddib	universit de Caen
Enrique Munoz de Cote	INAOE, Mexico
Simon Parsons	City University of New York
Praveen Paruchuri	Carnegie Mellon University
David Pynadath	Institute for Creative Technologies, USC
Zinovi Rabinovich	Mobileye
Anita Raja	University of North Carolina at Charlotte
Paul Scerri	Carnegie Mellon University
Jiaying Shen	Nuance Communications
Matthijs Spaan	Delft University of Technology
Peter Stone	University of Texas at Austin
Karl Tuyls	Maastricht University
Jianhui Wu	Amazon
Ping Xuan	Hewlett-Packard
Makoto Yokoo	Kyushu University
Chongjie Zhang	University of Massachusetts
Shlomo Zilberstein	University of Massachusetts

## Table of Contents

Planning under Uncertainty for Coordinating Infrastructural Maintenance . . . . .	1
<i>Joris Scharpff, Matthijs T.J. Spaan, Leentje Volker and Mathijs de Weerd</i>	
Asynchronous Execution in Multiagent POMDPs: Reasoning over Partially-Observable Events . .	9
<i>João V. Messias, Matthijs T.J. Spaan and Pedro U. Lima</i>	
Organizational Design Principles and Techniques for Decision-Theoretic Agents . . . . .	16
<i>Jason Sleight and Edmund H. Durfee</i>	
Rehearsal Based Multi-agent Reinforcement Learning of Decentralized Plans . . . . .	24
<i>Landon Kraemer and Bikramjit Banerjee</i>	
Counterfactual Regret Minimization for Decentralized Planning . . . . .	32
<i>Bikramjit Banerjee and Landon Kraemer</i>	
Automated Generation of Interaction Graphs for Value-Factored Decentralized POMDPs . . . . .	40
<i>William Yeoh, Akshat Kumar and Shlomo Zilberstein</i>	
Opponent modeling and planning against non-stationary strategies . . . . .	47
<i>Pablo Hernandez-Leal, Enrique Muñoz de Cote and L. Enrique Sucar</i>	
Qualitative Planning under Partial Observability in Multi-Agent Domains . . . . .	55
<i>Ronen I. Brafman, Guy Shani and Shlomo Zilberstein</i>	
Solving Dec-POMDPs by Genetic Algorithms: Robot Soccer Case Study . . . . .	61
<i>Okan Aşık and H. Levent Akin</i>	
A Coordinated MDP Approach to Multi-Agent Planning for Resource Allocation, with Applica- tions to Healthcare . . . . .	69
<i>Hadi Hosseini, Jesse Hoey and Robin Cohen</i>	
Bayesian Reinforcement Learning for Multiagent Systems with State Uncertainty . . . . .	76
<i>Christopher Amato and Frans A. Oliehoek</i>	

# Planning under Uncertainty for Coordinating Infrastructural Maintenance\*

Joris Scharpff Matthijs T.J. Spaan Leentje Volker Mathijs de Weerd

Delft University of Technology, Delft, The Netherlands  
{j.c.d.scharpff, m.t.j.spaan, l.volker, m.m.deweerd}@tudelft.nl

## ABSTRACT

We address efficient planning of maintenance activities on infrastructural networks, inspired by the real-world problem of servicing a highway network. A road authority is responsible for the quality, throughput and maintenance costs of the network, while the actual maintenance is performed by autonomous, third-party contractors.

From a (multi-agent) planning and scheduling perspective, many interesting challenges can be identified. First, planned maintenance activities might have an uncertain duration due to unexpected delays. Second, since maintenance activities influence the traffic flow in the network, careful coordination of the planned activities is required in order to minimise their impact on the network throughput. Third, as we are dealing with selfish agents in a private-values setting, the road authority faces an incentive design problem to truthfully elicit agent costs, complicated by the fact that it needs to balance multiple objectives.

The main contributions of this work are: 1) implicit multi-agent coordination on a network level through a novel combination of planning under uncertainty and dynamic mechanism design, applied to real-world problems, 2) accurate modelling and solving of the maintenance planning problem and 3) empirical exploration of the complexities that arise in these problems. Using two real-world application examples we introduce a formal model of the problem domain, present experimental insights and identify open challenges for both the planning and scheduling as well as the mechanism design communities.

## Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Multiagent systems

## General Terms

Algorithms, Experimentation

## Keywords

Multiagent systems, Planning and Scheduling, Coordination

## 1. INTRODUCTION

The planning and scheduling of maintenance activities on large infrastructural networks, such as a national highway network, is a

challenging real-world problem. While improving the quality of the infrastructure, maintenance causes temporary capacity reductions of the network. Given the huge impact of time lost in traffic on the economic output of a society, planning maintenance activities in a way that minimises the disruption of traffic flows is an important challenge for the planning and scheduling field. In this paper, we address this challenge by a novel combination of stochastic multi-agent planning, captured in Markov Decision Processes (MDPs), and dynamic mechanism design.

Maintenance activities in infrastructural networks not only incur maintenance costs, but also incur *social costs* based on the effects maintenance has on the throughput of the network. Put simply, closing off a main highway will lead to traffic delays, resulting in quantifiable losses in economic activity. For instance, an hour of time lost in traffic is valued between €10 for leisure traffic and €40 for business traffic [22]. Since maintenance activities are unavoidable, traffic delays are inevitable. However, they can be *minimised* using planning and scheduling techniques. In our work, we use planning-under-uncertainty methodology to schedule maintenance activities across the entire network, taking into account the effect that road closures have on traffic on nearby road segments.

A powerful real-world example of the benefits that careful maintenance planning can provide is the summer 2012 closure of the A40 highway in Essen, Germany. Instead of choosing for the default option of restricting traffic to fewer lanes for 2 years, authorities fully closed off a road segment for 3 months and diverted traffic to parallel highways. Traffic conditions on the other highways hardly worsened, while an estimated €3.5M in social costs due to traffic jams were avoided (besides lowering building costs) [9].

As maintenance activities often have an uncertain duration due to delays in construction, it is important to take uncertainty into account while planning. Also, there may be multiple ways to perform a certain maintenance action by varying the amount of resources dedicated to it, leading to options that have different duration, cost, risk and effect on asset quality. Furthermore, long-term planning is required to ensure overall network quality. Markov Decision Processes provide a suitable framework to model and solve these types of planning-under-uncertainty problems [19].

A complicating factor, however, is that while a single public road authority is responsible for the quality, throughput and costs of the network, the actual maintenance is performed by autonomous contractors, typically third-party companies interested primarily in maximising their profits. Road authorities face the problem of aligning objectives; we introduce monetary incentives for the contractors to consider global objectives. But an agent servicing one part of the network also influences agents in other parts as his work has a negative impact on the traffic flow. As a consequence, the payments we introduce lead to very high throughput penalties for all agents if

\*This work will also appear in the Proceedings of the ICAPS 2013 conference [20].

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with AAMAS, May 2013, St. Paul, Minnesota, USA.

they do not coordinate their maintenance plans on a network level.

In this work we focus on socially optimal joint maintenance planning that maximises the sum of contractor utilities, in the presence of such monetary incentives, and therefore we have chosen a centralised coordination approach. The authority is given the responsibility to develop socially optimal plans, while considering the individual interests of all contractors expressed through cost functions. However, as these cost functions are private information, optimal coordination and hence outcomes can only be achieved if the contractors report these costs *truthfully*. Ensuring this truthfulness is the key motivation to combine stochastic planning with mechanism design.

Our main contribution is the application of a combination of stochastic planning and dynamic mechanism design to realise truthful coordination of autonomous contractors in a private-values setting. Typical one-shot mechanisms often used to elicit private values are not suitable for contingent and repeated settings. Instead we focus on *dynamic mechanisms* that define payments over all expected outcomes such that in expectation it is in the agent's best interest to be truthful during the entire plan period. Applying dynamic mechanism design to (real-world) settings such as the one we consider is relatively unexplored territory [6].

### Related Work

Other approaches towards solving the problems discussed here have been considered, although they can not be applied to our setting for various reasons. Multi-agent MDP [4] assumes co-operative agents that are willing to share private information and have the same utility functions. This is not the case with decentralised MDP [3], however agent decisions are made solely on locally available information and are therefore inadequate in optimising network objectives. Moreover, both methods are not suitable when agents misreport their private information to 'cheat' the center into different outcomes. Non-cooperative settings have been studied in the classical planning literature [5, 14, 23], but uncertainty is not addressed.

Multi-machine scheduling has also been considered for the planning of maintenance activities, but we found this infeasible for our contingent setting. Moreover, finding optimal policies in multi-machine problems with general cost functions is highly intractable. The only work we are aware of in this area is [10], in which only non-decreasing regular step functions are considered. In our problem agents could both profit as well as suffer from concurrent maintenance, therefore our cost functions do not have the non-decreasing property.

Another interesting related approach is that of reinforcement learning [15, 16] and in particular Collective Intelligence [25]. In this an approach agents learn how and when to coordinate and, in the case of collective intelligence, strive to optimise a global goal, without substantial knowledge of the domain model. Nevertheless, the absence of domain knowledge makes it impossible for such methods to provide theoretical guarantees regarding agent and system performance – crucial if this method is to be applied in practice (e.g. as part of a dynamic contracting procedure) – and will most likely not produce socially optimal solutions.

Applying mechanism design to large multi-agent systems is challenging. First, few results are known for dynamic settings (but see [21]), but taking into account changes in the state of the system is crucial for planning ahead. Second, for large systems, computing optimal solutions might not be possible. When resorting to approximate solutions, however, standard theory for strategy-proof mechanisms does not immediately apply [18]. Finally, existing general solutions fail to apply in settings with actuation uncertainty. For instance, fault-tolerant mechanism design has been proposed to take

into account that agents might not be able to accomplish an assigned activity [17], but no techniques exist for the quite general forms of uncertainty that we address.

Both stochastic planning and mechanism design have been well studied independently, however only a handful of papers address dynamic mechanism design and/or a combination of the two. [2] proposed a dynamic variant of the VCG mechanism for repeated allocation, implementing the mechanism desiderata in a within-period, ex-post Nash equilibrium. [1] studied a dynamic variant of the AGV mechanism [8] that is budget-balanced in the weaker Bayes-Nash equilibrium solution concept. Highly related is the work by [7], in which the authors also study dynamic mechanism design to obtain desirable outcomes in multi-agent planning with private valuations. However, the focus is on allocation problems that can be modelled as multi-armed bandit problems, instead of the richer problem domains with dynamic states that we consider.

### Outline

In the next section, we present a theoretical framework for maintenance planning obtained and refined through interviews and discussions with public road and rail network authorities, as well as asset managers of several larger contractors. We then introduce the theoretical background of both stochastic planning and mechanism design (Section 3), and show how to combine work on planning with uncertainty and dynamic mechanism design to solve two example applications, derived from practice (in Section 4). We present experimental insights where we compare this with uncoordinated agents as well as with best-response (Section 5). In our conclusion, we summarise our findings and present open challenges for both the planning and scheduling as well as the mechanism design communities (Section 6).

## 2. MAINTENANCE PLANNING

Commonly in infrastructural maintenance planning there is one (public) institution responsible for the network on behalf of the network users. This road authority is given the task to maintain a high (i) network quality and (ii) throughput (iii) at low costs (although other objectives are also possible, e.g. environmental concerns, robustness, etc.). To this end, network maintenance has to be performed with minimal nuisance. However, the actual maintenance is performed by several autonomous, independent contractors and therefore some coordination of maintenance activities is required.

The maintenance planning problem discussed in this paper is part of the dynamic contracting procedure introduced in [24]. Here we focus on the execution phase, i.e. the planning and execution of maintenance activities. The assessment, pricing and allocation of maintenance activities is performed in the preceding procurement phase, not discussed in this paper.

In the infrastructural maintenance planning problem we are given a network of roads  $E$ . On this network we have a set  $N$  of agents (the contractors), with each agent  $i \in N$  responsible for the maintenance of a disjoint subset  $E_i \subseteq E$  of roads over a set of discrete periods  $T$ . An edge  $e_k \in E$  has a quality level  $q_{e_k} \in [0, 1]$  and a function  $\hat{q} : q \times T \rightarrow q$  that models the quality degradation of a road given the current state and time (new roads degrade less quickly, seasons influence degradation).

For each edge  $e_k \in E_i$ , an agent  $i$  has a set of possible maintenance activities  $A_k$  that have been identified and assigned in the aforementioned procurement phase. We write  $A_i$  to denote all possible activities by an agent  $i$ , i.e.  $A_i = \cup_{\{k|e_k \in E_i\}} A_k$ . Each of the activities  $k \in A_k$  has a duration  $d_k \in \mathbb{Z}^+$ , a quality impact function  $\Delta q_k : q_{e_k} \times T \rightarrow q_{e_k}$  that depends on the current road

quality and time, and a constant revenue  $w_k \in \mathbb{R}$  that is obtained upon completion of the activity. Moreover, the agent has a (private) cost function  $c_i : A_i \times T \times \mathbb{Z}^+ \rightarrow \mathbb{R}$  that represents the cost of performing an activity  $k \in A_i$  at time  $t \in T$  for a duration  $d \in \mathbb{Z}^+$ . The dependency on time enables modelling of different costs for example for different seasons, or for periods where the agent has fewer resources available.

We model the limited resources (machinery, employees, etc.) available to an agent by allowing at most one activity at a time. This restriction does not have much impact on the model we propose here but does greatly simplify resource reasoning and therefore the complexity of finding optimal maintenance plans.

Each agent strives to plan their maintenance activities in such a way that their profits are maximised, but plan execution is unlikely to be perfect. Uncertainties in various forms – for example delays, unknown asset states, failures, etc. – may be encountered during execution and hence ‘offline optimal’ plans might lead to rather poor results. To this end we focus on contingent plans, or *policies*, that dictate the best action to take *in expectation* for all possible agent states. Note that actions here are operations available to the contractors (e.g. start activity, do nothing, etc.) and states contain all relevant information for its planning problem. We formalise all these concepts in Section 3.1, for now it is sufficient to know that we can always obtain contractor plans by evaluating the most-likely path of a policy (i.e. the sequence of actions that in expectation optimise the contractor reward). We use  $(k, t) \in \pi_i$  to denote that from the policy  $\pi_i$  we can derive that starting activity  $k$  at time  $t$  is expected to yield the highest reward.

Given a policy  $\pi_i$ , the expected profit for agent  $i$  is defined as

$$C_i(\pi_i) = \sum_{(k,t) \in \pi_i} \bar{w}_k - \sum_{(k,t) \in \pi_i} c_i(k, t, d_k) \quad (1)$$

with  $\bar{w}_k$  being  $w_k$  if the activity is completed within the period  $T$  and 0 otherwise. As activity rewards follow directly from the procurement, we assume that agents in expectation are always able to achieve a positive profit for completing their activities, otherwise they would not have bid on the activity during procurement. Note that we do not explicitly require all activities of an agent to be planned or that they can be completed within the period  $T$ , but because agents will not receive rewards  $w_k$  for each uncompleted activity  $k$  they will be stimulated to complete them.

For the agents to also consider the global objectives, we introduce payments such that their profits depend on the delivered quality and additional congestion caused by their presence. The quality payment  $Q_i$  for each agent  $i$  can be both a reward as well as a penalty, depending on the final quality state of its roads (e.g. based on contracted demands). Again given a policy  $\pi_i$ , we can determine the resulting quality state  $q_e^T$  at the end of the period  $T$  using the recursive formula

$$q_{e_k}^{t+1} = \begin{cases} \Delta q_k(q_{e_k}^t, t) & \text{if } (k, t) \in \pi_i \\ \hat{q}_{e_k}(q_{e_k}^t, t) & \text{otherwise} \end{cases} \quad (2)$$

with (given) initial quality  $q_{e_k}^0$ . Consequentially, we define the quality payment for agent  $i$  as a result of his policy by  $Q_i(\pi_i) = \sum_{e \in E_i} Q_i(q_e^T)$ .

Congestion payments, i.e. social costs, cannot be considered from just the single agent perspective because network throughput depends on the planning choices of all agents. Let  $A^t$  denote the set of activities performed by all agents at time  $t$ , then the social cost of this combination is captured by  $\ell(A^t)$ . The impact of an individual agent, given the choices made by others, can be determined

by  $\ell_i(A^t) = \ell(A^t) - \ell(A_{-i}^t)$  in which  $A_{-i}^t$  denotes the set of activities performed at time  $t$  minus any activity by agent  $i$ . The social cost function can for example capture the costs of traffic jams due to maintenance activities. In realistic scenarios where a good estimate of origins and destinations of the traffic flow is known, these costs can be learned empirically.

Recapitulating the above, each agent  $i$  is interested in maximising its expected profit, trivially comprised of only maintenance rewards and costs  $C_i$ . In order to stimulate agents to plan maintenance in favour of global objectives, we introduce quality and throughput payments such that their profit  $u_i$ , given the *joint* policy  $\pi = \bigcup_{i \in N} \pi_i$ , is now given by:

$$u_i(\pi) = C_i(\pi_i) + Q_i(\pi_i) + \ell_i(\pi) \quad (3)$$

in which  $\ell_i(\pi) = \sum_{t \in T} \ell_i(A^t)$ . Here,  $A^t$  can be easily derived from  $\pi$  considering planned start times and activity durations as before.

Now given the utility function of Eq. 3, how should we define these payments such that the right balance is made between these costs and the agents’ private costs, which are not known to the road authority? This is exactly a mechanism design problem. As the scenarios both contain a form of uncertainty and thus dynamics, there are few known mechanisms that can be applied; we are aware of only two (dynamic-VCG [2, 6] and dynamic AGV [1]). Both require an optimal solution (in expectation) of the planning problem, otherwise agents can benefit from (deliberately) misreporting their private costs.

In the next section we therefore start by discussing how to compute optimal solutions to the problem variants introduced in this section, followed by a summary of how this can be combined with a dynamic mechanism.

### 3. BACKGROUND

We briefly introduce the two concepts our work builds on, planning under uncertainty and dynamic mechanism design.

#### 3.1 Planning under Uncertainty

To deal with uncertainties we model the planning problem using Markov Decision Processes (MDPs), which capture this type of uncertainty rather naturally [19]. For each agent  $i \in N$  we have an MDP  $M_i = \langle S_i, \mathcal{A}_i, \tau_i, r_i \rangle$  that defines its local planning problem. In this definition,  $S_i$  is the set of states and  $\mathcal{A}_i$  a set of available actions (see Sections 4.3 and 4.4). The function  $\tau_i : S_i \times \mathcal{A}_i \rightarrow \Delta(S_i)$  describes the transition probabilities where  $\tau_i(s_i, \mathcal{A}_i, s'_i)$  denotes the probability of transitioning to state  $s'$  if the current state is  $s_i$  and action  $\mathcal{A}_i$  is taken. Finally,  $r_i : S_i \times \mathcal{A}_i \rightarrow \mathbb{R}$  is the reward function where  $r_i(s_i, a)$  denotes the reward that the agent will receive when action  $a \in \mathcal{A}_i$  is taken in state  $s_i$  (e.g. the utility of Eq. 3). We formalise the rewards and actions for the agents in Section 4, as they depend on the encoding used to solve the MDP.

Solutions to MDPs are policies  $\pi : S \rightarrow \mathcal{A}$  that dictate the best action to take in expectation, given the current state it is in. Formally, the optimal policy  $\pi^*$  is defined such that for all start states  $s \in S$ :

$$\pi^*(s) = \arg \max_{\pi \in \Pi} V^0(\pi, s) \quad (4)$$

with

$$V^{t_0}(\pi, s) = \mathbb{E} \left[ \sum_{t=t_0}^{\infty} \gamma^t r(s^t, \pi(s^t)) \mid s^{t_0} = s \right] \quad (5)$$

in which  $s^t$  is the state at time  $t$  and  $\gamma \in [0, 1)$  is a shared dis-

count factor commonly used to solve problems with infinite horizons.

We can obtain the individual policies  $\pi_i$  for each agent by solving its local plan problem MDP  $M_i$ . However, in order to develop an (optimal) joint policy  $\pi^*$ , required to consider throughput payments, we need to solve the multi-agent MDP that results from combining all individual MDPs. Formally, the joint MDP is defined by  $M = \langle S, \mathcal{A}, r, \tau \rangle$  where  $S = \times_{i \in N} S_i$  is the joint state space containing in each state  $s \in S$  a local state  $s_i$  for all agents  $i \in N$ ,  $\mathcal{A}$  is the set of combined actions,  $r$  the reward function defined as  $\forall s \in S, a \in \mathcal{A} : r(s, a) = \sum_{i \in N} r_i(s_i, a_i)$  and  $\tau$  the combined transition probability function. The joint action set can always be obtained by including an action for each element of the Cartesian product set of all individual action spaces but smarter construction can greatly reduce the joint action set. For planning problems (at least) we have developed a two-stage MDP encoding that effectively reduces the joint action set size from exponential to linear in the number of players and their action sets. This is discussed in detail in Section 4.2.

### 3.2 Dynamic mechanism design

Although MDPs facilitate modelling uncertainties, they assume global knowledge of all these uncertainties, as well as costs and rewards. As the maintenance activities are performed by different, usually competing companies, we cannot assume that this knowledge is globally available. We therefore aim to design a game such that utility-maximising companies behave in a way that (also) maximises the global reward. This is exactly the field of mechanism design, sometimes referred to as inverse game theory.

Formally, in a static or one-shot game, each agent  $i \in N$  has some private information  $\theta_i$  usually known as its *type*. In so-called direct mechanisms, players are asked for their type, and then a decision is made based on this elicited information. Groves mechanisms [12] take the optimal decision ( $\pi^*$ ) and define payments  $\mathcal{T}$  such that each player's utility is maximised when it declares its type truthfully.

Dynamic mechanisms extend 'static' mechanisms to deal with games in which the outcome of actions is uncertain and private information of players may evolve over time. In each time step  $t$ , players need to determine the best action (in expectation) to take while considering current private information and possible future outcomes. Private rewards are therefore defined depending on the state and the policy, given by  $r_i(s^t, \pi(s^t))$ , in which the state contains the player's type. This type is denoted by  $\theta_i^t$  to express the possibility of this changing over time. With  $\theta^t$  we denote the type of all players at time  $t$  which are encoded in the state  $s^t$ .

An extension of Groves mechanisms for such a dynamic and uncertain setting is dynamic-VCG [2, 6]. For dynamic-VCG the decision policy is the optimal one maximising the reward of all players, identical to Eq. 4 when the types  $\theta^t$  are encoded into the state  $s^t$ . We denote this optimal policy for time step  $t$  given the reported types  $\theta^t$  encoded in state  $s^t$  by  $\pi^*(s^t)$ . A policy optimised for the game with all players except  $i$  is denoted by  $\pi_{-i}^*(s^t)$  and we define  $r_i(s_i^t, \pi_{-i}^*(s_i^t)) = 0$ .

In every time step each player  $i$  pays the expected marginal cost he incurs on other players  $j$  for the current time step. This is defined as the difference between the reward of the other players for the socially optimal decision for the current time step  $t$ , i.e.  $\sum_{j \neq i} r_j(s^t, \pi^*(s^t))$  and their expected reward optimised for just them in future time steps, i.e.  $V^{t+1}(\pi_{-i}^*, s^{t+1})$  (Eq. 5) minus the expected reward of the other players for a policy optimised for them for all time steps including the current one, i.e.  $V^t(\pi_{-i}^*, s^t)$ . Summarising, the payment  $T_i(\theta^t)$  for an agent  $i$  at time step  $t$  given that

reports  $\theta^t$  are encoded in state  $s^t$  is thus defined as

$$T_i(\theta^t) = \sum_{j \neq i} r_j(s^t, \pi^*(s^t)) + V^{t+1}(\pi_{-i}^*, s^{t+1}) - V^t(\pi_{-i}^*, s^t) \quad (6)$$

The dynamic-VCG mechanism yields maximum revenue among all mechanisms that satisfy efficiency, incentive compatibility and individual rationality in within-period, ex-post Nash equilibrium. This means that at all times for each player the sum of its expected reward and its expected payments is never more than when declaring its true type.

## 4. COORDINATING MAINTENANCE PLANNING

In this work we combine existing work on planning under uncertainty and dynamic mechanism design to solve the complex problem of maintenance planning where agents are selfish and execution is uncertain. Using the dynamic VCG mechanism we ensure that agents are truthful in reporting their costs. Then, using these reports to model agent rewards, we apply planning-under-uncertainty techniques to find efficient policies and finally we determine the payments of the mechanism, as discussed in the previous section. Basically this approach can be seen as a series of auctions in which agents bid activities and their associated costs for an assignment of time-slots.

An important condition for the dynamic VCG mechanism is that the chosen policy is optimal. If it is not, the payments are not guaranteed to achieve truthful cost reports and agents may want to deviate. Therefore we focus on exact solving methods in our approach.

We implemented our mechanism using the SPUDD solver [13] to determine optimal policies. The SPUDD solver allows for a very compact but expressive formulation of MDPs in terms of algebraic decision diagrams (ADDs) and uses a structured policy iteration algorithm to maximally exploit this structure. This allows it to find optimal solutions to moderately sized problems. We note, however, that our mechanism is independent of the particular MDP solver used, as long as it returns optimal solutions.

### 4.1 MDP models for maintenance planning

Recall that we want to find an efficient joint policy  $\pi^*$  that maximises the sum of all agent utilities  $u_i$  (Eq. 3). However, this utility function cannot be directly translated into an equivalent MDP encoding. Although in our model  $C$ ,  $Q$  and  $\ell$  can be general functions, encoding general functions in the MDP formulation potentially requires exponential space. Hence to be able to use the SPUDD solver in our experiments, we necessarily restricted ourselves to only linear functions.

The current state of the network, i.e. the quality levels  $q_e$ , are modelled using a 5 star classification (from (0) very bad to (5) excellent) are encoded as discrete variables [0, 5]. Road degradation functions  $\hat{q}$  are modelled using decision diagrams that probabilistically decrease the road quality in each time slot by one state. Completing an activity  $k'$  increases the corresponding road quality  $q'_k$  by a specified number of states (additive), corresponding to its effect  $\Delta q'_k$ .

Encoding the social cost  $\ell$  can be cumbersome, depending on the complexity of the chosen cost model. Again, general cost models could result in exponential MDP encoding sizes. Using only unary and binary rules to express social cost, we can overcome this exponential growth. (at the cost of losing some expressiveness). The unary rules  $l : A \rightarrow \mathbb{R}$  express the marginal latency introduced by each activity independently. Dependencies between activities are expressed using binary relations  $l : A_i \times A_j \rightarrow \mathbb{R}$  that specify the



additional social cost when both activities are planned concurrently. The costs incurred by the set of chosen activities  $A^t$  can be computed using  $\ell(A^t) = \sum_{k \in A^t} \ell(k) + \sum_{k_1 \in A^t} \sum_{k_2 \neq k_1 \in A^t} \ell(k_1, k_2)$ .

## 4.2 Avoiding exponentially-sized action spaces

Factored MDP solvers are typically geared towards exploiting structure in transition and reward models, but scale linearly with the number of actions. In multi-agent problem domains such as ours, however, a naive construction of the joint action set – such as enumerating all elements of the Cartesian product of individual action sets – can be exponential in the number of agents. To overcome this issue, we model each time step in the real world by two stages in the multi-agent MDP, resulting in a larger number of search backups due to additional variables, but crucially avoiding exponentially-sized action spaces. Note that the encoding technique we discuss in this section is not restricted to our problem; they can be applied on any multi-agent decision problem MDP formulation in which agent actions are dependent only through their rewards.

In our MDP encoding we have used a two-stage approach for each time step in the plan problem length  $T$ . In the first step agents decide on the activity to perform (or continue) and this activity is then ‘executed’ in the second stage (illustrated in Sections 4.3 and 4.4 for two example scenarios). We implement this separation through the use of additional variables that for each agent state the activity to perform in the current time step. These variables can be set independent from the actions available to other players (unlike the Cartesian product action space). The second stage then encodes the ‘execution’ of their choices using one additional action. Still there are multiple ways in which this first-stage activity selection can be implemented. Again enumeration is possible (although obliterating the purpose of the two-stage approach) but we have developed two smarter encodings: action chains and activity chains.

The action chain encoding exploits the fact that we can decide on an action for each player sequentially, instead of deciding on them all at once (as with enumeration). Through the use of a player token, each agent gets a ‘turn’ to determine his action within a single time step. Therefore we require only  $|A_i|$  actions for each agent  $i$ , one for each activity it can choose, and hence a total of  $\sum_{i \in N} |A_i|$  states (and one additional variable), instead of the  $\prod_{i \in N} |A_i|$  actions needed for enumerating the Cartesian product.

For activity chains we exploit a similar idea. We group the activities of agents into activity chains to obtain an even smaller set of joint MDP actions. Let  $D = \max_{i \in N} |A_i|$  be the size of the largest activity set of any player, then the activity chains are defined as  $AC_m = \bigcup_{i \in N} k_m \in A_i$  for  $m = 1, 2, \dots, D$ . Hence we group all  $m$ -th activities of each player into set  $AC_m$ . If a player  $i$  has no  $m$ -th activity, i.e.  $m > |A_i|$ , we exclude the player from this activity chain using a high penalty. Again using the player token we enforce that each player sequentially chooses an activity from one of these chains. This encoding requires exactly  $D$  actions in the joint MDP for the first stage and is therefore often more compact than the  $\sum_{i \in N} |A_i|$  actions required for action chains.

In the second stage we model the execution of these choices, i.e. apply maintenance effects, and compute the sum of utilities (Eq. 3) for this time step as the reward. Note that we only proceed in time after the second stage, hence both stages are effectively within one time slot  $t \in T$ .

So far we have introduced a general encoding for maintenance scheduling problems. Now we will go into the specifics for two real-world application we have chosen to study in this paper: one with unit-time activities that may fail, and one where activities always succeed, but possibly have a much longer duration. A summary of the main differences can be found in Table 1.

	repeat	duration	success prob.	delay dur.	delay prob.
		$d_k$	$\alpha_k$	$h_k$	$\beta_k$
1	yes	1	$[0, 1]$	0	0
2	no	$\mathbb{Z}^+$	1	$\mathbb{Z}^+$	$[0, 1]$

Table 1: The differences between scenario 1 and 2

## 4.3 Scenario 1: Activities with failures

As a step towards network maintenance, we first focus on scheduling repeatable unit-time activities with possible failures. Although this problem is conceptually rather simple, it captures essential parts of real-world applications such as factory scheduling and supply chain planning problems. In this scenario, activities  $k \in A_i$  are repeatable, of unit-time ( $d_k = 1$ ) and succeed with probability  $\alpha_k \in [0, 1]$ . It is possible for any activity  $k \in A_i$  to fail with probability  $1 - \alpha_k$ . Whether an activity fails will become apparent at its actual execution time. When an activity fails, it has no positive effect on the quality but its associated maintenance and throughput costs are still charged. If the agent still wants to perform the maintenance it has to include the activity in his plan again at a later time.

Because activities in this scenario are unit-time and repeatable, we can directly translate these into actions of the single-agent MDPs. For each activity  $k \in A_i$  of agent  $i$  we create an action  $a_k$  with reward  $c(k, t, 1)$ . This action improves the quality level  $q_k$  by the number of levels corresponding to  $\Delta q_k$  with probability  $\alpha_k$ . Thus with probability  $1 - \alpha_k$  the maintenance fails and the quality level remains unchanged.

## 4.4 Scenario 2: Portfolio Management

Portfolio management is a second variant of our model. Inspired by real-world consequences of signing a maintenance contract, in this setting agents have to perform each activity exactly once, although multiple alternatives exist for the activity, and instead of activity failure we consider delays. More formally, for each activity  $k$  we now additionally have a delay duration  $h_k$  and delay probability  $\beta_k$ .

Encoding the portfolio management planning in an MDP requires a substantially greater effort as we can no longer translate activities directly to actions. This problem is more complex because of (1) possible non-unit activity durations, (2) activities can be delayed, (3) for each road we can only choose one activity to perform, and (4) each road can be serviced only once. The latter two are easily resolved by introducing a variable that flags whether a road has been serviced and using corresponding penalties to prohibit planning of these activities in a later time; the first two require more work.

From the single agent MDP perspective, non-unit activity durations (including possible delay) do not pose any difficulties. We could use actions that update the time variable  $t$  according to the activity duration. For the joint MDP however, this time variable is shared over all the agents. Increasing the time by the activity duration makes it impossible for other agents to start their activities in this time period.

Our solution is to decompose each activity  $k$  into a series of unit-time MDP actions  $\{start_k, do_k, delay_k, done_k\}$  and use a timer variable to keep track of the remaining activity duration and its delay status (*pending*, *no* or *yes*). The  $start_k$  action marks the beginning of the activity. This action sets the delay status to *pending* and the activity timer to the duration  $d_k$ . In subsequent time steps, the agent has to perform a  $do_k$  action until the activity timer reaches zero. At this point, the activity delay status is pending and the activ-

ity is delayed with probability  $\beta_k$  (also updating the delay status).

If the activity is not delayed, the  $done_k$  action is executed and the associated road  $e_k$  is flagged as serviced. When an activity is delayed however, we set the activity timer to the delay duration  $h_k$  and continue with  $do_k$  actions until again the timer reaches zero, at which point the  $stop_k$  action is executed (not  $delay_k$  again because of the delay status value).

Important to keep in mind is that during the search for optimal policies, a solver might decide on any order of these actions. Hence we need to constrain the actions such that only feasible action sequences are considered. For example, the  $do_k$  action can only be chosen if the activity timer is greater than zero, otherwise a high penalty results.

Rewards are encoded using the two-stage approach as before. In the first stage, each agent chooses a *start*, *do*, *delay* or *stop* action. Then the second stage implements these actions and incurs maintenance, quality and social costs for the current time step  $t$ .

## 4.5 Planning Methods

Using the encodings we discussed, we can find the optimal policy  $\pi^*$  that minimises costs over all three objectives. In the experiments, we then compare this centralised computation that relies on truthful reporting to (1) the approach where each agent plans its own actions optimally individually, i.e. disregarding other agents, and (2) a best-response approach [14].

In the best-response approach, agents alternatingly compute their best plan (in expectation) in response to the current (joint) plan of the others. This approach allows us to solve much easier single agent problems but still consider agent dependencies (e.g. social cost). Of course, the downside of this approach is that we will have to settle for Nash equilibria (if they exist).

## 5. EVALUATION

The combination of planning under uncertainty and mechanism design has been studied in only a handful of papers, their focus mainly on theoretical aspects. Little is known about application of such a combination on practical problems. We have performed a substantial number of experiments to gain insight into this previously uncharted area.

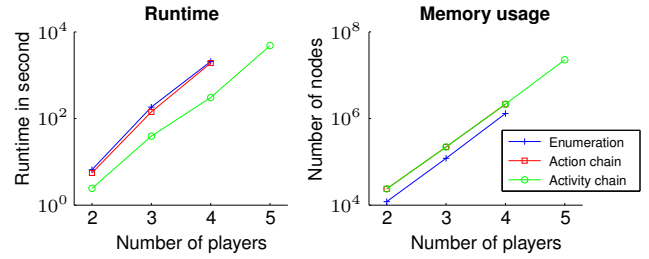
For both problem scenarios we have generated large benchmark sets on which we tested the various planning approaches and their encodings discussed in the previous section. These experiments are mainly of an exploratory nature in which we study the effect of each of the problem variables.

The solver used in these experiments has been implemented in Java, using SPUDD as its internal MDP solver. All experiments have been run on a system with an 1.60 Ghz Intel i7 processor.

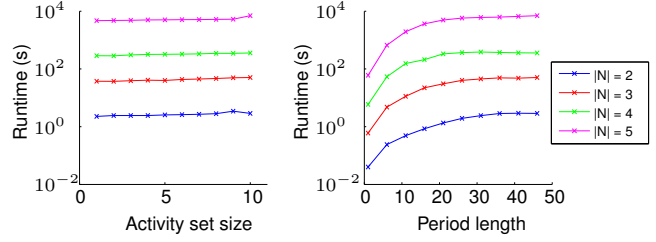
### 5.1 Activities with failures

In the first series of experiments we have been mainly interested in exploring the computational limits of solving the problem centrally using an exact algorithm. To this end we generated a set of simple instances that vary in both the number of players  $N$  (2-5) and activity set  $A_i$  sizes (1-15). We solved these instances using different planning period lengths  $T$  (1-46). From these experiments we identify the parameters that contribute the most to the complexity of the problem.

Activity sets are generated using random, linear, time-dependent cost functions and always increase the quality level of the associated road by one. Quality cost functions are also generated for each road. Road quality is decreasing linearly in the quality with a random factor from [1, 3], which is fixed per road. Recall from Section 4.1 that linearity of this and other cost functions is a restriction not



**Figure 1: Comparison of runtime (left) and memory use (right) for different encoding methods and number of player,  $|A_i| = 3$ ,  $|T| = 46$ ,  $|Q| = 6$  (both log scale).**



**Figure 2: Runtime for different activity set sizes with  $T = 46$  (left) and plan period lengths with  $|A_i| = 10$  (right) using the activity chain encoding (again both log scale).**

imposed by our model but is required to combat a potential exponential MDP encoding size.

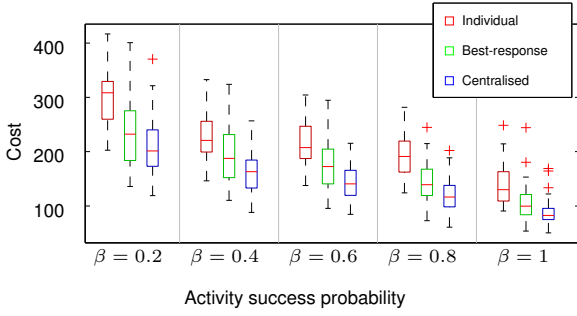
The choice of activity and quality cost functions mostly influence the rewards players can obtain, their impact on the computational complexity is negligible (unless very complex models are used to compute costs). Regarding the social costs  $\ell$  we study the worst-case where all activities always interfere and define these costs using randomly chosen (marginal) cost  $l(k_1, k_2) \in [1, 10]$  for each  $k_1 \in A_i$  and  $k_2 \in A_j$  where  $i \neq j$ . We do not consider the marginal cost for individual actions, i.e.  $l(k) = 0$ .

In Figure 1 we have depicted both the runtime (left) and the memory (right) required to solve each of these instances, under different encoding methods. The memory required is expressed in the number of nodes SPUDG generates.

Not surprisingly this figure illustrates that the performance of the solver is exponential in both time and memory, and greatly depends on the structure of its input. By exploiting the problem structure, the activity chain encoding is able to greatly reduce the required runtime. With it we have been able to solve instances with 5 players and 3 activities per player within the time limit of 3 hours, whereas the other two failed on such instances. Observe that activity chain encoding requires slightly more memory, but this will be of no burden to most modern day computer systems.

For the reasons stated above, we have illustrated the results of the remaining experiments only using the activity chain encoding (which indeed outperformed the others in all tests). In Figure 2 we have plotted the required runtime for solving instances using activity chains for various activity set sizes and period lengths.

From the figure we can conclude that the runtime is only linearly affected by the number of activities each player has. The plan period length shows almost the same: although the required runtime increases rapidly at first, for larger plan horizons the increase is again almost linear. It is expected that instances with small plan lengths are easily solvable because only a small number of plans



**Figure 3: Total cost using different planning approaches for the activities with failure problems (lower is better).**

is possible. Increasing the plan length introduces an exponential number of new possible plans and therefore the computation time increases rapidly, up to the point where the roads reach maximum quality. From this time on, agents have to consider planning an activity only when the quality degrades.

Having identified the computational boundaries of the centralised problem, we compared the performance of different planning approaches discussed in Section 4.5 in terms of total reward obtained. For these experiments we have used 60 generated two-player instances in which each player is responsible for one road.

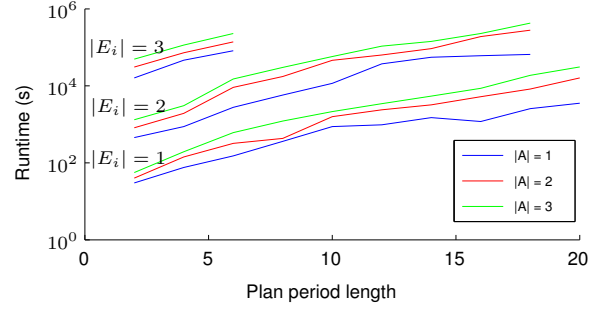
The activity set of each player contains the no-operation and 1, 2 or 3 available maintenance operations that improve the quality of the road by 1, 2 or 3 levels respectively. The cost of each action  $k \in A_i$  is drawn randomly from  $[1, 3 * \Delta q_k]$  and is therefore independent from its execution time.

We have generated 20 instances for each different activity set size and we ran our planning algorithms from Section 4.5. In these experiments all activities have the same success rate  $\alpha$  and we solved all the instances with rates  $[0.2, 0.4, 0.6, 0.8, 1]$ . For the best-response algorithm we have used 3 iterations and in each of these iterations the order of agents is randomly determined. Smaller experiments support our choice for 3 iterations: less iterations result in far worse results while more iterations only slightly improve the quality but increase the runtime substantially. Note that we have no guarantee that the best-response approach will converge to an equilibrium at this point. This is an issue that remains open at this point, however early experiments have shown that best-response almost always improves the initial solution at least.

Figure 3 illustrates the total cost obtained for each of the methods under different levels of uncertainty with a box plot. In the plot, the box contains the upper and lower quartile of the result values with the mean shown by the horizontal line. The whiskers show the smallest and largest values and outliers are plotted as red crosses.

The centralised algorithm always computes the social optimal solution in which the total cost is minimal. As to be expected, the individual planning method perform much worse on these instances. Because in this approach the dependencies between agents are ignored, the resulting plan may suffer from high social cost. Indeed this figure shows that the total costs are much higher on average, compared to the central solution.

Using only 3 iterations, the best-response algorithm produces fairly acceptable plans. As we have mentioned before, best-response can be seen as a compromise between individual and central planning. Indeed our experiments show that the total cost is lower on average than when using individual planning, but higher than the centralised method.



**Figure 4: Runtimes of best-response planning for portfolio management for various road set sizes  $|E_i|$ , activities per road  $|A|$  and plan length  $T$  (log scale).**

## 5.2 Portfolio Management

For portfolio management we have performed similar experiments. We have generated a set of 5 games for each combination of  $|N| \in [2, 5]$ ,  $|E_i| \in [1, 5]$ ,  $|A_i| \in [1, 3]$  and  $\beta \in [0.2, 0.4, 0.6, 0.8, 1.0]$  (delay risk is the same for all activities in these instances). We ran our solver on these instances for different values of  $T$ . Again we study the worst case in which players are tightly coupled (all activities interfere with at least one of another agent), and we strive to gain insight in the factors contributing to the complexity.

Although exact solving for multiple agents poses a difficult challenge, we have been able to solve several non-trivial instances using the best-response approach. Figure 4 shows the runtime required for several of these instances. These early experiments show that best-responses can be computed in the order of a few minutes for problems where agents are responsible for multiple roads with several activities to choose from.

## 6. CONCLUSIONS AND CHALLENGES

This paper introduces the practically very relevant problem of infrastructural maintenance planning under uncertainty for selfish agents in a private-values setting. With the help of experts in the field of asset management we developed a model that captures the essence of this coordination problem. Dynamic mechanism design combined with optimally solving MDPs theoretically solves this modelled problem. Through experimental analysis with different encodings in an existing solver, we found that run times become large for more than 5 players, 3 activities, 6 quality levels and 46 time steps (for scenario 1), which is reasonable in practice. For scenario 2, run times for best-response can be computed for multiple agents in a small network. We have thus made an important step towards this practical planning problem, but, additionally we can identify a number of challenges for our community.

In this paper, we used scalar weighting to balance the different objectives in the system. However, asset maintenance planning for infrastructures is inherently a multi-objective problem, even though this has not been acknowledged in procurements until recently. The weighting model has two difficulties. Firstly, it requires accurate and exhaustive operationalisation of objectives in terms of monetary rewards schemes. Secondly, in any practical application, human decision makers are more likely to prefer insight into possible solutions trade-offs over a single black-box solution. In this context, the work by [11] is relevant, in which the authors study approximation techniques for mechanism design on multi-objective problems. Nevertheless, their work has only been applied to static mechanisms. Developing methods combining multi-objective plan-

ning under uncertainty with dynamic mechanism design is a hard challenge for the community, but with high potential payoffs in terms of real-world relevance.

Scaling MDP solvers in terms of number of actions has not received large amounts of attention, but is crucial for solving multi-agent problems that suffer from exponential blow up of their action space. Furthermore, the best-response approach that we employed is not guaranteed to converge to the optimal solution, except for special cases such as potential games [14]. Bounding the loss, e.g., by building on those special cases, will provide benefits to the adoption of best-response methods.

Finally, as mentioned in the related work section, approximate solutions often preclude many of the theoretical mechanism-design results to apply. A major challenge here is to identify mechanisms that are more robust to such approximations.

With respect to the implications of our work, it is clear that the planning and coordination of (maintenance) activities in the presence of uncertainty is a complex problem. However, applications exist in several other domains such as bandwidth allocation or smart power grids, and hence the need for a practical solution is high.

The concept of traffic time loss can also be used to stimulate market parties in rethinking current working methods. By adjusting tendering criteria to specific needs on certain areas of the network, bidders can distinguish themselves by offering innovative proposals with limited traffic loss hours. The road authority and several provinces are currently experimenting with this method.

## Acknowledgements

This research is part of the Dynamic Contracting in Infrastructures project and is supported by Next Generation Infrastructures and Almende BV. Matthijs Spaan is funded by the FP7 Marie Curie Actions Individual Fellowship #275217 (FP7-PEOPLE-2010-IEF).

## 7. REFERENCES

- [1] S. Athey and I. Segal. An efficient dynamic mechanism. Technical report, UCLA Department of Economics, 2007.
- [2] D. Bergemann and J. Valimaki. Efficient dynamic auctions. *Cowles Foundation Discussion Papers*, 2006.
- [3] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.
- [4] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proc. of 6th Conf. on Theoretical Aspects of Rationality and Knowledge*, pages 195–201, 1996.
- [5] R. I. Brafman, C. Domshlak, Y. Engel, and M. Tennenholtz. Planning games. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pages 73–78, 2009.
- [6] R. Cavallo. Efficiency and redistribution in dynamic mechanism design. In *Proc. of 9th ACM conference on Electronic commerce*, pages 220–229. ACM, 2008.
- [7] R. Cavallo, D. C. Parkes, and S. Singh. Optimal coordinated planning amongst self-interested agents with private state. In *Proc. of Conf. on Uncertainty in Artificial Intelligence*, pages 55–62, 2006.
- [8] C. d’Aspremont and L. Gérard-Varet. Incentives and incomplete information. *Journal of Public Economics*, 11(1):25–45, 1979.
- [9] Der Spiegel. A40: Autobahn nach dreimonatiger sperre freigegeben, 2012. Online, Sep 30.
- [10] B. Detienne, S. Dauzère-Pérès, and C. Yugma. Scheduling jobs on parallel machines to minimize a regular step total cost function. *Journal of Scheduling*, pages 1–16, 2009.
- [11] F. Grandoni, P. Krysta, S. Leonardi, and C. Ventre. Utilitarian mechanism design for multi-objective optimization. In *Proc. of 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–584. Society for Industrial and Applied Mathematics, 2010.
- [12] T. Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973.
- [13] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. In *Proc. of Conf. on Uncertainty in Artificial Intelligence*, pages 279–288, 1999.
- [14] A. Jonsson and M. Rovatsos. Scaling up multiagent planning: A best-response approach. In *Int. Conf. on Automated Planning and Scheduling*, pages 114–121, 2011.
- [15] J. R. Kok, P. Hoen, B. Bakker, and N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *Proc. Symp. on Computational Intelligence and Games*, pages 29–36, 2005.
- [16] F. S. Melo and M. Veloso. Learning of coordination: Exploiting sparse interactions in multiagent systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 773–780. International Foundation for Autonomous Agents and Multiagent Systems, 2009.
- [17] R. Porter, A. Ronen, Y. Shoham, and M. Tennenholtz. Fault tolerant mechanism design. *Artificial Intelligence*, 172(15):1783–1799, 2008.
- [18] D. Procaccia and M. Tennenholtz. Approximate mechanism design without money. In *Proc. of ACM Conf. on Electronic Commerce*, pages 177–186, 2009.
- [19] M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.
- [20] J. Scharpff, M. T. J. Spaan, L. Volker, and M. M. de Weerd. Planning under Uncertainty for Coordinating Infrastructural Maintenance. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2013. To appear.
- [21] S. Seuken, R. Cavallo, and D. Parkes. Partially-synchronized DEC-MDPs in dynamic mechanism design. In *Proc. of National Conf. on Artificial Intelligence*, pages 162–169, 2008.
- [22] J. Snellen, D. Ettema, and M. Dijst. Activiteitenpatronen en reistijdwaardering (dutch). Technical report, Transumo, December 2007.
- [23] R. P. van der Krogt, M. M. de Weerd, and Y. Zhang. Of mechanism design and multiagent planning. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *European Conf. on Artificial Intelligence*, pages 423–427, 2008.
- [24] L. Volker, J. Scharpff, M. De Weerd, and P. Herder. Designing a dynamic network based approach for asset management activities. In *Proc. of 28th Annual Conference of Association of Researchers in Construction Management (ARCOM)*, 2012.
- [25] D. H. Wolpert, K. Tumer, and J. Frank. Using collective intelligence to route internet traffic. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 952–958. MIT Press, 1999.

# Asynchronous Execution in Multiagent POMDPs: Reasoning over Partially-Observable Events

João V. Messias  
Institute for Systems and  
Robotics  
Instituto Superior Técnico  
Lisbon, Portugal  
jmessias@isr.ist.utl.pt

Matthijs T. J. Spaan  
Delft University of Technology  
Delft, The Netherlands  
m.t.j.spaan@tudelft.nl

Pedro U. Lima  
Institute for Systems and  
Robotics  
Instituto Superior Técnico  
Lisbon, Portugal  
pal@isr.ist.utl.pt

## ABSTRACT

This paper proposes a novel modeling approach to problems of multiagent decision-making under partial observability, building on the framework of Multiagent Partial Observable Markov Decision Processes (MPOMDPs). Unfortunately, the size of MPOMDP models (and their solutions) grows exponentially in the number of agents, and agents are required to act in synchrony. In the present work, we show how these problems can be mitigated through an event-driven, asynchronous formulation of the MPOMDP dynamics.

We introduce the necessary extensions to the dynamics and solution algorithms of standard MPOMDPs. In particular, we prove that the optimal value function in our Event-Driven Multiagent POMDP framework is piecewise linear and convex, allowing us to extend a standard point-based solver to the event-driven setting. Finally, we present simulation results, showing the computational savings of our modeling approach.

## INTRODUCTION

Many multiagent applications take place in stochastic domains, in which the interaction of each agent with its environment carries a measure of uncertainty with respect to its outcome. In particular, the information that each agent can extract from that environment may be limited or noisy. Examples can be found in sensor networks [7], cooperative robotics [15], and distributed manufacturing systems [12].

Most existing approaches to multiagent decision making under uncertainty are grounded in the theory of Markov Decision Processes (MDPs), for example Decentralized Partially Observable MDPs (Dec-POMDPs) [2] which are intractable without communication (NEXP-Complete); and Multiagent MDPs (MMDPs) / POMDPs (MPOMDPs), in which free communication between agents is assumed [3, 11]. In the latter case, the planning problem is of equivalent complexity to that of solving a centralized single-agent problem defined over the whole team [11]. We focus on situations in which it is reasonable to assume that agents can communicate freely, but the system is partially observable, even given the observations of all agents. This is typical, for example, when dealing with teams of mobile robots, or in autonomous surveillance systems. We will therefore rely on

the MPOMDP paradigm.

Modeling a multiagent problem as an MPOMDP carries a notable drawback: the complexity of solving an MPOMDP model, even approximately, is exponential in the number of agents. This undesirable property, however, follows from the implicit assumption that agents are performing actions and observing their outcomes *in synchrony*. That is to say, at each decision step, each agent is expected to simultaneously perform individual local actions, and to receive simultaneous individual observation symbols.

The present work proposes Event-Driven MPOMDPs, an alternative description of the dynamics of multiagent decision making under uncertainty, based on the operation of real-time discrete-event systems [5]<sup>1</sup>. In our approach, agents must react to events, which are detected locally, and *asynchronously*, by each agent. Through the assumption of free communication, each local event triggers a joint observation, which is shared by the team. Since multiple events cannot occur simultaneously, this means that the total number of joint observations in this model grows *linearly* in the number of agents (instead of exponentially), allowing these methods to scale better to larger scenarios, while retaining MPOMDP functionality.

Furthermore, the processes through which events are detected are considered to be susceptible to error. An agent may signal the detection of an event which did not actually take place (a false positive), or fail to detect a real event (a false negative). It will be shown that the latter case, in particular, implies minimal modifications with respect to the dynamics of a standard POMDP. We prove that the optimal value function is piecewise linear and convex, allowing us to extend a point-based solver to the event-driven setting. Lastly, we consolidate the proposed methods through simulated results, comparing the performance of our event-driven models to that of equivalent synchronous MPOMDPs.

## BACKGROUND

This section introduces the necessary background regarding MPOMDPs and their solution.

**DEFINITION 1.** *A Multiagent Partially Observable Markov Decision Process (MPOMDP) is a tuple  $\langle d, S, A, \mathcal{O}, T, O, R \rangle$ , where:*

<sup>1</sup>Our definition and use of “events” differs from existing work [1], and concerns different purposes. There, events model interdependencies between agent policies in Dec-MDPs. Here, events are simply state changes: the system dynamics are driven by events.

$d$  is the number of agents;

$\mathcal{S} = \mathcal{X}_1 \times \mathcal{X}_2 \times \dots \times \mathcal{X}_k$  is the state space, a discrete set of possibilities for the state  $s$  of the process. The (joint) state  $s \in \mathcal{S}$  is a tuple of state factor assignments:  $s = \langle x_1, x_2, \dots, x_k \rangle$ , where  $x_i \in \mathcal{X}_i$ ;

$\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_d$  is a set of joint actions. Each joint action  $\mathbf{a} \in \mathcal{A}$  is a tuple of individual actions:  $\mathbf{a} = \langle a_1, a_2, \dots, a_d \rangle$ , where  $a_i \in \mathcal{A}_i$ ;

$\mathcal{O} = \mathcal{O}_1 \times \mathcal{O}_2 \times \dots \times \mathcal{O}_d$  is the space of joint observations  $\mathbf{o} = \langle o_1, \dots, o_d \rangle$ , where  $o_i \in \mathcal{O}_i$  are the individual observations of each agent.

$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  is the transition function, such that  $T(s, \mathbf{a}, s') = \Pr(s' | s, \mathbf{a})$ ;

$O : \mathcal{A} \times \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$  is the observation function, such that  $O(\mathbf{a}, s', \mathbf{o}) = \Pr(\mathbf{o} | \mathbf{a}, s')$ ;

$R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the instantaneous reward function. The value  $R(s, \mathbf{a})$  represents the “reward” that the team of agents receives for performing joint action  $\mathbf{a}$  in state  $s$ ;

The state of the system cannot be observed directly in an (M)POMDP, and therefore agents only have access to probability distributions over the state space, also known as belief states,  $b \in \Pi(\mathcal{S})$ . The belief state at time step  $t$ ,  $b|_t$ , can be computed using Bayes’ rule given the initial belief  $b|_0$  and the sequence of actions taken and observations received. The goal of the planning problem is to maximize the expected discounted reward of a system over a given number of decisions (or planning horizon)  $h$ . Specifically, we seek to obtain a sequence of decision rules  $\pi = \{\pi_{h-1}, \dots, \pi_0\}$ , where  $\pi_i : \Pi(\mathcal{S}) \rightarrow \mathcal{A}$ , which maximizes the quantity:

$$V_{h-1}^\pi(b) = E \left[ \sum_{t=0}^{h-1} \gamma^t \sum_{s \in \mathcal{S}} b|_t(s) R(s, \pi_{h-1-t}(b|_t)) \mid b = b|_0 \right], \quad (1)$$

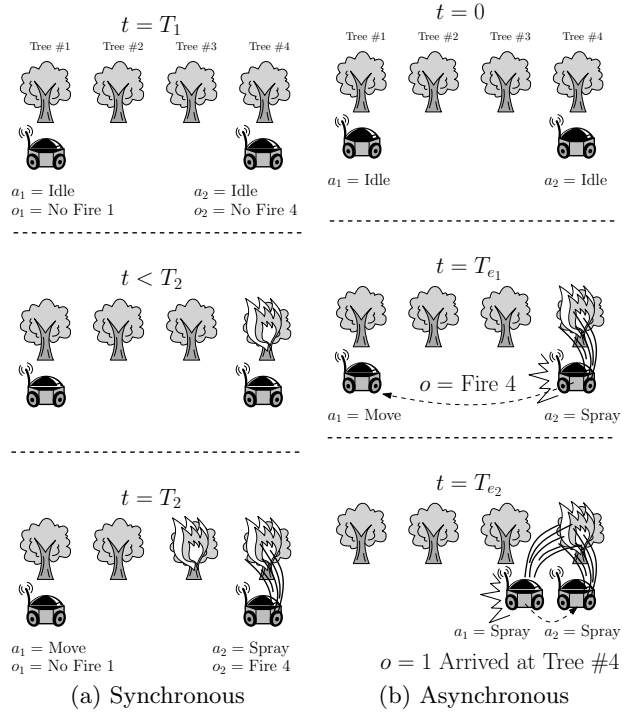
where  $\gamma \in [0, 1]$  is a specified discount factor. The function  $V_{h-1}^\pi(b)$  is also known as a *value function* for horizon  $h$ , under the policy  $\pi$ . We will restrict our attention to optimal value functions,  $V_n^*(b)$ , for  $n = h-1, \dots, 0$ . It is well known that POMDP value functions are piecewise linear and convex (PWLC) for  $h < \infty$ , and can be arbitrarily well approximated by a PWLC function if  $h = \infty$  [13]. Therefore, they admit a compact representation:

$$V_n^*(b) = \max_{v_n \in \Upsilon_n} v_n \cdot b, \quad (2)$$

where  $\Upsilon_n$  is a set of  $|\mathcal{S}|$ -dimensional vectors. A related concept is that of  $Q$ -value functions:  $V_n^* = \max_{\mathbf{a} \in \mathcal{A}} Q_n^*(b, \mathbf{a})$ . These functions represent the expected value of performing joint action  $\mathbf{a}$  when there are  $n$  remaining decisions, and following the optimal policy afterwards [10].

## EVENT-DRIVEN MPOMDPS

In this section we propose a novel modeling approach to multiagent decision making under partial observability. We overview the multiagent dynamics of typical MPOMDPs, how they affect the potential applications of the framework, and how these limitations are addressed through an event-driven perspective. We then formalize our proposed model.



**Figure 1: A graphical representation of the dynamics of a synchronous MPOMDP (a) and of an Event-Driven MPOMDP (b). Two agents must extinguish forest fires. In (a), agents cannot react instantaneously, for example, when a fire breaks out between instants  $t = T_1$  and  $t = T_2$ ; In (b), at instants  $t = T_{e_1}$  and  $t = T_{e_2}$ , agents 1 and 2 respectively detect events, triggering new decision episodes for the team.**

## Synchronous vs. Asynchronous Execution

From a planning perspective, the dynamics of a multiagent POMDP are a trivial extension of those of a single-agent model: agents select an individual action based on a joint belief state,  $b$ , according to a joint policy,  $\pi(b)$ . In practice, however, this implies that agents must only communicate their observations at the *end of each decision episode*, so as not to exclude any potential information. Since, in most cases, no agent can individually determine if a decision episode has finished or not the team must operate *synchronously*: taking actions and sharing observations at a fixed rate, or at predefined time instants. However, this means that agents can no longer react immediately to sudden changes in the state, which, in some cases, can be vital to the outcome of their plan. Furthermore, there are exponentially many possible combinations of individual observations which can be shared amongst the team, which is an evident computational drawback to any planning algorithm which iterates over possible joint observations. The synchronous dynamics of an MPOMDP are illustrated in Figure 1a.

We propose an alternate approach, represented in Figure 1b, that is arguably more intuitive: team decisions are triggered by changes in the state of the system. These changes may be detected by any agent, in the form of a characteristic observation, and are promptly communicated to the rest of the team. The detecting agent can select an appropriate action immediately, and other agents may do so as soon as

they receive that information. In addition to preserving responsiveness, this approach implies that, as a function of the number of agents, there are linearly many possible observations to be considered, corresponding to the sum of the possible detections of each of them. This forms the core of our approach.

## Stochastic, Partially Observable Events

Before formally defining a model which operates according to the requirements formulated above, we need to have a rigorous definition of what constitutes an “event”.

**DEFINITION 2.** A set  $\mathcal{E}$  is a set of events of an MPOMDP if and only if there is a surjective mapping  $\Phi : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{E}$  such that if  $\Phi(s_1, s'_1) = \Phi(s_2, s'_2)$ , then  $\Pr(s'_1 | s_1, \mathbf{a}) = \Pr(s'_2 | s_2, \mathbf{a})$  and  $\Pr(\mathbf{o} | s_1, \mathbf{a}, s'_1) = \Pr(\mathbf{o} | s_2, \mathbf{a}, s'_2)$ , for every  $s_1, s'_1 \in \mathcal{S}, s_2 \neq s_1, s'_2 \neq s'_1 \in \mathcal{S}, \mathbf{a} \in \mathcal{A}, \mathbf{o} \in \mathcal{O}$ .

An event  $e \in \mathcal{E}$  is said to be enabled at  $\langle s, \mathbf{a} \rangle$  if, for  $s'$  such that  $e = \Phi(s, s')$ ,  $\Pr(s' | s, \mathbf{a}) > 0$ . The set of all enabled events in these conditions will be represented as  $E(s, \mathbf{a})$ .

Through this definition, events are seen as abstractions to the state transitions  $\langle s, s' \rangle$  which share the same stochastic properties. Their probability of occurrence can be influenced locally by the actions of each agent, or cooperatively through joint actions. In fact, since trivially  $\Pr(s, s' | s, \mathbf{a}) = \Pr(s' | s, \mathbf{a})$ , we can also represent the transition function as  $T(s, \mathbf{a}, e)$  with  $e = \Phi(s, s')$ . Note that, in contrast to standard POMDP models, we include both  $s$  and  $s'$  in  $O$  as we are interested in observing characteristics of state transitions, as opposed to characteristics of states. As such, we have  $O(s, \mathbf{a}, s', \mathbf{o}) = \Pr(\mathbf{o} | s, \mathbf{a}, s')$ , allowing the observation model to be indexed through events, as  $O(\mathbf{a}, e, \mathbf{o})$ .

Noisy event detection processes are typically characterized through their susceptibility to false positive and false negative errors. However, in a system where decision episodes are driven by detected events, the latter case raises a fundamental problem – if all agents fail to detect the occurrence of an event, agents will not be able to change their actions. This has evident implications for the correctness of the expected value which is calculated during planning and, as such, will be explicitly taken into account.

## A Model for Event-Driven MPOMDPs

We can now formally introduce our modeling approach:

**DEFINITION 3.** An Event-Driven Multiagent POMDP is a tuple  $\langle d, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, T, O, \mathcal{C}, R \rangle$  where:

$d, \mathcal{S}, \mathcal{A}, R$  are defined as in an MPOMDP (Definition 1);

$\mathcal{O}$  is a set of observations  $o$  which can be generated by the environment upon the occurrence of a state transition. Observations are shared by agents;

$\mathcal{E}$  is a set of events (Definition 2);

$T : \mathcal{S} \times \mathcal{A} \times \mathcal{E} \rightarrow [0, 1]$  is the transition function, such that  $T(s, \mathbf{a}, e) = \Pr(e | s, \mathbf{a})$  for  $e \in \mathcal{E}, s \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$ ;

$O : \mathcal{A} \times \mathcal{E} \times \mathcal{O} \rightarrow [0, 1]$  is the observation function, such that  $O(\mathbf{a}, e, o) = \Pr(o | \mathbf{a}, e)$  for  $o \in \mathcal{O}, e \in \mathcal{E}, \mathbf{a} \in \mathcal{A}$ ;

$\mathcal{C} : \mathcal{A} \times \mathcal{O} \rightarrow PS(\mathcal{A}) \setminus \emptyset$ , where  $PS(\mathcal{A})$  is the power set of  $\mathcal{A}$ , is a constraint-generating function which returns, for each pair  $\langle \mathbf{a}, o \rangle$ , a constrained action set  $\mathcal{C}(\mathbf{a}, o) \subseteq \mathcal{A}$ . This set represents the joint actions which are available to the agents at the onset of a decision episode, given that, at the previous step, the team of agents executed  $\mathbf{a}$  and observed  $o$ .

The presence of the constraint function  $\mathcal{C}$  in this definition addresses the problem of unobservable events raised before. In particular, during planning, we can model the occurrence of such events through token observations  $f$  such that  $\mathcal{C}(\mathbf{a}, f) = \mathbf{a}$ . These observations are never received by an agent during plan execution (hence why they are simply *tokens*), but they force planning algorithms to select the same action across different time steps, to account for the fact that agents will not be able to observe the events associated to  $f$  and change their actions accordingly.

## SOLVING EVENT-DRIVEN MPOMDPs

Having defined a framework for Event-Driven MPOMDPs, we will show in this section that these models retain the necessary properties that allow them to be solved through dynamic programming approaches. We will then present an example of how to modify a typical POMDP solution algorithm to allow it to provide (optimal or approximate) policies for our proposed framework.

## Dynamic Programming

We can show that a value function for an Event-Driven MPOMDP in the presence of action constraints is still PWLC, which enables the use of dynamic programming techniques to calculate (or approximate) an optimal policy.

**THEOREM 1.** For an Event-Driven MPOMDP, and for finite  $n$ , the optimal value function  $V_n^*$  can be written as:

$$V_n^*(b) = \max_{v_n \in \Upsilon_n} v_n \cdot b, \quad ,$$

where  $\Upsilon_n$  is a set of  $|\mathcal{S}|$ -dimensional vectors.

**PROOF.** Given a constraint-generating function  $\mathcal{C}$ , any policy in an Event-Driven MPOMDP is subject to the following restrictions:

$$\begin{cases} \pi_n(b|_n) \in \mathcal{A} & \text{if } n = h \\ \pi_n(b|_n) \in \mathcal{C}(\pi_{n+1}(b|_{n+1}), o|_{n+1}) & \text{if } n < h \end{cases},$$

where  $b|_n$  and  $o|_n$  are, respectively, the belief state and the observation received when there are  $n$  steps remaining. Taking these restrictions into consideration, the Bellman backup for this model can be written as:

$$V_n^*(b) = \max_{\mathbf{a} \in \mathcal{A}} \left\{ \sum_{s \in \mathcal{S}} b(s) R(s, \mathbf{a}) + \gamma \sum_{\substack{s \in \mathcal{S}, o \in \mathcal{O} \\ e \in E(s, \mathbf{a})}} b(s) O(\mathbf{a}, e, o) T(s, \mathbf{a}, e) \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a}, o)} Q_{n-1}^*(b_o^{\mathbf{a}}, \mathbf{a}') \right\}, \quad (3)$$

where  $b_o^{\mathbf{a}}$  is the updated belief state according to  $\langle \mathbf{a}, o \rangle$ .

For  $\mathbf{a} \in \mathcal{A}, o \in \mathcal{O}$ , let  $H_o^{\mathbf{a}}$  be a  $|\mathcal{S}| \times |\mathcal{S}|$  matrix with

$$\begin{aligned} [H_o^{\mathbf{a}}]_{i,j} &= \Pr(s_i | s_j, \mathbf{a}) \Pr(o | s_j, \mathbf{a}, s_i) \\ &= T(s_j, \mathbf{a}, \Phi(s_j, s_i)) O(\mathbf{a}, \Phi(s_j, s_i), o) \end{aligned}$$

The Bellman backup is then, in vectorial form:

$$V_n^*(b) = \max_{\mathbf{a} \in \mathcal{A}} \left\{ r^{\mathbf{a}} \cdot b + \gamma \sum_{o \in \mathcal{O}} \mathbf{1}^T H_o^{\mathbf{a}} b \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a}, o)} Q_{n-1}^*(b_o^{\mathbf{a}}, \mathbf{a}') \right\}, \quad (4)$$



where  $r^{\mathbf{a}}$  denotes the  $\mathbf{a}$ -th column of  $R(s, \mathbf{a})$ . Also in this notation, the belief update step is:

$$b_o^{\mathbf{a}} = \frac{H_o^{\mathbf{a}} b}{\mathbf{1}^T H_o^{\mathbf{a}} b} \quad , \quad (5)$$

where  $\mathbf{1}_i = 1$ ,  $i = \{1 \dots |\mathcal{S}|\}$ .

At the final decision step,  $n = 0$ , we have that:

$$V_0^*(b) = \max_{\mathbf{a} \in \mathcal{A}} r^{\mathbf{a}} \cdot b, \quad (6)$$

And therefore  $V_0^*$  is clearly PWLC with  $\Upsilon_0 = \{r^{\mathbf{a}} | \mathbf{a} \in \mathcal{A}\}$ . Inductively, at  $n - 1$  steps-to-go:

$$V_{n-1}^*(b_o^{\mathbf{a}}) = \max_{v_{n-1} \in \Upsilon_{n-1}} v_{n-1} \cdot b_o^{\mathbf{a}} \quad .$$

Also, since  $V_{n-1}^*(b_o^{\mathbf{a}})$  is PWLC iff  $Q_{n-1}^*(b_o^{\mathbf{a}}, \mathbf{a}')$  is PWLC:

$$Q_{n-1}^*(b_o^{\mathbf{a}}, \mathbf{a}') = \max_{q_{n-1}^{\mathbf{a}'} \in \mathcal{K}_{n-1}^{\mathbf{a}'}} q_{n-1}^{\mathbf{a}'} \cdot b_o^{\mathbf{a}} \quad , \quad (7)$$

where  $\mathcal{K}_n^{\mathbf{a}}$  is a set of  $|\mathcal{S}|$ -dimensional vectors. Taking this form for the  $Q$ -value, and substituting (5):

$$Q_{n-1}^*(b_o^{\mathbf{a}}, \mathbf{a}') = \max_{q_{n-1}^{\mathbf{a}'} \in \mathcal{K}_{n-1}^{\mathbf{a}'}} \frac{(q_{n-1}^{\mathbf{a}'})^T H_o^{\mathbf{a}} b}{\mathbf{1}^T H_o^{\mathbf{a}} b} \quad .$$

Let

$$q_{n-1}^{*, \mathbf{a}'} | b, \mathbf{a}, o = \arg \max_{q_{n-1}^{\mathbf{a}'} \in \mathcal{K}_{n-1}^{\mathbf{a}'}} \left\{ (q_{n-1}^{\mathbf{a}'})^T H_o^{\mathbf{a}} b \right\} \quad . \quad (8)$$

Then,

$$Q_{n-1}^*(b_o^{\mathbf{a}}, \mathbf{a}') = \frac{(q_{n-1}^{*, \mathbf{a}'})^T H_o^{\mathbf{a}} b}{\mathbf{1}^T H_o^{\mathbf{a}} b} \quad .$$

Returning to (4), and reorganizing and simplifying terms:

$$\begin{aligned} V_n^*(b) &= \max_{\mathbf{a} \in \mathcal{A}} \left\{ \left( r^{\mathbf{a}} \cdot b + \gamma \sum_{o \in \mathcal{O}} \mathbf{1}^T H_o^{\mathbf{a}} b \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a}, o)} \frac{(q_{n-1}^{*, \mathbf{a}'})^T H_o^{\mathbf{a}} b}{\mathbf{1}^T H_o^{\mathbf{a}} b} \right) \right\} \\ &= \max_{\mathbf{a} \in \mathcal{A}} \left\{ \left( r^{\mathbf{a}} \cdot b + \gamma \sum_{o \in \mathcal{O}} \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a}, o)} (q_{n-1}^{*, \mathbf{a}'})^T H_o^{\mathbf{a}} b \right) \right\} \end{aligned} \quad (9)$$

Therefore,  $V_n^*(b) = \max_{v_n \in \Upsilon_n} v_n \cdot b$ , with

$$v_n = r^{\mathbf{a}} + \left( \gamma \sum_{o \in \mathcal{O}} \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a}, o)} (q_{n-1}^{*, \mathbf{a}'})^T H_o^{\mathbf{a}} \right)^T \quad . \quad (10)$$

It should be noted that each vector  $v$  is associated with a particular action  $\mathbf{a}$ . This concludes the proof that Event-Driven MPOMDPs have PWLC optimal value functions.  $\square$

Next, we will show how this result can be used by most current POMDP planning algorithms, with minor modifications, for Event-Driven MPOMDPs.

## A Randomized Point-Based Algorithm

We now turn our attention to the problem of calculating approximately optimal policies for Event-Driven MPOMDPs. We here focus explicitly on *approximately* optimal policies for computational reasons. However, that does not preclude the possibility of adapting optimal algorithms as well.

A particularly efficient family of approximate POMDP solvers is that of point-based algorithms [8, 9, 14, 6]. We propose an adaptation of the PERSEUS randomized point-based

algorithm that can handle Event-Driven MPOMDPs. The basic premise of any point-based algorithm is that, given a belief state  $b \in \Pi(\mathcal{S})$ , and a value function (or set of  $Q$ -value functions) at stage  $n - 1$ , it is possible to obtain the stage- $n$  maximizing vector at that point at a relatively low computational cost.

We are therefore interested in obtaining:

$$q_n^{\mathbf{a}, b} = \arg \max_{q_n^{\mathbf{a}} \in \mathcal{K}_n^{\mathbf{a}}} q_n^{\mathbf{a}} \cdot b \quad (11)$$

From (10), we have that:

$$q_n^{\mathbf{a}} = r^{\mathbf{a}} + \left( \gamma \sum_{o \in \mathcal{O}} \max_{\mathbf{a}' \in \mathcal{C}(\mathbf{a}, o)} (q_{n-1}^{*, \mathbf{a}'})^T H_o^{\mathbf{a}} \right)^T \quad . \quad (12)$$

Note that this already implicitly defines an optimal  $q_{n-1}^{*, \mathbf{a}'}$  at a given point  $b$  for a particular  $\langle \mathbf{a}, o \rangle$  pair, see Eq. (8). If, instead of taking the maximum, we evaluate the expected future reward for each vector in  $\mathcal{K}_{n-1}^{\mathbf{a}'}$ , and for a given  $\langle \mathbf{a}, o \rangle$ :

$$q_{n, o, \mathbf{a}}^{k, \mathbf{a}'} = \gamma \left( (q_{n-1}^{k, \mathbf{a}'})^T H_o^{\mathbf{a}} \right)^T \quad . \quad (13)$$

Taking the action constraints into consideration, we can select from these vectors the best at  $b$ :

$$q_{n, o}^{\mathbf{a}, b} = \arg \max_{q_{o, \mathbf{a}}^{k, \mathbf{a}'} | \mathbf{a}' \in \mathcal{C}(\mathbf{a}, o)} q_{o, \mathbf{a}}^{k, \mathbf{a}'} \cdot b \quad . \quad (14)$$

And finally, summing over observations and adding the immediate reward:

$$q_n^{\mathbf{a}, b} = r^{\mathbf{a}} + \gamma \sum_{o \in \mathcal{O}} q_{n, o}^{\mathbf{a}, b} \quad . \quad (15)$$

Equipped with this result, we can formulate our variant of the PERSEUS algorithm, which we refer to as *Constraint-Compliant PERSEUS* (CC-PERSEUS). The belief-backup (15) is applied to a subset of belief points in a sampled set  $\mathcal{B}$  [14], for all possible actions. The resulting set of vectors for each action is taken as an approximation of  $Q_{n+1}^{*, \mathbf{a}}$ , and their union as  $V_{n+1}^*$ . Our explicit use of the  $Q$ -value functions stems from the fact that the sets  $\mathcal{K}_n^{\mathbf{a}}$  (c.f. (7)) must never be empty. Otherwise, if an action had no previous-stage vectors associated to it, we would not have an estimate of its expected value when “forced” by  $\mathcal{C}$ .

A note on complexity: since this algorithm is keeping track of all  $Q$ -value functions, in the worst case, it has to perform  $|\mathcal{A}|$  times as many evaluations over the set  $\mathcal{B}$  as the standard version of PERSEUS. It is expected, then, that it should underperform PERSEUS when running over the same model. However, recall that the main advantage of our formulation is that it allows considerably smaller representations (particularly in  $|\mathcal{O}|$ ) of the same problem.

Finally, we emphasize that the considerations made in this section could be applied to virtually any POMDP solution algorithm. The only requirements are that:  $Q$ -value functions must be maintained for all actions (so value functions can only be pruned action-wise); and the constraints generated by  $\mathcal{C}$  should be satisfied when performing backups.

## Execution-Time Belief Updates

In a standard POMDP, a belief state  $b$  can be updated by an agent during plan execution, following the execution of  $\mathbf{a}$  (by the team), and observation of  $o$ , through Eq. (5). In an Event-Driven model, this update step is not always applicable. Planning algorithms, such as CC-PERSEUS, can



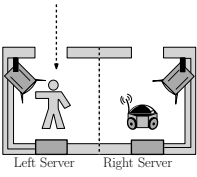


Figure 2: Left: A layout of the *Access2* problem; Right: Size of the model components for the tested scenarios, using event-driven (E) and synchronous (S) approaches.

explicitly model the occurrence of false negative detections of events as symbolic observations. During execution, however, agents will not have access to any information indicating false negative detections. Therefore, agents must take into account the fact that the system can undergo several unobserved transitions between any two belief update steps.

**THEOREM 2.** Let  $f \in \mathcal{O}$  represent false negative detections of events. For an infinite-horizon agent in an Event-Driven POMDP, given that the team is executing  $\mathbf{a}$  and observing  $o$  in belief state  $\hat{b}$ , the belief update step is:

$$\hat{b}_o^{\mathbf{a}} = \frac{\left( H_o^{\mathbf{a}} (I - H_f^{\mathbf{a}})^{-1} \hat{b} \right)}{\mathbf{1}^T \left( H_o^{\mathbf{a}} (I - H_f^{\mathbf{a}})^{-1} \hat{b} \right)}, \quad (16)$$

iff for all eigenvalues  $\lambda$  of  $H_f^{\mathbf{a}}$ ,  $|\lambda_i| < 1$ .

**PROOF.** For  $f \in \mathcal{O}$  indicating false negative observations,  $o \in \mathcal{O} \setminus f$  and  $\mathbf{a} \in \mathcal{A}$ , we have that:

$$\begin{aligned} \hat{b}_o^{\mathbf{a}}(s') &\propto \sum_{s \in S} \Pr(o|s, \mathbf{a}, s') \Pr(s'|s, \mathbf{a}) \times \\ &\left( 1 + \sum_{s'' \in S} \Pr(f|s, \mathbf{a}, s'') \Pr(s'|s'', \mathbf{a}) \right. \\ &\left. + \left( \sum_{s'' \in S} \Pr(f|s, \mathbf{a}, s'') \Pr(s'|s'', \mathbf{a}) \right)^2 + \dots \right) \hat{b}(s) \end{aligned}$$

In matrix notation, as before, this is:

$$\hat{b}_o^{\mathbf{a}} \propto H_o^{\mathbf{a}} \left( \sum_{k=0}^{\infty} (H_f^{\mathbf{a}})^k \right) \hat{b} \quad (17)$$

If  $|\lambda_i| < 1$  for all eigenvalues  $|\lambda|$  of  $H_f^{\mathbf{a}}$ :

$$\hat{b}_o^{\mathbf{a}} = \frac{1}{\eta} H_o^{\mathbf{a}} (I - H_f^{\mathbf{a}})^{-1} \hat{b} \quad (18)$$

with  $\eta = \mathbf{1}^T H_o^{\mathbf{a}} (I - H_f^{\mathbf{a}})^{-1} \hat{b}$ , since  $\mathbf{1}^T \hat{b}_o^{\mathbf{a}} = 1$ . We note that if any  $|\lambda_i| = 1$ , this implies that the system has completely unobservable ( $\Pr(f|\cdot) = 1$ ) chains, and the belief state cannot be tracked.  $\square$

The notation  $\hat{b}$  here indicates run-time belief states, so as to clarify that Eqs. (5) and (16) will produce different outputs. In large systems, if the computational complexity of obtaining  $(I - H_f^{\mathbf{a}})^{-1}$  is prohibitive, it can be approximated through a finite number of sums (see proof).

## EXPERIMENTS

In this section, we present an evaluation of our proposed methodology in simulated multiagent decision-making problems. We consider an autonomous multiagent surveillance

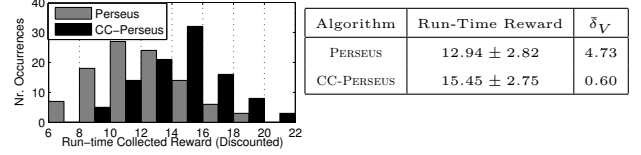


Figure 4: Reward accumulated at run-time for *Access2*. Left: Comparative histogram for 100 runs of 25 steps. Right: respective mean/deviation, and mean error between collected reward and expected value ( $\delta_V$ ).

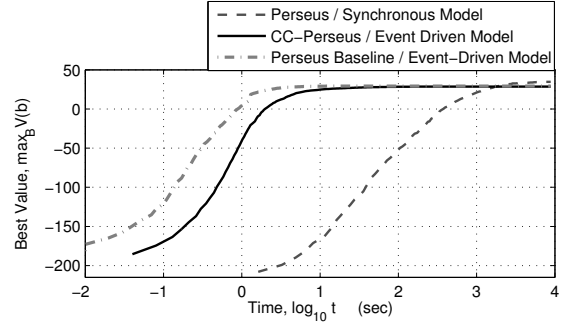
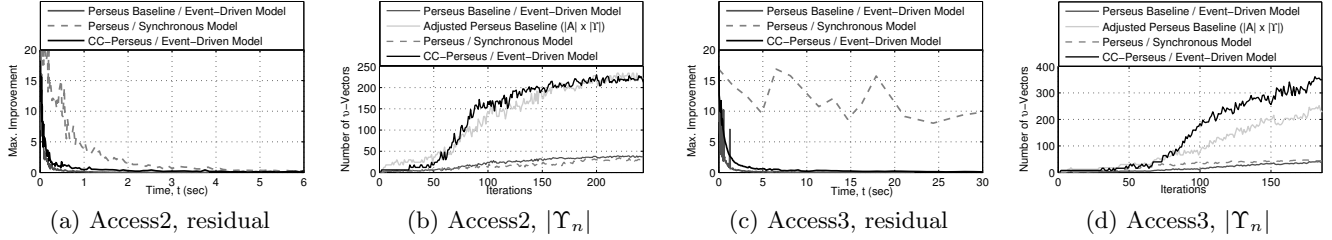


Figure 5: Evolution of  $\max_B V_n(b)$  in real time for the various models/solvers in *Access3*, showing similar final results, but faster convergence in the event-driven case.

system, where static agents (e.g. sensors) and mobile agents (e.g. robots) must cooperate in order to control the access of human operators to sensitive equipment. In the *Access2* problem, two sensors are connected to two restricted-access servers located in adjacent rooms (see Figure 2). Either sensor can mistake the validity of a user’s credentials, or fail to detect that a user is there at all (0.2 probability each). A robot aids these sensors by performing additional measurements, reducing the possibility of false positives and negatives, and also acts as a failsafe in case one of the sensors malfunctions. The robot can move between the two rooms, and knows its position. The goal of the problem is to authorize as many valid users as possible.

This task was represented using both an event-driven and a standard synchronous approach, using factored models [4, 9]. The sizes of the respective model components can be seen in Figure 2. To see why there is such a pronounced difference in  $|\mathcal{O}|$ , consider that each sensor can observe authorized and non-authorized users, hardware failures, or nothing at all. The robot can also observe users (or nothing), along with its own position. In the synchronous approach, we must take the product of all of these possibilities. In the event-driven approach, the only events we need to model are the arrival/exit of a user, the failure of either sensor, and a change of state by the robot, plus an “unobserved” event. Additionally, we can reduce the number of joint actions by ruling out inconsistent decisions. Both the normal PERSEUS algorithm [14] and CC-PERSEUS were run on this problem, over a set of 50 sampled belief points with  $\gamma = 0.95$  and  $h = \infty$ . As a baseline, the PERSEUS algorithm was run on our event-driven model, disregarding the effect of unobservable events. Figure 3a shows the maximum improvement (or



**Figure 3:** (a), (c) Residual difference between successive value function approximations,  $\max_B \{V_n(b) - V_{n-1}(b)\}$ . (b), (d) Size of the value function,  $|\Upsilon_n|$ , as a function of  $n$ .

residual) to the value function between steps using these algorithms, which is indicative of their real-time convergence. From here we can see that CC-PERSEUS on the event-driven model outperforms its standard version in the synchronous setting (total run-time was 22.78 s vs. 29.03 s, respectively, until a residual of  $10^{-4}$ ). Figure 3b also shows how the total number of vectors of CC-PERSEUS follows  $|A|$  times that of the baseline, since  $Q$ -functions are explicitly maintained for each action. In Figure 4 we show how the baseline policy, even though faster to compute, is outperformed by CC-PERSEUS, since action constraints are not considered in the former case. Due to this fact, the expected value calculated by PERSEUS does not correspond to the reward accrued at run-time (an error of 27%, vs. 3% with CC-PERSEUS).

In order to showcase the scalability of these methods, a larger version of the above problem was implemented. *Access3* has 3 rooms/sensors/servers along a corridor, but the sensor at the center can only detect authorized personnel (or nothing). Therefore, there is only one more event with respect to the previous problem, but the presence of another agent causes an exponential increase in the number of observations of the synchronous model. Figure 2 shows the sizes of the models for this problem, and Figures 3c and 3d show performance results. Although synchronous and event-driven models are not strictly comparable with regard to expected reward, since their dynamics are inherently different, we show in Figure 5 an overlay of the best expected value (in the sample set) for *Access3* using either model, to establish that they in fact converged to similar near-optimal policies. Running times to a residual of  $10^{-4}$  were 6m 52.39 s for event-driven CC-PERSEUS and 2h 27m 24.27 s for synchronous PERSEUS. This shows that the simple addition of an agent increased the computational advantage of the event-driven model over its alternative by more than an order of magnitude (also clearly visible in Figure 5). This establishes the scalability of our methods to large partially observable domains, with no assumptions regarding action/observation independence between agents.

## CONCLUSIONS

In this work, we propose a novel modeling approach for multiagent decision-making under partial observability, based on the MPOMDP framework, which draws from the concepts of asynchrony in Discrete-Event systems to allow a more compact representation of such scenarios than what is typically possible through decision-theoretic frameworks.

We have described how such a model could be formalized and shown that it still retains the essential properties that allow it to be solved through dynamic programming. We

have also shown how a common POMDP-solving algorithm could be adapted to function in an event-driven paradigm, and how agents can track belief states at run-time in the presence of false negative observations.

As future work, our event-driven models can be extended to continuous-time dynamics (such as in Semi-Markov Decision Processes). This would facilitate the application of these methods in other domains. We will also investigate specific solvers for this class of models, which could exploit their asynchrony for further computational gains.

## ACKNOWLEDGMENTS

This work was funded in part by the Carnegie Mellon - Portugal Program (project CMU-PT/SIA/0023/2009), and by Fundação para a Ciência e a Tecnologia (FCT), through the PID-DAC Program funds (ISR/IST pluriannual funding, ref. PEst-OE/EEI/LA0009/2011).

J.M. was supported by a PhD Student Scholarship, from the Portuguese FCT POCTI programme, SFRH/BD/44661/2008.

M.S. is funded by the FP7 Marie Curie Actions Individual Fellowship #275217 (FP7-PEOPLE-2010-IEF).

## REFERENCES

- [1] R. Becker, S. Zilberstein, and V. Lesser. Decentralized markov decision processes with event-driven interactions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 302–309. IEEE Computer Society, 2004.
- [2] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.
- [3] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.
- [4] C. Boutilier, R. Dearden, M. Goldszmidt, et al. Exploiting structure in policy construction. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1104–1113, 1995.
- [5] C. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer academic publishers, 1999.
- [6] H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.
- [7] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the National Conference on Artificial Intelligence*, 2005.
- [8] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In

- International Joint Conference on Artificial Intelligence*, volume 18, pages 1025–1032. Citeseer, 2003.
- [9] P. Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto, 2005.
  - [10] M. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.
  - [11] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.
  - [12] W. Shen and D. H. Norrie. Agent-based systems for intelligent manufacturing: A state-of-the-art survey. *Knowledge and Information Systems*, 1:129–156, 1999.
  - [13] E. Sondik. *The optimal control of partially observable markov processes*. PhD thesis, Stanford, 1971.
  - [14] M. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24(1):195–220, 2005.
  - [15] F. Wu and X. Chen. Solving large-scale and sparse-reward DEC-POMDPs with correlation-MDPs. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Computer Science*, pages 208–219. Springer Berlin / Heidelberg, 2008.

# Organizational Design Principles and Techniques for Decision-Theoretic Agents\*

Jason Sleight and Edmund H. Durfee  
Computer Science and Engineering  
University of Michigan  
Ann Arbor, MI 48109  
{jsleight,durfee}@umich.edu

## ABSTRACT

Recent research has shown how an organization can influence a decision-theoretic agent by replacing one or more of its model components (transition/reward functions, action/state spaces, etc.), and how each of these influences impacts the agent’s decision-making performance. This paper delves more precisely into exactly which parts of an agent’s model should be organizationally influenced, and asserts a broader principle for delineating what aspects of an agent’s behavior an organization should be sanctioned to influence. We present a formal framework for specifying factored organizational influences and incorporating them into agents’ decision models, and empirically demonstrate that organizational specifications based on our proposed principle outperform the alternatives. We further describe an algorithm for automating the organizational-design process that is inspired by this principle, and demonstrate empirically that its organizational designs are both intuitively sensible and also find and exploit domain structure that our hand-generated designs miss.

## Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Coherence & Co-ordination, Multiagent Systems*

## General Terms

Design, performance

## Keywords

Organization, organizational design

## 1. INTRODUCTION

A well-designed organization judiciously applies its available levers of influence to lightly, but firmly, guide its members into working well together. For decision-theoretic agents, the available levers of influence correspond to shaping the components of the agents’ decision-theoretic models, including their reward and transition functions and their state and action spaces (e.g., [1, 11, 14, 16]). Firm but light guidance means that components should be chosen and shaped by the organization to help agents avoid doing (and preferably

even avoid thinking about doing) redundant, contradictory, or counterproductive actions, while leaving them freedom to exercise their abilities at their own discretion otherwise.

Recently [11], we posited that the components of a decision-theoretic model define a vocabulary in which organizational designs can be expressed to decision-theoretic agents, and examined how organizationally influencing the agents via different (combinations of) components impacts the quality and costs of coordinated decision making. Our work demonstrated how replacing components with organizationally-provided ones could firmly guide agents’ behavior in effective ways. We contend, though, that this process of replacing entire components was also unnecessarily heavy-handed.

In this paper, we factor the components of the decision-theoretic model to create a richer vocabulary in which organizational design can more flexibly be expressed. A contribution of this paper (Section 3) is a description of this vocabulary, along with the semantics attached to it by an agent incorporating a design into its local model. Unfortunately, the richer vocabulary also induces an exponential growth in the space of expressible organizations, which can bog down the design process. To combat this growth, we also contribute a strategy for forming organizational designs based on the principle that organizational influence should be limited to addressing agents’ reward and/or transition dependencies. In Section 4, we provide our rationale for this principle and provide experimental evidence of its effectiveness.

Even with this design principle, however, the organizational design process can be time-consuming and error-prone if done by hand, as teasing out the pertinent interdependencies can be difficult. This argues for automated techniques for conducting the organizational design process (ODP); such techniques are the other main contribution of this paper (Section 5). Briefly, the underlying insight behind our automated technique for ODP is to also cast the ODP in decision-theoretic terms, modeling and reasoning about the agents’ joint behaviors abstractly to predict desirable patterns of joint action, and then influencing the agents into these coordination patterns. Section 5 provides details of the ODP, and presents preliminary empirical results showing that it finds organizational designs that make sense intuitively and that can exploit structure in the domain. We end the paper (Sections 6 and 7) by summarizing our work and discussing how it relates to other past and ongoing work in organizational design for multiagent systems. We next (Section 2) describe our agents’ decision-theoretic framework and introduce the stylized firefighting domain that we use for illustration and experimentation throughout this paper.

\*This paper appears in AAMAS 2013.

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with AAMAS, May 2013, St. Paul, Minnesota, USA.

## 2. PROBLEM DOMAIN

Like in our previous work [11], we adopt a standard decentralized partially observable Markov decision process (Dec-POMDP) paradigm [2],  $\mathcal{M} = \langle \mathcal{N}, \mathcal{S}, \alpha, A, R, P, \Omega, O, T \rangle$ , where:  $\mathcal{N}$  is the set of  $n$  cooperative agents;  $\mathcal{S}$  is the (finite) set of global states;  $\alpha$  is a probability distribution over initial global states;  $A$  is the (finite) set of possible joint actions;  $R$  is the joint reward function;  $P$  is the joint transition function;  $\Omega$  is the (finite) set of possible joint observations;  $O$  is the joint observation function; and  $T$  is the finite time horizon. Given a full specification of the Dec-POMDP, an optimal joint policy,  $\pi^*$ , can be formulated in principle. In practice, however, finding such a policy for anything but very simple problems (with few agents and small state and action spaces) is intractable [2], and even if found, executing such a policy is problematic because it generally assumes that all agents have the same beliefs about the global state.

For these reasons, multiagent approaches to solving such problems often assume that each agent possesses a local view of the joint problem, and utilize factored models (e.g., [4]) to more compactly represent the decision problem. For example, global state is factored into a set of  $m_S$  state features, such that  $S = F_1 \times \dots \times F_{m_S}$ , where  $F_j$  is the (finite) domain of state feature  $j$ . Each agent  $i$  has a local state representation  $S_i$  consisting of a subset of the  $m_S$  features. Each  $X$  of the other model components (i.e.,  $X \in \{\alpha, A, R, P, \Omega, O\}$ ), is then defined in terms of this factored state representation, and similarly factored into  $m_X$  features. The number of factors in a model-component can differ from agent to agent, but to avoid notational clutter this subtlety is ignored in the description that follows. We further adopt the common assumption of local full observability (each agent's local observations uniquely determine its local state).

Given these assumptions, the local decision model  $\mathcal{M}_i$  of agent  $i$  represents a local Markov decision process (MDP)  $\mathcal{M}_i = \langle S_i, \alpha_i, A_i, R_i, P_i, T_i \rangle$ , which defines the local states, actions, rewards, etc.

- $S_i = F_{i_1} \times F_{i_2} \times \dots \times F_{i_{m_S}}$ .
- $\alpha_i = \langle \alpha_{i_1}, \dots, \alpha_{i_{m_\alpha}} \rangle$  where  $\alpha_{i_j} : (\otimes_k F_{i_k}) \rightarrow [0, 1]$  and  $\alpha_i$  partitions  $\{F_{i_k}\}$ .
- $A_i = \{a_{i_1}, \dots, a_{i_{m_A}}\}$ .
- $R_i = \sum_{j=1}^{m_R} R_{i_j}$  where  $R_{i_j} : (\otimes_k F_{i_k}) \times A_i \rightarrow \mathbb{R}$ .
- $P_i = \langle P_{i_1}, \dots, P_{i_{m_P}} \rangle$  where  $P_{i_j} : (\otimes_k F_{i_k}) \times A_i \times (\otimes_{k'} F_{i_{k'}}) \rightarrow [0, 1]$  and the target of  $P_i$  partitions  $\{F_{i_{k'}}\}$ .
- $T_i \in \mathbb{Z}^+$ .

Each agent can use its local MDP to compute its (optimal) local policy  $\pi_i^*$  with respect to  $\mathcal{M}_i$ . The joint policy is then simply defined as  $\pi = \langle \pi_1^*, \pi_2^*, \dots, \pi_n^* \rangle$ .

To illustrate a problem of this type, we reuse a simplified firefighting scenario [11], where firefighting agents and fires to be fought exist in a grid world (Figure 1). The global state consists of: the locations of the agents,  $\ell_i \in \text{Cells}$  for agent  $i$ ; the fire intensity,  $I_c \in \mathbb{Z}^+$  for each cell  $c$ . Further, compared to our prior work [11], we also add a delay,  $\delta_c \in [0, 1]$  for each cell  $c$ , which stochastically prevents movement into that cell with probability  $\delta_c$ . Figure 1 shows an initial global state, where the locations of agents A1 and A2 are shown, along with the intensity of fire in the 2 cells with  $I_c > 0$ . In Figure 1, (H)igh, (M)edium, and (L)ow delay correspond to  $\delta$  equal 0.8, 0.5, and 0.0 respectively. Each agent has 6 actions: a NOOP action that makes no change to the world state; 4

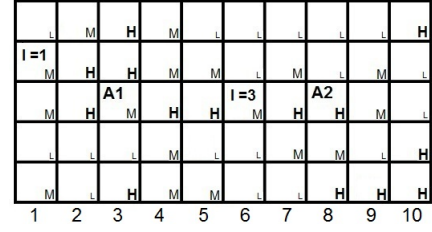


Figure 1: Example initial state in the firefighting grid world.  $Ai$  is the position of agent  $i$ , and  $I = x$  indicates that there is a fire in that cell with intensity  $x$ . Letters designate a (H)igh, (M)edium, or (L)ow delay in that cell.

possible movement actions (N, S, E, W) that move the agent one cell in the specified direction with probability  $1 - \delta_{c_{dest}}$ , and thus equates to a NOOP with probability  $\delta_{c_{dest}}$  (or if there is no cell in that direction); and a fight-fire (FF) action that decrements by 1 the intensity of the agent's current cell (to a minimum of 0) and otherwise behaves like a NOOP. Joint actions are defined as the aggregation of the agents' local actions. Movement actions are independent (agents can occupy the same location), but FF actions are not: the intensity of a cell only decreases by 1 even if multiple agents simultaneously fight it. The joint reward for the agents in a state prior to reaching time horizon  $T$  is  $-\sum_c I_c$ . When  $T$  is reached, the problem episode ends, and the joint reward is  $-10 \sum_c I_c$ , encouraging the agents to put all the fires out before the deadline.

An example of an agent's local model from this joint model follows. An agent  $i$ 's local state consists of  $\ell_i$ ,  $I_c$  for each cell, and  $\delta_c$  for each cell. That is, it does *not* include the positions of other agents. Hence, its local action space only includes its 6 actions, and its local transition model only models how its local actions affect its local state. Its local reward function is the same as the global reward function; note that in this case the sum of the agents' local rewards will overestimate the true (negative) reward. Its local finite time horizon is identical to the global finite time horizon, and its local initial state distribution is calculated by directly mapping the initial distribution of global states into the local state space. Figure 2 shows the local model for agent  $i$  represented as a dynamic Bayesian network (DBN). Given such a local model, each agent will formulate a local policy that would fight the fires optimally if the agent were alone in the world. Note that, in general, the joint policy formed by the combination of these optimal local policies will not itself be optimal. For example, in Figure 1, both agents will be drawn to the high intensity cell first and try to redundantly fight its fire rather than dividing up to fight the two cells with fires concurrently.

## 3. FACTORED ORGANIZATION

Our prior work showed that organizational influence can be exerted on decision-theoretic agents by replacing components of their decision models [11]. Our objective in this paper is to allow more nuanced organizational influence through selected *factors* of components, and so organizational designs are factored like local models. To selectively modify a local model, an organizational design should be able to express: alterations to an agent's existing factors, such as altering a transition factor to reflect expectations about fires in nearby cells being

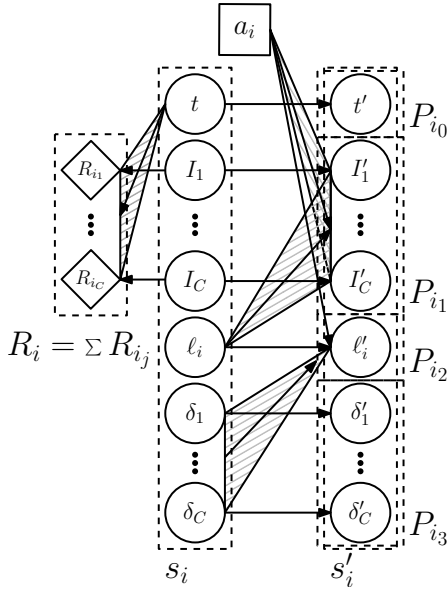


Figure 2: An example factoring for agent  $i$  represented as a dynamic Bayesian network (DBN).

extinguished (due to other agents' efforts); additions to an agent's factors, such as introducing a new reward factor to induce an agent to fight fires in a region of responsibility and punish it for fighting fires in others' regions; and deletions of an agent's factors, such as blocking an agent from including distant cells' intensities in its local state.

Our organizational specification classifies factors into two sets: those to be added and those to be blocked, where if an added factor already has a corresponding existing factor for the agent, it simply alters (replaces) it. We formally define an organization  $\Theta = \langle \theta_1, \dots, \theta_n \rangle$ , where  $\theta_i$  is the organizational component for agent  $i$ , defined as  $\theta_i = \langle \{F_{ij}^\Theta\}, \{\bar{F}_{ij}^\Theta\}, \{\alpha_{ij}^\Theta\}, \{\bar{\alpha}_{ij}^\Theta\}, \{R_{ij}^\Theta\}, \{\bar{R}_{ij}^\Theta\}, \{P_{ij}^\Theta\}, \{T_i^\Theta\} \rangle$ , where:

- $\{F_{ij}^\Theta\}, \{\bar{F}_{ij}^\Theta\}$  specify the sets of local state factors organizationally added to and blocked from agent  $i$ 's model respectively.
- $\{\alpha_{ij}^\Theta\}$  is the organizational augmentation for agent  $i$ 's local initial-state distribution.
- $\{a_{ij}^\Theta\}, \{\bar{a}_{ij}^\Theta\}$  specify the sets of agent  $i$ 's local actions organizationally added and blocked respectively.
- $\{R_{ij}^\Theta\}, \{\bar{R}_{ij}^\Theta\}$  specify the sets of agent  $i$ 's local reward factors organizationally added and blocked respectively.
- $\{P_{ij}^\Theta\}$  is the organizational augmentation to agent  $i$ 's local transition function.
- $T_i^\Theta$  is agent  $i$ 's organizational finite time horizon.

We restrict specifications to those where the state factor, action, and reward components are consistent such that no factor appears in both sets; for example  $\{R_{ij}^\Theta\} \cap \{\bar{R}_{ij}^\Theta\} = \emptyset$ . Note that some components ( $\alpha, P, T$ ) do not have "blocked" counterparts like the other components, because an agent must always have a model of this information or its decision-making process is under-defined. Hence, an organizational design cannot block a factor in these components without replacing it, which is equivalent to altering it. We further restrict specifications to be consistent with what an agent is internally capable of modeling.

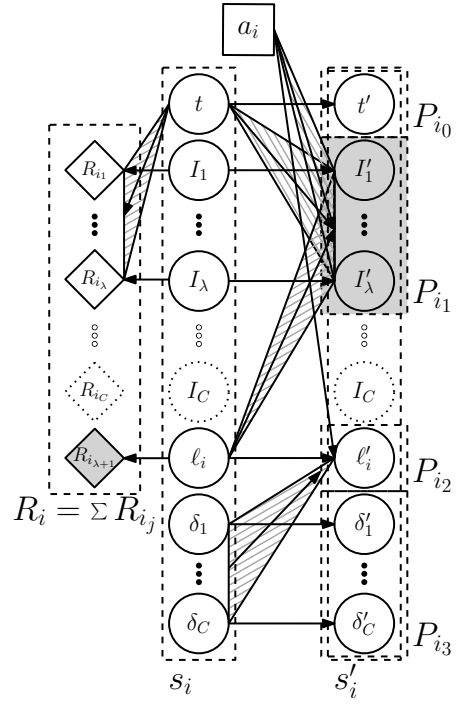


Figure 3: An example organizational augmentation to the DBN from Figure 2. Shaded regions indicate factors that were organizationally altered or added, while dotted regions indicate factors that were organizationally blocked.

When incorporating an organizational specification into its local model, an agent  $i$  overlays  $\theta_i$  onto  $\mathcal{M}_i$  to create its augmented model by adding entirely new local factors, removing blocked local factors, and overwriting replaced local factors. This overlaying process thus resembles how, for example, coordination locales model domain dynamics by overriding an agent's local transition/reward models, and social model shaping augments those local models to coerce coordination [1, 14, 16]. Figure 3 shows an example organizational transformation to the DBN from Figure 2, where agent  $i$  is assigned responsibility for cells 1 through  $\lambda$ . The organization specifies this responsibility by blocking  $I_C$ 's outside of the agent's region, adding a new reward factor for being located within its region, and modifying the  $I_C$ -transition factor to account for  $I_C$ 's in its region decreasing over time (due to other agents' efforts).

In this paper, we will assume that an organization,  $\Theta$ , is fixed, and thus the agents will reuse it over a series of problem episodes sampled from a distribution, even though the influences might be suboptimal for some episodes. Depending on the context, however, an agent  $i$  might be permitted to disregard some (or all) of  $\theta_i$ , and rely instead on its local model, or could even try  $\theta_i$ , keep the useful portions, and disregard (or replace) the rest. Organizationally-Adept Agents [3], for example, could make such decisions. Alternatively, the organizational design process might police itself to only specify factors that support coordination more generally without micromanaging. In the remainder of this paper, we assume agents will adopt  $\Theta$  as given, and so focus on the question of which factors  $\Theta$  should specify, and the values those factors should take, to provide effective influences.

## 4. DESIGN PRINCIPLES

The organizational syntax we defined in Section 3 is capable of specifying much more than organizational influences, and in actuality is a general-purpose programming language for decision-theoretic agents. For example, a syntactically correct  $\Theta$  could completely overwrite an agent’s entire local model down to the smallest detail of how tasks are performed, the epitome of micromanagement. However, such usage would exceed the commonsense bounds of what organizations are customarily expected to influence. A natural question, therefore, is whether we can define explicit principles that embody an intuitive understanding of how organizations should be designed, and that apply to our new, richer vocabulary for factored influence on decision-theoretic agents. In this paper, we answer this question by proposing and testing one such principle to guide decisions about which portions of the agents’ models an organizational design should influence.

Organizations should not dictate or micromanage because individual agents might (and generally do) possess their own expertise, and leaving them room to exercise their local expertise benefits the collective organizational objectives. Assuming that agents are locally skilled, then, we observe that what an organizational view has that individuals lack is a more global awareness across agents’ activities, where if individual agents had such awareness they could make more informed decisions. Hence, an organizational design should use its more global perspective to influence agents into acting like they would if they were more globally aware themselves.

Restated, the claim is that an organizational design should use its global perspective to improve agents’ decisions that impact each other, and otherwise should allow agents to exercise their local capabilities. We codify this assertion in the following **organizational design principle**: a well-designed organization should influence only the factors of agents’ models that are associated with agent interactions. This principle is surprisingly applicable for creating organizational designs for decision-theoretic agents, where factors associated with interactions are directly captured in joint reward/transition functions, and indeed where the specification of agents’ decision models often explicitly separates out the dependent factors (e.g., coordination locales [14, 16]) from the independent ones. This is neatly illustrated in our specification of the firefighting domain, where agent movements are independent (where one agent moves does not affect the states/rewards of another agent), but firefighting actions are not (one agent’s states/rewards are affected by another’s fighting of a fire).

### 4.1 Evaluation Strategy

To test this organizational design principle, we enumerated a space of factored organizations, each of which draws on the same, fully-specified organization, but adopts a different subset of factors taken from that full specification. We use our previously developed `smallOverlapOrg` [11] as our fully-specified organization in these experiments, which, roughly speaking, assigns an overlapping primary area of responsibility (PAR) to each agent. For example, in Figure 1, agent 1’s PAR is the cells in columns 1–7, and agent 2’s PAR is the cells in columns 4–10. The `smallOverlapOrg`’s state component fully overwrites an agent’s state space to block factors for  $I_c$ ’s outside the agent’s PAR (i.e., agents ignore fires outside of their PARs). Its action component overwrites an agent’s action space to block the agent from wasting time considering actions that would cause it to leave its PAR.

The `smallOverlapOrg`’s transition component replaces an agent’s model of what its movement, firefighting, and NOOP actions do with what the organizational design thinks that they do (including capturing probabilistic changes to  $I_c$ ’s in the agents’ overlapping PARs caused by the other agent).

Previously [11], we explored the effects on group performance of including various combinations of these components. Here, we further subdivide the components into their factors. In particular, in its factored form, the transition component has one factor for the effects of agent movement actions, and another for firefighting actions. We draw on our organizational design principle to combine only the factors that correspond to **reward/transition-dependencies (R/T-Ds)**. Revisiting the `smallOverlapOrg`’s components, R/T-D factors include: the entire state and action components (since these reflect the organization’s global awareness that agents can depend on each other to fight fires in their respective PARs); and the transition factor for  $I_c$ ’s (summarizing expectations of when fires will be extinguished by other agents in the overlapping PARs). On the other hand, the transition factor for agents’ movement actions is a non-R/T-D factor, because agent movements are independent.

Our organizational design principle thus leads us to the hypothesis that a specification including only these R/T-D factors will capitalize on the global perspective of an organization without overstepping the bounds into micromanagement: it will perform as well or better than other combinations of factors. To test this hypothesis, we constructed this **R/T-DOrg** organizational specification, as well as an organization that includes only non-R/T-D factors, the **non-R/T-DOrg**. For completeness, we also considered the full set of factors yielding the **unfactoredOrg** (identical to the unfactored `smallOverlapOrg`), and the empty set of factors which yields the **localOrg** (where agents are uninfluenced by any organization). To exercise the assumption that agents in an organization can contribute expertise of their own, we evaluated each of these four organizations across a spectrum of settings where we varied the relative quality of agent expertise compared to the organization’s model. Specifically, we degraded the organization’s view of the cell delays by applying a smoothing filter over the true environment delays according to the particular settings of that experiment. For example, the 100-smoothed setting had a smoothing filter iteratively applied to every cell 100 times, whereas in the 0-smoothed setting the organization’s view precisely matches the real delays, etc. In this way, as the organization’s model of delays blurs, it remains accurate in terms of cells’ mean delays but loses precision (about the physical distribution of delays).

### 4.2 Results

To test the degree to which an organizational design provides long-term benefit to a multiagent system, we run each fixed organizational design over 3000 randomly-generated problem instances, where each instance is an episode that begins with a randomized configuration of cell intensities and delays, and ends when the time horizon is reached. By the luck of the draw, some problem instances might be well suited to one organization over another. We focus on aggregate performance over the episodes not only to smooth out the randomness of the instances but also to assess an organization’s effectiveness over the long term. The performance measures of interest are the expected joint reward and the

% Results Included	100%			Top 25%			Top 5%		
# Smoothing	0	10	100	0	10	100	0	10	100
R/T-DOrg	1.45%	1.40%	1.40%	6.05%	5.92%	5.98%	23.27%	23.55%	23.44%
unfactoredOrg	1.45%	-6.24%	-7.30%	6.10%	-9.61%	-10.70%	23.21%	-19.48%	-19.76%
non-R/T-DOrg	0.01%	-7.41%	-8.29%	0.00%	-12.26%	-13.02%	0.00%	-24.88%	-24.47%

(a) Expected improvement in joint reward

% Results Included	100%			Top 25%			Top 5%		
# Smoothing	0	10	100	0	10	100	0	10	100
R/T-DOrg	34.08%	33.02%	33.12%	34.81%	34.04%	33.77%	27.88%	27.79%	27.76%
unfactoredOrg	34.08%	33.00%	33.13%	34.84%	30.83%	30.90%	28.03%	28.63%	29.36%
non-R/T-DOrg	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

(b) Expected reduction in number of states in agent planning process

Table 1: Percent improvement compared to **localOrg** for experiments in Section 4. %-results-included refers to filtering out episodes based upon the magnitude of the percent reward difference until only the top  $x\%$  of episodes remain (those same episodes are used for the corresponding expected-number-of-states results). #-smoothing refers to the number of times the smoothing filter was applied to each cell in the ODPs input model.

local planning overhead of the agents. A high-performing organization is one that improves joint reward while also simplifying each agent’s local planning problem.

To run our experiments, each agent independently overlays its organizational specification (see Section 3) to create its combined local MDP. It then uses this model to create the reachable state space from the episode’s initial state forward. The agent creates its optimal local policy for the reachable state space using CPLEX [7] to solve the linear program as formulated by Kallenberg [8]. To reduce the impact of stochastic transitions, we simulated each joint policy 5000 times in the execution environment to calculate the expected joint reward. As we did previously [11], if during execution an agent reaches an unplanned state (e.g., due to another agent unexpectedly fighting a fire), it constructs a new policy going forward from its current state.

Table 1 presents results from our experiments and highlights several important points. Firstly, we observe that, in expectation, the value of organizational influence is only 1.45% in this domain. While this initially seems disappointing, on reflection it is unsurprising: for most episodes, an agent’s lack of global awareness is inconsequential, because the initial placement of agents (Figure 1) and fires is such that, for most cases, agents’ local decisions lead them into complementary actions. However, in episodes where a high-intensity fire is located between the agents’ initial positions, agents acting locally often miscoordinate, providing opportunities for organization to improve performance. This suggests that the average performance hides a heavy-tailed distribution. If we sort the episode results by the magnitude of the percent difference versus **localOrg**,  $\frac{|localOrg - org|}{localOrg}$ , and filter the sorted results to only include the top percentages, then we observe that when organizational design does impact performance, it has a noticeable impact. For example, **R/T-DOrg** has a 23.27% expected improvement in 5% of the episodes. Note that in Table 1b, the improvement declines of **unfactoredOrg** and **R/T-DOrg** from the 25% to 5% settings are also caused by this domain property. The R/T-D based organizations coordinate correctly when a high-intensity fire is located between the agents due to  $I_c$  transition shaping; however, transition shaping also increases the size of the agents’ state spaces.

Secondly, as can be seen in any of the columns, the organizations that influence R/T-D factors (i.e., **unfactoredOrg** and **R/T-DOrg**) outperform those without R/T-D based influences, both in terms of expected joint reward as well as computational costs. Moreover, the non-R/T-D based influences can severely degrade system performance as the organization’s model deviates from the agents’ true models, as demonstrated by the 10- and 100-smoothed cases for **non-R/T-DOrg** and **unfactoredOrg**. This illustrates the costs of heavy-handed micromanagement that undervalue agents’ expertise, and supports our claim of superiority for our factored organizational formulation, as organizations that omit non-R/T-D based influences allow agents to exercise their expertise to avoid these problems.

Unfortunately, we are unable to fully contrast our organizations against computing the optimal joint policy,  $\pi^*$ , as we were computationally unable to calculate  $\pi^*$  for all 3000 experiments. However, in a small subsample of the problems that we could complete, we observed that  $\pi^*$  contains approximately two orders of magnitude more states and achieves approximately three percent more expected reward as compared to the **localOrg**.

## 5. AUTOMATED DESIGN

Intuitively, organizational design should use a global perspective to identify patterns of interactions that would arise when agents cooperate effectively, and then codify these patterns into influences that agents internalize. In Section 4, for example, the **R/T-DOrg** stops agents from even thinking about fighting fires that another agent is clearly better positioned to fight, and focuses them on fighting nearby fires. Furthermore, the factored model and organizational design principle we presented in the previous sections suggest the foundations of a process for automating organizational design: identify the R/T-Ds using the Dec-POMDP model; deduce from these a space of joint behaviors to seek/avoid; and use these to select and shape factors for agents’ local models that steer agents to/from local decisions that lead to the good/bad interactions.

Unfortunately, identifying R/T-Ds can sometimes be difficult in models where joint interactions are not explicitly provided as part of the model specification. For this rea-



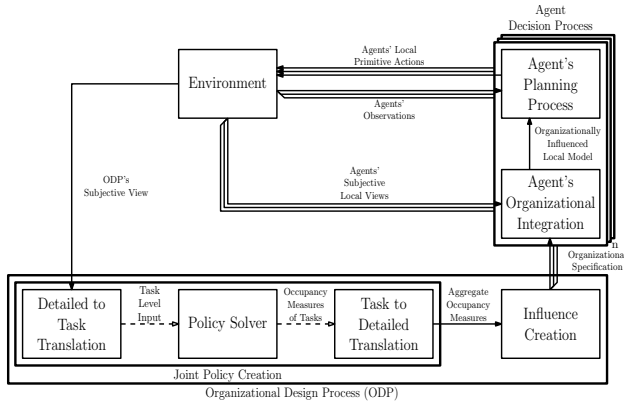


Figure 4: A conceptual overview of our ODP and how it interacts with the environment and the agents.

son, the automated organizational design process (ODP) we have devised exploits domain knowledge if it is provided, but still functions (although with increased computational costs) without explicit R/T-D specifications. At a high level (Figure 4), our ODP begins with joint policy creation, wherein it calculates optimally-coordinated joint policies over a space of possible problem instances to estimate the aggregate occupancy measures for the problem space. An occupancy measure  $x(s, a)$  for a policy tree gives the probability of reaching state  $s$  and taking action  $a$ , and is directly optimized via the linear program [8] our ODP uses for policy creation. Our ODP then uses the aggregate occupancy measures as the basis for influence creation, such that  $\Theta$ 's influences guide the agents into the behavior patterns captured by the aggregate occupancy measures.

## 5.1 Organizational Design Process

We now step through our currently implemented automated ODP in more detail, and then at the end of this section discuss some ways in which it can generalize to incorporate additional knowledge if it is available. We assume there is a “true” environment in which the agents are operating, but neither an agent nor the ODP is assumed to have a perfect model of that environment. Rather, each agent has a subjective local view, which is represented as a local MDP, such as those described in Section 2, where an agent does not model other agents but can have accurate models of the environment such as the delays (as in the experiments in Section 4). The ODP has a subjective view of the environment and agents in the form of a Dec-POMDP, describing the environment as well as the agents and their capabilities, but may be imperfect in either or both aspects (e.g., imprecise models of the cell delays as in Section 4).

Our ODP begins by using the options framework [13] from the hierarchical learning community to abstract its Dec-POMDP into a task-level model that focuses on tasks to accomplish rather than actions to take. As is customary, one option is created for each task in the domain, where a task corresponds to achieving a particular subgoal. For simplicity, in the experiments that follow, we informed our ODP that good subgoals are states where  $I_c \rightarrow 0$ ; however, subgoal detection could be automated using techniques from the hierarchical learning community (e.g., [12]). Reasoning with task-level options not only reduces computation, but

also naturally emphasizes the most significant interactions among agents, while remaining largely agnostic about how the agents will translate their options into detailed actions. Of course, the ODP still requires an estimate of each option's properties (i.e., its expected reward, termination states, and primitive actions it might translate into). Our ODP creates this information itself by using its detailed Dec-POMDP to heuristically calculate the likely effects of an agent's policy within an option.

Our ODP then solves a linear program representation of the task-level Dec-POMDP (as a centralized process) analogously to the policy-solving process our agents used in Section 4. This results in occupancy measures for state-option pairs in the joint task-level policy. The ODP then inverts its abstraction process using the properties of each option, which projects state-option occupancy measures downward to estimate state-action occupancy measures. This joint policy creation process (i.e., abstracting to a task-level model, solving for the task-level joint policy, then inverting the abstraction) is repeated for a space of problems sampled from the Dec-POMDP, which results in aggregate occupancy measures that identify patterns of optimal task-level interactions.

Specifically, our ODP uses the aggregate state-action occupancy measures to create influences for the agents from the following patterns.

**Actions:** For agent  $i$ , if the occupancy measure  $x(s_i, a_i) = 0$  then block  $a_i$  from the set of available actions in  $s_i$ . For example, in the firefighting domain, if the ODP's policies never have an agent move into certain cells (e.g., cells always serviced by closer agents), then actions that would move the agent into those cells are blocked.

**States:** If agent  $i$ 's action choice under the joint policy is invariant with respect to state factor  $F_{i_k}$  given any values for the other state factors, then  $F_{i_k}$  can be blocked from agent  $i$ 's state factors (since it contributes no information). For example, in the firefighting domain, the intensity of cells distant to an agent (always fought by someone else) do not impact the agent's action selection and thus are blocked.

**Transitions:** For an agent, modify the transition factors for each of its remaining state factors (after blocking state factors as above) to include the probabilistic effects of the other agents. For example, in the firefighting domain, an agent's transition factor for an overlapping cell's intensity would be altered to reflect the probability that some other agent executes the FF action in that cell at certain times.

Notice that the above influence mechanisms use very strict criteria for when to remove a factor. Essentially, our ODP finds the maximal reduction to an agent's model that does not decrease the expected joint reward. In principle, however, further reductions could sacrifice reward in order to further influence the agents. For example in the firefighting domain, it could be the case that agent  $i$  rarely needs to know the intensity of a relatively distant cell, so the expected reward loss from not knowing it is very small. Thus, blocking that factor has negligible impact on the expected joint reward, but could yield significant computational savings during agent planning. Tradeoffs like these are the focus of model abstraction research [9], and in the future we plan to extend our work to address these issues.

We now briefly discuss ways of generalizing our ODP to incorporate additional knowledge, should it be available. If an explicit specification of the R/T-Ds is provided, then the joint policy creation process could simply create a policy

directly from the R/T-Ds. Further, if (partial) information of good joint interactions is known (e.g., a partial-order plan), then the joint policy creation process could be constrained to reflect that knowledge. Alternatively, knowledge of the R/T-Ds could be reflected by simply inputting a corresponding option-level model to the ODP (along with properties of the options to be used for inverting the abstraction) as opposed to a detailed Dec-POMDP. Moreover, if the ODP has an option-level model, but lacks knowledge of how options translate into local actions, states, and transitions, it could influence agents by adding/blocking local reward factors to induce coordinated behavior. Reward influences thus provide a fallback means for exerting organizational influences. Typically, however, action, state, and/or transition influences are preferable since they can reduce agents’ reasoning efforts by preventing consideration of ineffectual behaviors.

## 5.2 Evaluation

Before presenting our evaluation, we must first discuss the parameters of our automated ODP in these experiments. That is, even in the relatively simple firefighting domain with restrictions on the starting positions of the agents and a maximum number of fires and their intensities, the space of possible initial global states is exceedingly large ( $\sim 22,000$ ). Furthermore, the total reachable state space from any initial state contains millions of states. For these reasons, our ODP is computationally unable to exactly solve for the complete, optimal joint policy in every state. Thus our ODP instead bases its designs on a representative sub-sampling from the entire initial state distribution (i.e., we biased the sampling to ensure that the ODP samples a diverse set of initial states). To test the impact the sample size has on the resulting organizations, we present results from two different parameter settings, 50 (0.23% of possible initial states) and 150 (0.68%). **XAutoOrg** refers to the organization designed by our ODP using  $X \in \{50, 150\}$  initial state samples.

We begin our evaluation by confirming that our ODP’s designs are intuitively sensible. Figures 5a/5b show the cumulative occupancy measures by cell (shaded by magnitude),  $\sum x_i(s_i, \cdot)$ , for each agent, created in response to the delays in Figure 5c, and represent a summary of the action shaping specified to each agent (i.e., cells with low cumulative occupancy measure typically have more tightly restricted actions). Darker shaded cells thus represent those that the ODP expects the agent will more likely visit. We observe in Figures 5a and 5b that the agents’ influences are correlated, and each agent is more or less expected to be responsible for a particular region (with some overlap in between). Further, as seen by comparing Figure 5c to Figures 5a/5b, the ODP recognized the domain structure, and tailored its influences to skew the agents’ regions towards those cells they can easily reach.

We also tested the **XAutoOrgs** using the same empirical methodology and episodes as in Section 4 to ensure that they yield high system performance in addition to being intuitively sensible. Table 2 presents the results of these experiments as well as repeats the results from the best hand-designed organization from Section 4. We present only the results for the 0-smoothed case; however, the other cases are nearly identical to the 0-smoothed one, implying that our ODP scales gracefully as its input model degrades (due to following our principle of specifying R/T-D based influences).

As Table 2 illustrates, our **XAutoOrgs** compare well

% Results Included	100%	Top 25%	Top 5%
<b>R/T-DOrg</b>	1.45%	6.05%	23.27%
<b>50AutoOrg</b>	2.06%	3.33%	4.11%
<b>150AutoOrg</b>	2.17%	5.25%	13.63%

(a) Expected improvement in joint reward

% Results Included	100%	Top 25%	Top 5%
<b>R/T-DOrg</b>	37.07%	34.81%	30.39%
<b>50AutoOrg</b>	38.84%	39.29%	49.15%
<b>150AutoOrg</b>	19.36%	20.75%	38.91%

(b) Expected reduction in number of states in agent planning

Table 2: Percent improvement compared to **localOrg** for experiments in Section 5. %-results-included data uses the same subset of episodes as in Table 1.

against the hand-designed **R/T-DOrg**, but make different tradeoffs as demonstrated by the top 25% and 5% columns. That is, **R/T-DOrg** has little to no impact in most episodes, but then has substantial gains in a few episodes, whereas the **XAutoOrgs** yield a slightly larger overall improvement, but accomplish this by having moderate performance gains in many episodes. This observation suggests that the **XAutoOrgs** are not over-specialized to particular situations the ODP might have encountered, but rather provide general influences, since their improvement gains are more uniformly distributed over the problem space relative to **R/T-DOrg**. We also observe that, as our ODP gains a more complete input model, it uses this additional information to infer more specialized behavior patterns and thus exerts more specialized influences, as evidenced by the **150AutoOrg** having moderately higher performance gains relative to **50AutoOrg** in the 25% and 5% cases. Finally, we observe that the **XAutoOrgs**’ state space improvements actually increase as we filter out episodes—in stark contrast to the **R/T-DOrg**. Recalling from Section 4, the episodes where organizational influence are most meaningful are those where a high-intensity fire is between the agents’ initial locations. While the **R/T-DOrg** approaches these cases with  $I_c$  transition shaping, the **XAutoOrgs** instead address them with state/action shaping (e.g., by delegating the northern cells to one agent and the southern to the other), which reduces the state space rather than increasing it.

## 6. RELATED WORK

One body of related work is organizational modeling languages (OMLs) such as MOISE<sup>+</sup> [6] and OMNI [15], among many others. OML research generally takes a problem-centric perspective, by creating a formal syntax that represents how to decompose and solve a target problem in terms of organizational roles for handling subproblems, role relationships for addressing subproblem interactions, etc. An organizational designer thus uses expertise about the target multiagent problem to specify a corresponding organization via an OML. The automated organizational design processes of Horling [5] and Sims [10] extend the OML work by searching for an optimal decomposition strategy, as configured from a library of problem-appropriate organizational goals, constraints, roles, agent capabilities, etc. Problem-centric approaches are particularly suited to open agent systems, where the organizational structure is built to address long-term problem needs, and

0.102	0.062	0.103	0.054	0.074	0.151	0.058	0.044	0.021	0.004
0.130	0.058	0.266	0.124	0.075	0.268	0.058	0.014	0.004	0.000
0.170	0.112	1.580	0.090	0.037	0.287	0.011	0.000	0.004	0.003
0.281	0.294	1.231	0.520	0.528	0.582	0.107	0.028	0.060	0.013
0.078	0.110	0.063	0.075	0.072	0.105	0.074	0.030	0.005	0.004

(a) Agent 1

0.003	0.008	0.056	0.186	0.272	0.210	0.203	0.303	0.196	0.082
0.000	0.002	0.031	0.127	0.192	0.384	0.202	0.764	0.194	0.126
0.003	0.001	0.001	0.017	0.033	0.111	0.104	1.485	0.213	0.237
0.030	0.041	0.072	0.111	0.228	0.309	0.285	0.571	0.215	0.089
0.005	0.003	0.021	0.043	0.049	0.039	0.081	0.106	0.057	0.035

(b) Agent 2

0	.5	.8	.5	0	0	0	0	0	.8
.5	.8	.8	.5	.5	0	.5	0	.5	0
.5	.8	.5	.8	.8	.5	.8	.8	.5	0
0	0	0	.5	0	0	.5	.5	0	.8
.5	0	.8	.5	.5	0	0	.8	.8	.8

(c) Cell Delays  $\delta_c$ Figure 5: Cumulative cell occupancy measures,  $\sum x_i(s_i, \cdot)$ , for each agent that our ODP calculated in response to the  $\delta_c$ 's in (c)

the organization persists even though agents performing particular roles can change.

In contrast to a problem-centric perspective, our work is agent-centric. We assume a group of agents in a multi-agent system already intends to cooperatively solve problems in their environment, and might even already be working together. The purpose of forming an organization, in this context, is to explicitly reason over and codify expectations about appropriate behaviors and interaction patterns in order to improve and streamline cooperation. Agent-centric approaches are thus advantageous for designing organizations to fit the objectives, capabilities, and limitations of a group of agents that will be cooperating over an extended time, even though the problems they face might vary. Hence, problem-centric and agent-centric approaches both emphasize the design of stable organizations, but differ in which aspects of the agents' task environment they treat as stable.

## 7. CONCLUSION

In this paper we presented a formal framework for specifying factored organizational influences and incorporating them into agents' decision models. We then argued that an ODP should restrict itself to R/T-D based influences, and empirically demonstrated the effectiveness of this design principle. Finally, we presented an automated ODP based upon this principle, and demonstrated how it creates sensible specifications that exploit structure within the domain. In the future, we plan to expand upon the functionality of our ODP, empowering it to make informed tradeoffs between restricting agent behaviors and the resulting potential loss of reward (e.g., by utilizing different, more sophisticated influence creation mechanisms). Additionally, we plan to relax some of the simplifying assumptions throughout this paper such as requiring that each agent always fully adopt its  $\theta_i$ , for example, with organizationally adept agents [3].

## 8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their thoughtful comments, and our collaborators at the University of Massachusetts for their consistently helpful feedback. This work was supported by NSF grant IIS-0964512.

## 9. REFERENCES

- [1] M. Babes, E. M. de Cote, and M. L. Littman. Social reward shaping in the prisoner's dilemma. In *AAMAS*, pages 1389–1392, 2008.
- [2] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control

of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

- [3] D. D. Corkill, C. Zhang, B. da Silva, Y. Kim, X. Zhang, and V. R. Lesser. Using annotated guidelines to influence the behavior of organizationally adept agents. In *COINS2012 Workshop at AAMAS*, 2012.
- [4] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *JAIR*, 19:399–468, 2003.
- [5] B. Horling and V. Lesser. Using quantitative models to search for appropriate organizational designs. *JAAMAS*, 16(2):95–149, 2008.
- [6] J. Hubner, J. Sichman, and O. Boissier. Developing organised multiagent systems using the MOISE<sup>+</sup> model: programming issues at the system and agent levels. *IJAOSE*, 1(3):370–395, 2007.
- [7] IBM. Ibm ilog cplex, 2012. See <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/>.
- [8] L. C. M. Kallenberg. *Linear Programming and Finite Markovian Control*. Mathematical Centre Tracts, 1983.
- [9] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for MDPs. In *ISAIM*, pages 531–539, 2006.
- [10] M. Sims, D. Corkill, and V. Lesser. Automated organization design for multi-agent systems. *JAAMAS*, 16(2):151–185, 2008.
- [11] J. Sleight and E. H. Durfee. A decision-theoretic characterization of organizational influences. In *AAMAS*, pages 323–330, 2012.
- [12] M. Stolle and D. Precup. Learning options in reinforcement learning. In *Lecture Notes in Computer Science*, pages 212–223, 2002.
- [13] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *AI*, 112:181–211, 1999.
- [14] P. Varakantham, J. Kwak, M. Taylor, J. Marecki, P. Scerri, and M. Tambe. Exploiting coordination locales in distributed POMDPs via social model shaping. In *ICAPS*, pages 313–320, 2009.
- [15] J. Vázquez-Salceda, V. Dignum, and F. Dignum. Organizing multiagent systems. *JAAMAS*, 11:307–360, 2005.
- [16] P. Velagapudi, P. Varakantham, K. Sycara, and P. Scerri. Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents. In *AAMAS*, pages 955–962, 2011.

# Rehearsal Based Multi-agent Reinforcement Learning of Decentralized Plans

Landon Kraemer  
The University of Southern Mississippi  
118 College Dr. #5106  
Hattiesburg, MS 39402  
Landon.Kraemer@eagles.usm.edu

Bikramjit Banerjee  
The University of Southern Mississippi  
118 College Dr. #5106  
Hattiesburg, MS 39402  
Bikramjit.Banerjee@usm.edu

## ABSTRACT

Decentralized partially-observable Markov decision processes (Dec-POMDPs) are a powerful tool for modeling multi-agent planning and decision-making under uncertainty. Prevalent Dec-POMDP solution techniques require centralized computation given full knowledge of the underlying model. Reinforcement learning (RL) based approaches have been recently proposed for distributed solution of Dec-POMDPs without full prior knowledge of the model, but these methods assume that conditions during learning and policy execution are identical. In practical scenarios this may not necessarily be the case, and agents may have difficulty learning under unnecessary constraints. We propose a novel RL approach in which agents are allowed to *rehearse* with information that will not be available during policy execution. The key is for the agents to learn policies that do not explicitly rely on this information. We show experimentally that incorporating such information can ease the difficulties faced by non-rehearsal-based learners, and demonstrate fast, (near) optimal performance on many existing benchmark Dec-POMDP problems. We also propose a new benchmark that is less abstract than existing problems and is designed to be particularly challenging to RL-based solvers, as a target for current and future research on RL solutions to Dec-POMDPs.

## Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Multiagent Systems*; I.2.8 [Problem Solving, Control Methods and Search]:

## General Terms

Algorithms, Experimentation, Performance

## Keywords

Multi-agent reinforcement learning, Dec-POMDPs

## INTRODUCTION

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a powerful model of decentralized decision making under uncertainty and incomplete knowledge. Many exact and approximate solution techniques have

been developed for Dec-POMDPs [9, 5, 7], but these approaches are not scalable because the underlying problem is provably NEXP-complete [3].

Recently, reinforcement learning (RL) techniques, particularly multi-agent reinforcement learning (MARL), have been applied [11, 2] to overcome other limitations of the Dec-POMDP solvers, viz., that they are centralized and model-based. That is, these traditional Dec-POMDP solvers compute the set of prescribed behaviors for agents centrally, and assume that a comprehensive set of model parameters are already available. While RL distributes the policy computation problem among the agents themselves, it essentially solves a more difficult problem, because it does not assume the model parameters are known a-priori. The hardness of the underlying problem translates to significant sample complexity for RL solvers as well.

One common feature of the existing RL solvers is that the learning agents subject themselves to the same constraints that they would encounter when executing the learned policies. In particular, agents assume that the environment states are hidden and the other agents' actions are invisible, in addition to the other agents' observations being hidden too. We argue that in many practical scenarios, it may actually be easy to allow learning agents to observe some otherwise hidden information *while they are learning*. We view such learning as a *rehearsal* – a phase where agents are allowed to access information that will not be available when executing their learned policies. While this additional information can facilitate the learning during rehearsal, agents must learn policies that can indeed be executed in the Dec-POMDP (i.e., without relying on this additional information). Thus agents must ultimately wean their policies off of any reliance on this information. This creates a principled incentive for agents to explore actions that will help them achieve this goal. Based on these ideas, we present a new approach to RL for Dec-POMDPs – Reinforcement Learning as a Rehearsal or RL<sub>aR</sub>, including a new exploration strategy. Our experiments show that this new approach can nearly optimally solve most existing benchmark Dec-POMDP problems with a low sample complexity. We raise the bar for RL solvers by proposing a new benchmark problem that is particularly challenging to RL solvers – robot alignment. We hope this problem will spur research to address the difficulties that it poses, leading to more competent RL solvers of Dec-POMDPs in the future.

## BACKGROUND

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with *AAMAS*, May 2013, St. Paul, Minnesota, USA.

## Decentralized POMDPs

We can define a Dec-POMDP as a tuple  $\langle n, S, A, P, R, \Omega, O \rangle$ , where:

- $n$  is the number of agents in the domain.
- $S$  is a finite set of (unobservable) environment states.
- $A = \times_i A_i$  is a set of joint actions, where  $A_i$  is the set of individual actions that agent  $i$  can perform.
- $P(s'|s, \vec{a})$  gives the probability of transitioning to state  $s' \in S$  when joint action  $\vec{a} \in A$  is taken in state  $s \in S$ .
- $R(s, \vec{a})$  gives the immediate reward the agents receive upon executing action  $\vec{a} \in A$  in state  $s \in S$ .
- $\Omega = \times_i \Omega_i$  is the set of joint observations, where  $\Omega_i$  is the finite set of individual observations that agent  $i$  can receive from the environment.
- $O(\vec{\omega}|s', \vec{a})$  gives the probability of the agents jointly observing  $\vec{\omega} \in \Omega$  if the current state is  $s' \in S$  and the previous joint action was  $\vec{a} \in A$ .

Additionally, for finite horizon problems, a horizon  $T$  is given that specifies how many steps of interaction the agents are going to have with the environment and each other. The objective in such problems is to compute a set of decision functions or *policies* – one for each agent – that maps the history of action-observations of each agent to its best next action, such that the joint behavior (note the transition, reward, and observation functions depend on *joint* actions) over  $T$  steps optimizes the total reward obtained by the team.

In Dec-POMDPs, it is generally assumed that agents cannot communicate their observations and actions to each other. These constraints are often present in real world scenarios, where communication may be expensive or unreliable. Consider a scenario in which a team of robots must coordinate to search a disaster area for survivors. In such a task, robots may need to disperse to efficiently cover the area and also may need to travel deep underneath rubble, both of which could interfere with wireless communication.

## Reinforcement Learning for Dec-POMDPs

Reinforcement Learning (RL) is a family of techniques applied normally to MDPs [8]  $\langle S, A, P, R \rangle$  (i.e., Dec-POMDPs with full observability and single agent). When states are visible, the task of an RL agent in a horizon  $T$  problem is to learn a *non-stationary* policy  $\pi : S \times t \mapsto A$  that maximizes the sum of current and future rewards from any state  $s$ , given by,

$$V^\pi(s^0, t) = E_P[R(s^0, \pi(s^0, t)) + R(s^1, \pi(s^1, t+1)) + \dots + R(s^{T-t}, \pi(s^{T-t}, T))]$$

where  $s^1, \dots, s^{T-t}$  are successive samplings from the distribution  $P$  following the Markov chain with policy  $\pi$ . A popular RL algorithm for such problems is  $Q$ -learning, which maintains an action-quality value function  $Q$  given by

$$Q(s, t, a) = R(s, a) + \max_{\pi} \sum_{s'} P(s'|s, a) V^\pi(s', t+1)$$

This quality value stands for the sum of rewards obtained when the agent starts from state  $s$  at step  $t$ , executes action

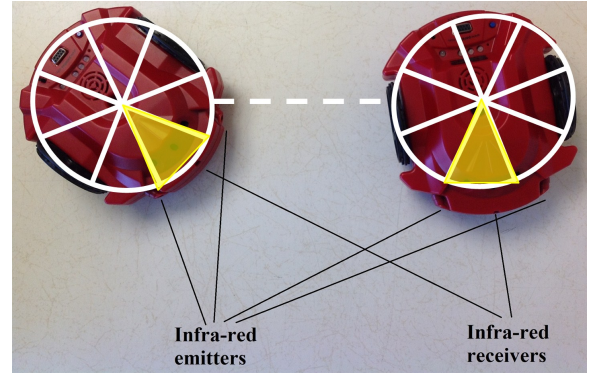


Figure 1: The Scribbler 2 robots featured in the robot alignment problem.

$a$ , and follows the optimal policy thereafter. These  $Q$ -values can be learned without prior knowledge of  $R$ ,  $P$ .

Multi-agent reinforcement learning (MARL) has recently been applied to Dec-POMDPs [11, 2], where a quality function  $Q(h_t, a)$  is learned, mapping the history of an agent's own past actions and observations for after  $t$  steps,  $h_t$ , and next actions ( $a$ ) to real values. While [11] assumes perfect communication among teammates in a special class of Dec-POMDPs (called ND-POMDPs), we assume indirect and partial communication among the agents, via a third party observer (but only while learning). Whereas the approach proposed in [2] has a large sample complexity, our results are all based on orders of magnitude fewer episodes in each setting. Furthermore, [2] proposes a turn taking learning approach where the non-learner's experience is wasted, whereas we propose a concurrent learning algorithm.

## MOTIVATING DOMAIN

Many benchmark problems have been developed for evaluation of Dec-POMDP solvers in the past [6]. The majority of these problems were developed to illustrate concepts pertaining to methods which *compute* solutions given the full Dec-POMDP model. Furthermore, these benchmarks tend to be rather abstract. We introduce a new, more concrete, Dec-POMDP benchmark problem that illustrates the difficulties faced by *learning-based* Dec-POMDP solvers which assume the model is unknown a priori.

Figure 1 shows two Scribbler 2 robots, each possessing two infra-red (IR) emitters and one IR receiver on their front faces – marked by the shaded sector as shown. The robots can rotate by some angle, emit or receive IR signals, and can also locomote. The IR system can be used for communication between robots, but only if the robots are facing each other.

A task that is often required in robotics is for robots to get aligned, i.e., face each other. The S2 robots, for instance, need to be aligned to communicate with IR. Not only is it important that the robots become aligned, but they must also *know* that they are aligned, since alignment is usually a precursor to some other behavior. This problem is more concrete than most Dec-POMDP benchmark problems because a computed policy can be readily implemented on real S2 robots.

## The Dec-POMDP Model

We model this problem using the Dec-POMDP framework as follows. The heading space of each agent is discretized into slices as shown in Figure 1. The state space of the Dec-POMDP model  $S$  is simply all possible combinations of joint headings (slices). It is assumed there is exactly one state, say  $a_0b_0$  (when both the shaded sectors touch the dotted line in Figure 1) in which agents are aligned.

Each agent is capable of four actions:  $A_i = \{done, no-op, emit-IR, turn-left\}$ . Observation sets are  $\Omega_i = \{IR, no-IR\}$ . An agent observes  $IR$  only if the other agent emitted  $IR$  and the state is  $a_0b_0$ , otherwise it observes  $no-IR$ . However, if both agents emit  $IR$  in state  $a_0b_0$ , neither observes  $IR$  due to the destructive interference of the  $IR$  waves. The transition function encodes state transitions when at least one agent turns-left, otherwise the state remains unchanged. Agents get maximum reward,  $R_{max}$ , if they jointly execute *done* in state  $a_0b_0$ . If agents execute the *done* action in any state other than  $a_0b_0$ , or if only one agent executes *done*, the agents receive  $-R_{max}$ . All other actions cost -1. The reward structure we place on the *done* action encourages agents to understand that they are aligned, and heavily penalizes incorrectly assuming they are aligned. Whenever an agent executes the *done* action, we assume that the state resets (i.e. a new state is chosen randomly from a uniform distribution). The initial belief  $b_0 \in \Delta S$  is also uniform.

## Difficulties for RL

Because  $P$ ,  $O$  are deterministic in the above problem, it is easily solved by traditional Dec-POMDP solvers, especially if the number of joint heading sectors is small. However, this problem embodies one of the most challenging scenarios for RL, viz., *combination lock*. A combination lock represents a scenario where an agent must execute a specific sequence of actions (a *combination*) to reach a desirable state or unlock high rewards, and the reward structure is such that the agent is not naturally guided to this state by exploring greedily. In a multi-agent scenario this is further compounded because the agents must execute a *joint* combination. With independent random exploration, it would be extremely difficult for them to hit the right joint combination even once, let alone learn good  $Q$  estimates by repeatedly reaching the goal.

In the robot alignment problem, the agents must *reach* a particular goal state (i.e.  $a_0b_0$ ), *probe* to understand that they have reached the goal, and then *coordinate* to signal that they are aligned. This constitutes a combination lock of significant proportions. Contrast this with problems such as DecTiger and Box Pushing. In DecTiger, agents do not need to reach a particular state and are only tasked with probing to understand which door conceals the tiger, and coordinating to open a door. In Box Pushing, on the other hand, agents do not need to probe to receive observations and are tasked with navigating the state space and coordinating to push boxes.

One considerable source of difficulty in the robot alignment problem is that agents must coordinate in order to receive informative observations about the state. If neither agent emits  $IR$  or if both agents emit  $IR$ , the agents will not observe  $IR$ , regardless of the state. Thus, at least two steps are required for both agents to be able to determine if a particular state is the goal, with a different agent emitting  $IR$  in each step. Contrast this with DecTiger. In DecTiger, agents must also coordinate to gather information about the

state in DecTiger; however, in DecTiger, to gain information both agents execute the same action, *listen*, and *both* receive information. This is not the case in robot alignment as there are two different roles for information gathering, the  $IR$ -emitter and the receiver, with the emitter receiving no information. Also, the reward structure of DecTiger is such that agents will automatically prefer *listen* when they are unsure of the state because it is a relatively safe action. In robot alignment, however, agents are not guided by rewards to prefer any action involved in information gathering, much less to coordinate with the other agent.

Just as the immediate rewards do not guide the agents to engage in information gathering, they also do not induce a non-uniform preference over the state space. Agents will receive the same immediate reward executing *no-op* every step as they will executing *turn left* every step unless *done* is executed properly. Thus, in order to learn that becoming aligned is worthwhile, agents must first become aligned, be able to recognize that they are aligned, and then signal that they are aligned.

## REINFORCEMENT LEARNING AS A REHEARSAL

RL has previously been applied in [11, 2] to enable agents to learn mappings from local action-observation histories to next actions. While their assumptions about local information differ – Banerjee et al. assume that agents only observe their own actions and observations, where Zhang and Lesser assume that agents observe those of their team mates as well – both methods assume that only local information is available during learning. That is, both assume that learning must occur under execution conditions. However, we argue this assumption is not always necessary and may make learning unnecessarily difficult.

There are many scenarios for which the training conditions need not be as strict as execution conditions. Consider, for example, the multi-robot rescue scenario mentioned previously. If those robots were trained in a simulator, rather than in the field, they could easily access information that would be hidden in a true rescue scenario. Consider also the robot alignment problem. Since the dynamics of the alignment problem do not depend on the environment, robots could easily be trained in a laboratory setting, where a computer connected to an overhead camera could relay states and others’ actions to the robots. Note that providing robots with this same information in a typical execution environment would be unwieldy as some sort of third robot with a camera would have to follow the robots around, requiring more coordination than the original problem. Clearly, then, while constraints on execution conditions may be justified, these constraints need not always be applied to the training environment.

To this end, we explicitly break up a problem into distinct “learning” and “execution” phases. We treat the MARL problem as a rehearsal *before* a final stage performance, so to speak. The inspiration comes from real life; while actors train together to produce a coordinated performance on stage, they do not necessarily subject themselves to the same constraints during rehearsal. For instance, actors routinely take breaks, use prompters, receive feedback, practice separately, etc. during rehearsals – things that are not available/doable during the final stage performance. Similarly



in MARL, while the agents must learn distributed policies that can be executed in the target task (the stage of performance), they do not need to be constrained to the exact setting of the target task while learning.

In this paper, we assume that agents rehearse under the supervision of a third party observer that can convey to them the hidden state *when they are learning/rehearsing, but not when they are executing the learned policies*. This observer also tells the agents what the other agents' last actions were. However, this observer cannot observe the agents' observations and hence cannot cross-communicate them at any time. We call the extra information available to a learner during rehearsal,  $(s \in S, a_- \in A_-)$  – i.e., the hidden state and the others' actions – the *rehearsal features*. *The key challenge is that agents still must learn policies that will not rely on these external communications, because the rehearsal features will not be visible at the time of executing the policies.* Therefore, the policies returned by our approach must be in the same language as those returned by the Dec-POMDP solvers. We call our algorithm (described in the next section) *Reinforcement Learning as a Rehearsal*, or RLaR.

## Model Estimation

RLaR, like previous RL approaches, assumes that the underlying Dec-POMDP model is not known a priori (even by the external observer); however, since the external observer informs agents of the state,  $s$ , and joint action,  $\vec{a}$  at each step, agents can maintain their own estimates of the transition function  $\hat{P}(s'|s, \vec{a})$ , the reward function  $\hat{R}(s, \vec{a})$ , the initial distribution over states  $b_0 \in \Delta S$ , and an *individual* observation probability function  $\hat{O}_i(\omega_i|s', \vec{a})$ . That is, the agents can maintain their own internal models of the environment's dynamics.

To be clear, the models the agents maintain here are *partial models* because they do not model the joint observation distribution. This is because agents do not have access to the observations of others. If *two-way* communication between the agents and the external observer was assumed, then the agents could report their observations to the external observer to be distributed to all agents. This would allow agents to observe joint observations, and agents could then estimate the full Dec-POMDP model. However, RLaR assumes that only a *one-way* communication channel exists from the observer to the agents.

While two-way communication may be available in some scenarios, we focus on one-way communication here for a number of reasons. The one-way communication paradigm requires only the observer be equipped to both send and receive (agents need only receive), but two-way communication requires the observer and each agent to be equipped with apparatus to both send and receive, which can be costly. Furthermore, the two-way paradigm requires that each agent send a message (its individual observation) to the observer before the observer can broadcast, which can be time-consuming.

If two-way communication was assumed, then reinforcement learning might be unnecessary because agents could sample the environment to construct an estimate of the full model and could then apply an offline planner such as GMAA\*-ICE [7]. Without full model estimates (i.e. without two-way communication), it seems unlikely that offline planners could be used to replace RL, since agents have no

concept of the observation dynamics of other agents. In the robot alignment problem, for instance, an agent would have no way of knowing under what circumstances its counterpart could observe IR. It does not know if the other agent has one sensor, no sensors, ten sensors, faulty sensors, etc. Furthermore, it has no idea that its own IR beam is required for the other agent to observe IR.

## The RLaR Algorithm

In addition to model estimation, the rehearsal features also allow agents to treat the problem as the fully-observable MDP  $\langle S, A, \hat{P}, \hat{R} \rangle$ , and learn an MDP policy via action-quality values given by

$$Q(s, t, \vec{a}) = \hat{R}(s, \vec{a}) + \sum_{s' \in S} \hat{P}(s'|s, \vec{a}) \max_{\vec{a}' \in A} Q(s', t+1, \vec{a}') \quad (1)$$

Obviously, this MDP policy will not be useful during policy execution as it assumes full observability. Instead, this policy could be a useful starting point – an idea studied before as *transfer learning* [10]. In the robot alignment problem, having learned the MDP policy, agents will have an understanding of the mechanics of aligning themselves; however, they will still have no understanding of how to coordinate to detect alignment when the third party observer disappears. But the MDP policy tells the agents that jointly executing *done* in state  $s = a_0 b_0$  is desirable, and as a result, they will have the incentive to learn how to detect alignment when  $s$  is not available. In general, RLaR agents will have an incentive to learn to predict the rehearsal features  $(s, a_-)$ , which creates a new, principled exploration strategy for such learners, that is unique to partially observable domains. We shall expand on this exploration strategy in the next section.

We break the rehearsal phase into 2 successive stages, where the MDP policy is learned as above in the first stage. In the second stage, agents learn joint action values given  $s$  and the current observable history  $h_t$ , i.e.,  $Q(s, h_t, \vec{a})$ , while also learning the correlation between the (ultimately forbidden) rehearsal features and the observable history. The MDP  $Q$  values are used to initialize the  $Q(s, h_t, \vec{a})$  values as

$$Q(s, h_t, \vec{a}) \leftarrow Q(s, t, \vec{a}).$$

Then the  $Q(s, h_t, \vec{a})$  are learned as

$$Q(s, h_t, \vec{a}) = \hat{R}(s, \vec{a}) + \sum_{\omega \in \Omega_i} \tilde{P}(\omega|s, \vec{a}) \max_{a' \in A_i} Q(h_{t+1}, a'), \quad (2)$$

where  $\vec{a} = \langle a, a_- \rangle$  with  $a_-$  representing the other agents' actions,  $h_{t+1}$  is the concatenation of  $h_t$  with  $(a, \omega)$ , i.e.  $(h_t, a, \omega)$ , and  $Q(h_{t+1}, a')$  is explained later.  $\tilde{P}(\omega|s, \vec{a})$  is calculated using the agent's internal model parameters via

$$\tilde{P}(\omega|s, \vec{a}) = \alpha \sum_{s' \in S} \hat{P}(s'|s, \vec{a}) \hat{O}_i(\omega|s', \vec{a}) \quad (3)$$

where  $\alpha$  is a normalization factor.

$Q(s, h_t, \vec{a})$  gives the immediate reward agents will receive when executing  $\vec{a}$  in state  $s$  plus the future reward agents can expect when the next state  $s'$  and other agent's next action  $a'_-$  cannot be observed. This future reward is encapsulated in the quantity  $Q(h_{t+1}, a')$  used in the update rule of equation 2, and maintained by the learners alongside  $Q(s, h_t, \vec{a})$  values.  $Q(h_t, a)$  values are the real objective of RLaR since they are independent of rehearsal features and can be used during the execution phase.

As with RL algorithms used for Dec-POMDPs,  $Q(h_t, a)$  gives the expected reward of executing individual action  $a$  having observed individual action-observation history,  $h_t$ . However, instead of learning these values, we estimate them based on the learned  $Q(s, h_t, \vec{a})$  values as follows:

$$Q(h_t, a) = \sum_{s \in S} \sum_{a_- \in A_-} P(s, a_- | h_t) Q(s, h_t, \vec{a}). \quad (4)$$

$P(s, a_- | h_t)$  gives the probability (or the learner’s *belief*) that after observing  $h_t$ , the state will be  $s$  and the other agent will execute  $a_-$ . This is essentially the agent’s prediction of the rehearsal features based on observable history. Agents could estimate this probability directly via sampling; however, we note that  $P(s, a_- | h_t) = P(a_- | h_t, s)P(s | h_t)$ . Therefore, agents can estimate  $P(a_- | h_t, s)$  by sampling, but *propagate* the belief  $P(s | h_t)$  using

$$P(s' | h_t) = \sum_{s \in S} \sum_{a_- \in A_-} \hat{O}_i(\omega | s', a) \hat{P}(s' | s, \vec{a}) P(s, a_- | h_{t-1}) \quad (5)$$

where  $h_t = (h_{t-1}, a, \omega)$ . This allows an agent to incorporate its model estimate into the calculation, which can improve belief estimates because while  $P(a_- | h_t, s)$  changes as the agents learn, the underlying internal model tracks stationary distributions.

The RLaR algorithm is outlined in Algorithm 1. Each agent executes a separate instance of RLaR, but concurrently with other agents. Since the Dec-POMDP framework assumes that actions are synchronized, calls to EXECUTEACTION represent synchronization points. EXECUTEACTION returns the next state  $s'$ , the last joint action  $\vec{a}$ , the agent’s own observation  $\omega$ , and the reward  $r$ . This feedback includes rehearsal features from the third party observer. Upon receiving this feedback, each agent updates its model  $\langle S, A, \hat{P}, \hat{R}, \Omega_i \rangle$  as well as its model of the other agent’s action  $P(a_- | h_t, s)$ . In the first stage agents do not require a model of other agent’s actions; however, we have found it useful to sample  $P(a_- | t, s)$  to replace missing  $P(a_- | h_t, s)$  values in stage 2. UPDATEQ( $s, t, \vec{a}$ ) implements equation 1, while UPDATEQ( $s, h, \vec{a}$ ) implements equations 3, 2, 4, 5 in that order.

---

**Algorithm 1** RLaR( $mdp\_episodes, total\_episodes$ )

---

```

1: for  $m = 1 \dots total\_episodes$  do
2:    $s \leftarrow \text{GETINITIALSTATE}()$ 
3:    $h \leftarrow \emptyset$ 
4:   for  $t = 1 \dots T$  do
5:     if  $m \leq mdp\_episodes$  then
6:        $a \leftarrow \text{SELECTACTIONMDP}()$ 
7:        $(s', \vec{a}, \omega, r) \leftarrow \text{EXECUTEACTION}(a)$ 
8:        $\text{UPDATEMODEL}(s, \vec{a}, s', \omega, r)$ 
9:        $\text{UPDATEQ}(s, t, \vec{a})$ 
10:    else
11:       $a \leftarrow \text{SELECTACTION}()$ 
12:       $(s', \vec{a}, \omega, r) \leftarrow \text{EXECUTEACTION}(a)$ 
13:       $\text{UPDATEMODEL}(s, \vec{a}, s', \omega, r)$ 
14:       $\text{UPDATEQ}(s, h, \vec{a})$ 
15:    end if
16:     $s \leftarrow s'$ 
17:     $h \leftarrow (h, a, \omega)$ 
18:  end for
19: end for

```

---

## Action Selection

One common method of action selection in reinforcement learning is the  $\epsilon$ -greedy approach [8], in which an agent chooses its action randomly with some probability  $\epsilon$  (i.e., explores randomly), but otherwise greedily chooses the action which maximizes the expected value, i.e.,  $\arg \max_a Q(h_t, a)$ . However, since the learners intend to output policies that are independent of the rehearsal features, there is a principled incentive to explore actions that help predict the rehearsal features. Thus, we propose an additional exploration criterion, beyond  $\epsilon$ -greedy, that values information gain rather than expected reward.

Having observed history  $h_t$ , a learner can pick action  $a_{\text{explore}}$  with some probability, given by

$$a_{\text{explore}} = \arg \max_{a \in A_i} \sum_{\omega \in \Omega_i} P(\omega | h_t, a) E(h_t, a, \omega) \quad (8)$$

where  $P(\omega | h_t, a) = \sum_{s, a_-} \tilde{P}(\omega | s, \vec{a}) P(s, a_- | h_t)$ , calculated from equation 3 and its prediction of the rehearsal features. While  $E(h_t, a, \omega)$  measures the entropy of the prediction distribution that would result if action  $a$  is executed and observation  $\omega$  is received, equation 8 measures the *expected* entropy accounting for the fact that action  $a$  has not been executed yet, and observation  $\omega$  has not actually been received.  $E(h_t, a, \omega)$  is estimated as

$$E(h_t, a, \omega) = - \sum_{s \in S} \sum_{a_- \in A_-} P(s, a_- | h_t, a, \omega) \log P(s, a_- | h_t, a, \omega)$$

where  $P(s, a_- | h_t, a, \omega)$  is estimated as  $P(s, a_- | h_t, a, \omega) = \alpha \tilde{P}(\omega | s, \vec{a}) P(s, a_- | h_t)$ , where  $\alpha$  is a normalization factor.

One of the difficulties the robot alignment problem poses is that agents have no incentive to gather information until they have gathered information frequently enough to learn that it is worthwhile. Therefore, we expect this exploration criterion – called *entropy based exploration* – to be particularly beneficial to robot alignment. However, given the general need of a RLaR learner to achieve independence from rehearsal features, we expect it to be valuable in all Dec-POMDP problems.

In our experiments, we use an upper-confidence bound (UCB) [1] approach for SELECTACTIONMDP; however, other action selection approaches might be applied successfully too. Under the UCB approach, agents select joint actions via

$$\arg \max_{\vec{a} \in A} Q(s, t, \vec{a}) + (R_{\max} - R_{\min}) C \sqrt{\left( \frac{2 \log n_{s,t}}{n_{s,t,\vec{a}}} \right)}, \quad (9)$$

for some constant  $C$ , where  $n_{s,t}$  is the number of times the state  $s$  has been encountered at step  $t$ , and  $n_{s,t,\vec{a}}$  is the number of times  $\vec{a}$  has been executed in state  $s$  at step  $t$ . The term added to  $Q(s, t, \vec{a})$  is an exploration bonus that balances exploration with exploitation with optimal asymptotic behavior.

## Ideal Solution

In this work, we only evaluate the convergence of RLaR empirically by comparing an agents learned Q-Values,  $Q_i(h_i, a)$ , against a set of target (optimal) Q-Values,  $Q_i^*(h_i, a^*)$ . In principle, we can calculate these  $Q^*$ -values by solving a constraint optimization problem with real variables  $Q_i^*(h_i, a)$ ,  $P(s, h_{-i} | h_i)$ , and binary variables  $I(h_i, a)$  for every agent  $i$ . The  $I(h_i, a)$  variables essentially describe the agents’ policies, i.e.  $I(h_i, a) = 1$  implies that agent  $i$  will execute action



$$Q_i^*(h_i, a_i) = \sum_{s \in S, a_{-i} \in A_{-i}, h_{-i} \in H_{-i}} I(h_{-i}, a_{-i}) P(s, h_{-i} | h_i) \cdot \left[ R(s, \vec{a}) + \sum_{\omega_i \in \Omega_i} P(\omega_i | s, \vec{a}) \max_{b \in A_i} Q_i^*((h_i, a_i, \omega_i), b) \right] \quad (6)$$

$$P(s', (h_{-i}, a_{-i}, \omega_{-i}) | (h_i, a_i, \omega_i)) = I(h_{-i}, a_{-i}) \sum_{s \in S} P(s', \omega_{-i} | s, \vec{a}, \omega_i) P(s, h_{-i} | h_i) \quad (7)$$

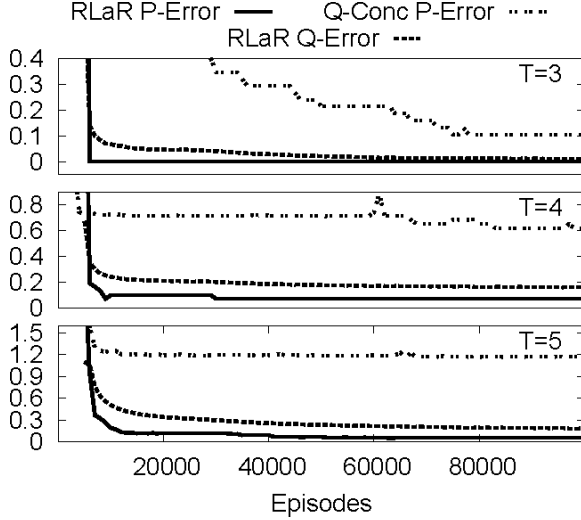


Figure 2: Dectiger

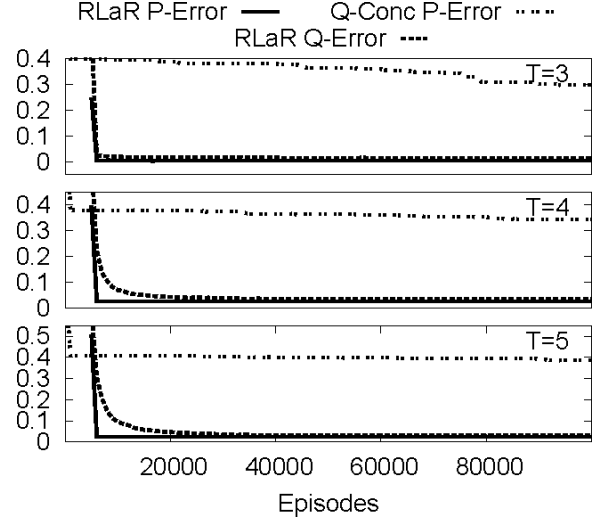


Figure 3: Recycling

$a$  after observing history  $h_i$ . The  $I(h_i, a)$  are constrained to be consistent with the  $Q^*$ -values so that  $\arg \max_{b \in A_i} Q_i^*(h_i, b) \neq a_i \implies I(h_i, a_i) = 0$ , and they are further constrained so that exactly one action must be chosen for each history, i.e.  $\sum_{a \in A_i} I(h_i, a) = 1$ .

The  $P(s, h_{-i} | h_i)$  variables are constrained by equation 7. A given  $P(s, h_{-i} | h_i)$  describes the probability that agent  $-i$  encountered  $h_{-i}$  and the state is  $s$  after agent  $i$  encountered  $h_i$ . Note the value of a given  $P(s, h_{-i} | h_i)$  depends on the policy of agent  $-i$  but not agent  $i$ 's policy.

Finally, the  $Q_i^*(h_i, a)$  values are constrained by equation 6, and the objective of the target optimization problem is given by  $\max \sum_{i=1}^n \sum_{a \in A_i} I(h_0, a) Q_i^*(h_0, a)$ , where  $h_0$  is the initial (empty) history.

Solving this optimization problem will yield an optimal policy via the  $I$  variables as well as the optimal  $Q$ -values corresponding to that policy. Unfortunately, this cannot be any easier than solving the Dec-POMDP, and furthermore, this optimization problem may have multiple solution sets corresponding to multiple optimal policies. The important thing to note is that while there may be multiple optimal policies, there is *exactly one set of  $Q^*$  values for a given optimal policy*.

## EXPERIMENTS

We evaluated the performance of RLaR for the robot alignment problem as well as four well-known benchmark problems: Dectiger, Meeting on a 2x2 Grid (GridSmall), Recycling Robots, and Box Pushing [6].

In the particular variant of the robot alignment prob-

lem<sup>1</sup> we used, agent 1's heading space is discretized into two slices and agent 2's heading space discretized into four slices. While noise could be added to transition and observation distributions, we assumed no noise in either, and set  $R_{\max}$  to 100.

For RLaR we used a combination of entropy-based exploration (equation 8) with a probability of 0.25 and  $\epsilon$ -greedy exploration with  $\epsilon = 0.005$ .

For each domain and horizon value  $T$ , we evaluated the performance of RLaR over 20 runs of 100000 episodes each. For the Box Pushing domain we allocated 10000 of those episodes to stage 1, and for the others we used 5000 stage 1 episodes. We used more episodes for the Box Pushing domain because it has an order of magnitude more states than the other benchmarks studied. While we do not assume that the model is known a priori, the size of the state-space can arguably be discerned without full knowledge of the model.

For comparison purposes, we also report the performance of concurrent Q-Learning without rehearsal, i.e. agents that learn  $Q(h_t, a)$  using only local information. This setting is labeled as "Q-Conc". No stage 1 episodes were allocated for Q-Conc because an MDP policy cannot be learned without access to the rehearsal features. Furthermore, while an initialization phase for Q-Conc was studied in [4], it was unclear that such an initialization improved results. We use  $\epsilon$ -greedy exploration for Q-Conc with  $\epsilon = 0.05$  which gave the best results for Q-Conc.

As noted previously, we only study the convergence of RLaR empirically by comparing the learned  $Q$ -Values  $Q_i(h_i, a)$

<sup>1</sup>See <http://www.cs.usm.edu/~banerjee/alignment> for more details, including problem definition files.

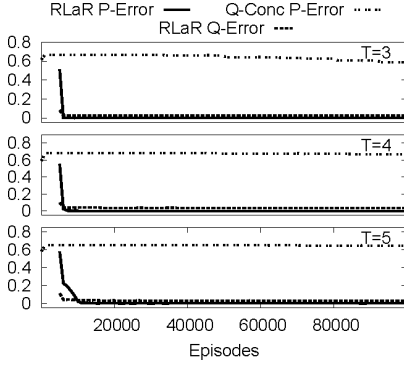


Figure 4: GridSmall

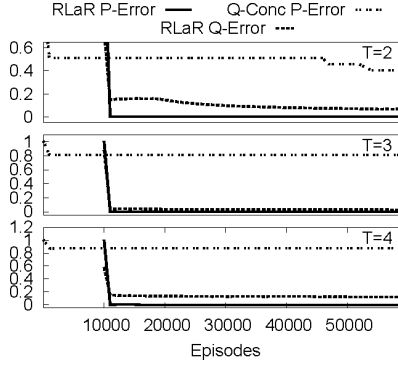


Figure 5: Box Pushing

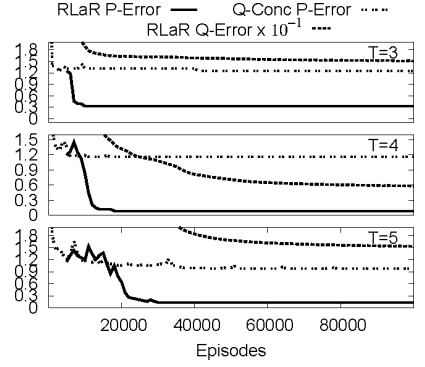


Figure 6: Robot Alignment

to the ideal  $Q_i^*(h, a)$  values described in equation 6. Despite the existence of multiple solution sets, if we already have an optimal policy,  $\pi$ , we can easily find a unique set of  $Q^*$  values by setting the  $I$  to be consistent with  $\pi$ . Note that when calculating  $Q^*$  this way, if a given history  $h_i$  is inconsistent with  $\pi$ ,  $Q_i^*(h_i, \cdot)$  will be undefined. Thus, we compare the set of  $Q$ -values only for those histories which are consistent with the known optimal policy. Some domains, such as the robot alignment and GridSmall have multiple optimal policies, which necessarily have different sets of consistent histories, and so, in order to measure convergence more accurately, when possible, we compared to the particular optimal policy that each run converged to. For runs that did not converge to an optimal policy we chose an optimal comparison policy arbitrarily. In the plots, we report this value as  $Q\text{-Error} = \sum_{h \in H^C} \frac{|Q_i^*(h, a^*) - Q_i(h, a^*)|}{|H^C|}$ , where  $H^C$  is the set of histories consistent with the comparison policy and  $a^*$  for a given history  $h$  is  $\arg \max_{a \in A_i} Q_i^*(h, a)$ .

$Q\text{-Errors}$  only indicate the quality of the learner’s value function for the histories in  $H^C$ . However, a learner potentially learns  $Q$ -values for a much larger set of histories,  $H^L$ , and poor learning on  $H^L - H^C$  can produce poor policies even when  $Q\text{-Errors}$  are low. Therefore, it is important to also evaluate the quality of the policies actually produced by RLaR. In order to measure the policy quality for RLaR and Q-Conc after each episode, we found the error of agents’ current policy relative to the known optimal policy value (the comparison policy), i.e.  $\frac{|v_{pol} - v_{opt}|}{|v_{opt}|}$ . We refer to this measure in our plots as the “policy error” or “P-Error”.

From Figures 2–6 we can see that RLaR converges rapidly (usually in fewer than 20000 episodes) to (near) optimal policies ( $< 0.1$  policy error) on all domains and horizons except for robot alignment  $T = 3, 5$ . Q-Conc performs much worse than RLaR in all domains, which highlights the impact of incorporating non-local information into the learning process, i.e., the efficacy of rehearsal based learning. Also note that, Q-Conc (and to a lesser extent, RLaR) performs substantially worse on the robot alignment problem than it does in the other domains, which reinforces our claim that robot alignment is a difficult problem for learning-based approaches.

The  $Q\text{-Error}$  was relatively small in all domains except Robot Alignment, suggesting that the learned  $Q$ -Values indeed converged to values close to the ideal  $Q^*$  values. Note that in robot alignment, multiple optimal policies exist for each horizon, and many runs did not converge to a clear

optimal policy, so the arbitrary comparison policy (as mentioned before) contributed to large  $Q\text{-Error}$ . Furthermore, due to the “combination lock” nature of the robot alignment problem, when agents fail to converge to the optimal solution, the histories in  $H^C$  tend to be unexplored, so the  $Q\text{-Error}$  is especially high for those runs.

## CONCLUSIONS

We have presented a novel reinforcement learning approach, RLaR with a new exploration strategy suitable for partially observable domains, for learning Dec-POMDP policies when agents are able to rehearse using information that will not be available during execution. We have shown that RLaR can learn near optimal policies for several existing benchmark problems, with a low sample complexity.

We have also introduced a new benchmark problem, robot alignment, which is easy for centralized solvers that *compute* policies given the full Dec-POMDP model, yet quite difficult for reinforcement-learning based Dec-POMDP solvers. Our hope is that this benchmark problem will help motivate more sophisticated RL-based Dec-POMDP solution techniques in the future.

## ACKNOWLEDGMENTS

We thank Matthijs Spaan for computing the optimal values for the alignment problem. This work was supported in part by the U.S. Army under grant #W911NF-11-1-0124.

## REFERENCES

- [1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.
- [2] B. Banerjee, J. Lyle, L. Kraemer, and R. Yellamraju. Sample bounded distributed reinforcement learning for decentralized pomdps. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*, pages 1256–1262, Toronto, Canada, July 2012.
- [3] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.
- [4] L. Kraemer and B. Banerjee. Informed initial policies for learning in dec-pomdps. In *Proceedings of the*

- AAMAS-12 Workshop on Adaptive Learning Agents (ALA-12)*, pages 135–143, Valencia, Spain, June 2012.
- [5] F. A. Oliehoek, M. T. J. Spaan, J. S. Dibangoye, and C. Amato. Heuristic search for identical payoff bayesian games. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, pages 1115–1122, Toronto, Canada, 2010.
  - [6] M. Spaan. Dec-POMDP problem domains and format. <http://users.isr.ist.utl.pt/~mtjspaan/decpomdp/>.
  - [7] M. T. J. Spaan, F. A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 2027–2032, Barcelona, Spain, 2011.
  - [8] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
  - [9] D. Szer and F. Charpillet. Point-based dynamic programming for dec-pomdps. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 1233–1238, Boston, MA, 2006.
  - [10] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.
  - [11] C. Zhang and V. Lesser. Coordinated multi-agent reinforcement learning in networked distributed POMDPs. In *Proc. AAAI-11*, San Francisco, CA, 2011.

# Counterfactual Regret Minimization for Decentralized Planning

Bikramjit Banerjee

The University of Southern Mississippi  
118 College Dr. #5106  
Hattiesburg, MS 39402  
Bikramjit.Banerjee@usm.edu

Landon Kraemer

The University of Southern Mississippi  
118 College Dr. #5106  
Hattiesburg, MS 39402  
Landon.Kraemer@eagles.usm.edu

## ABSTRACT

Regret minimization is an effective technique for almost surely producing Nash equilibrium policies in coordination games in the strategic form. Decentralized POMDPs offer a realistic model for sequential coordination problems, but they yield doubly exponential sized games in the strategic form. Recently, counterfactual regret has offered a way to decompose total regret along a (extensive form) game tree into components that can be individually controlled, such that minimizing all of them minimizes the total regret as well. However, a straightforward extension of this decomposition in decentralized POMDPs leads to a complexity exponential in both the joint action and joint observation spaces. We present a more tractable approach to regret minimization where the regret is decomposed along the nodes of agents' policy trees that yields a complexity exponential only in the joint observation space. We present an algorithm, REMIT, to minimize regret by this decomposition and prove that it converges to a Nash equilibrium policy in the limit. We also use a stronger convergence criterion with REMIT, such that if this criterion is met then the algorithm must output a Nash equilibrium policy in finite time. We found empirically that in every benchmark problems that we tested, this criterion was indeed met and (near) optimal Nash equilibrium policies were achieved.

## Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Multiagent Systems*; I.2.8 [Problem Solving, Control Methods, and Search]:

## General Terms

Algorithms, Experimentation, Performance

## Keywords

Game and decision theory, Decentralized partially observable Markov decision processes

## INTRODUCTION

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a powerful and realistic model for multi-agent coordination problems. On the one hand

many exact solvers have been developed for Dec-POMDPs, which yield the optimal Nash equilibrium solution, but are highly inefficient due to the inherent complexity of the problem [18, 17]. On the other hand, many approximate solvers are known that produce high quality solutions more efficiently, but generally do not guarantee that the returned solution will be a Nash equilibrium [15, 9]. There also exist *local search* techniques, which aim for a locally optimal Nash equilibrium solution [11, 13]. Stability of the solution is not important for the exact or approximate solvers, since they are evaluated by the quality of the solution produced, and any agent's incentive for deviation at policy execution time can only improve the group utility. However, stability is a key question for local search techniques. Since a local optimum is their logical objective, it is useful to know whether they reach such a solution. Our work falls in the category of local search techniques. In this paper we examine the application of *regret minimization* [19] – an effective (and often efficient) class of techniques from the literature on learning in games – to Dec-POMDPs.

Recently, regret minimization has been shown to almost surely achieve Nash equilibrium solution in strategic form coordination games [10]. We argue that while Dec-POMDPs can be represented as strategic form coordination games, the resulting strategy space would be hopelessly intractable for the application of regret-based techniques. On the other hand, a recent approach called counterfactual regret which has been very successful in poker variants, has shown a more compact way to minimize regret, albeit in extensive form competitive game trees. The main idea is to decompose overall regret into components that can be minimized independently. A straightforward extension of this method to Dec-POMDPs would decompose total regret along the histories of action-observations for each agent, which would yield a complexity exponential in both joint action and joint observation spaces. We present a more compact way to decompose total regret in Dec-POMDPs – one where the individually controllable components are defined at each node of each agent's policy tree. We show that this decomposition is indeed valid, i.e., minimizing all component regrets will indeed minimize the total regret. We present an algorithm, REMIT, for minimizing regret components, with a complexity that is exponential only in the joint observation space, and show that it converges to a Nash equilibrium policy in the limit. We also present a stronger convergence criterion, that, if satisfied, will yield a Nash equilibrium in finite time. We show experimentally that this criterion is indeed satisfied in a range of benchmark problems, with REMIT producing

---

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with *AAMAS*, May 2013, St. Paul, Minnesota, USA.

(near) optimal Nash equilibrium policies.

A major motivation for investigating regret based techniques is the fact that they are often precursors to effective sample based reinforcement learning (RL) algorithms. A most recent RL algorithm for Dec-POMDPs has agents learning alternately, and has a complexity exponential in both joint action and joint observation spaces for each agent's learning phase [2]. Therefore, a sampling based variant of REMIT, where agents can also update simultaneously, holds the promise of being relatively more scalable. This paper establishes some of the theoretical substrate on which such a distributed learning approach could be built.

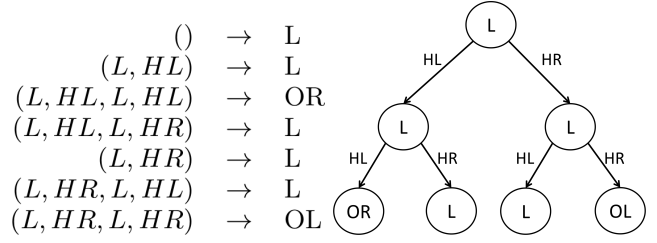
## BACKGROUND

### Dec-POMDPs

A Decentralized POMDP (Dec-POMDP) is defined as a tuple  $\langle k, S, b_0, A, P, R, \Omega, O \rangle$ , where:

- $k$  is the number of agents in the system.
- $S$  is a finite set of (unobservable) environment states.
- $b_0 \in \Delta(S)$  is the initial belief state.
- $A = \times_i A_i$  is a set of joint actions, where  $A_i$  is the set of individual actions that agent  $i$  can execute.
- $P(s'|s, a)$  gives the probability of transitioning to state  $s' \in S$  when joint action  $a \in A$  is taken in state  $s \in S$ .
- $R : S \times A \rightarrow \mathbb{R}$ , where  $R(s, a)$  gives the immediate reward the agents receive upon executing joint action  $a \in A$  in state  $s \in S$ .
- $\Omega = \times_i \Omega_i$  is the set of joint observations, where  $\Omega_i$  is the finite set of individual observations that agent  $i$  can receive from the environment.
- $O(\omega|s', a)$  gives the probability of the agents jointly observing  $\omega \in \Omega$  if the current state is  $s' \in S$  and the previous joint action was  $a \in A$ .

Additionally, a horizon  $T$  is also specified for finite horizon problems, and the transition and observation probabilities are often jointly represented as  $P(s', \omega|s, a)$ . The reward function  $R$ , transition model  $P$ , and observation model  $O$  are defined over joint actions and/or observations, which forces the agents to coordinate. The goal of the Dec-POMDP problem is to find a set of policies – one for each agent,  $\pi_i$  – that maximizes the total expected reward over  $T$  steps of interaction, given that the agents cannot communicate their observations and actions to each other. For finite horizon problems,  $\pi_i$  is a mapping from the histories of action-observation pairs of agent  $i$  to actions in  $A_i$ . A  $t$ -step history is represented as  $h_t = (a^0, \omega^0, \dots, a^{t-1}, \omega^{t-1})$ . Figure 1 shows the example of an agent's policy in the DEC-TIGER domain for horizon  $T = 3$ , with the policy represented as a mapping from histories to actions on the left, and equivalently as a policy tree on the right. In this paper we will refer to a policy  $\pi_i$  in the tree form. The problem of finding an optimal joint policy (i.e., set of trees, one for each agent) has been proven to be NEXP-complete [3].



**Figure 1: Two equivalent representations of an example policy for the single-agent tiger problem. The goal is to open a door that conceals treasure, instead of one that hides a tiger. The agent can execute actions listen (L), open right door (OR) or open left door (OL). It can hear the tiger's growl behind the left door (HL) or the right door (HR).**

### Strategic Games and Regret

A game in strategic form has three components: a set of players  $\{1, 2, \dots, k\}$ , a *pure strategy* space  $S_i$  for each player  $i$ , and payoff functions  $u_i$  that specifies the  $i$ th player's Von Neumann-Morgenstern utility  $u_i(s)$  for each strategy profile  $s = (s_1, s_2, \dots, s_k)$ . In this paper, we are particularly interested in cooperative games of identical payoffs, i.e., where  $u_i(s) = u(s)$  for all  $i, s$ . Henceforth we will only use this common utility function. We follow the game theoretic convention of representing all players except  $i$  as  $-i$ . Also, a variable will indicate a joint over all agents, unless subscripted by an index  $i$  or  $-i$ .

A *mixed* strategy of player  $i$ ,  $\sigma_i$ , is a probability distribution over  $i$ 's pure strategies, i.e.,  $\sigma_i \in \Delta(S_i)$ . It is well-known that every finite strategic form game has at least one solution in the form of a mixed strategy Nash equilibrium [12], given by  $\sigma_i^* \in \Delta(S_i)$  for all  $i$ , such that

$$u(\sigma_i^*, \sigma_{-i}^*) \geq u(s_i, \sigma_{-i}^*), \forall s_i \in S_i.$$

It is known that in games of identical payoffs, at least one pure strategy Nash equilibrium always exists. However, there may be multiple equilibria, so the goal is generally to find the (Pareto) optimal one.

The notion of *regret* is one of the key concepts to learning in games [19]. A player's regret for not playing a certain strategy in the past is determined by the ex-post amount of utility improvement that would have obtained had the player played that fixed strategy in every past round instead of the actual strategies that he did. Formally, given the sequence of mixed strategy profiles actually played for  $\tau$  rounds,  $\sigma^1, \dots, \sigma^\tau$ , average overall regret of player  $i$  is

$$R_i^\tau = \frac{1}{\tau} \max_{\sigma_i^*} \sum_{t=1}^{\tau} (u(\sigma_i^*, \sigma_{-i}^t) - u(\sigma^t)) \quad (1)$$

Obviously there is no way to know whether the other agents,  $-i$ , would have played the same sequence  $\{\sigma_{-i}^t\}$  had  $i$  played  $\sigma_i^*$  all along, or would have responded differently to the hypothetical change. It turns out, however, that it is possible to drive the average regret  $R_i^\tau$  to zero, *irrespective of*  $\{\sigma_{-i}^t\}$ . A simple yet elegant way to do this is *regret matching* [8] which updates

$$\sigma_i^{\tau+1}(s_i) = \frac{[R_i^\tau(s_i)]_+}{\sum_{s_j \in S_i} [R_i^\tau(s_j)]_+} \quad (2)$$

where

$$R_i^\tau(s_i) = \frac{1}{\tau} \sum_{t=1}^{t=\tau} (u(s_i, \sigma_{-i}^t) - u(\sigma^t))$$

and  $[x]_+ = \max(x, 0)$ . If the denominator of equation 2 is zero, the probabilities are set arbitrarily.

[8] has proven that in a finite general game, regret matching by a given player almost surely yields no regret against every possible sequence  $\{\sigma_{-i}^t\}$ . A direct consequence of this is that if all players use regret matching, then the empirical distribution of joint strategies converges almost surely to the set of *coarse correlated equilibria* (CCE) [8].

This result in general games is rather weak in two respects: first the CCE is a (potentially vast) superset of Nash equilibria, and secondly the almost sure convergence is in terms of empirical play, not the  $\sigma^t$  directly. Recently, [10] has shown that in the special class of *weakly acyclic games*, a regret minimizing sequence  $\{\sigma^t\}$  almost surely converges to a pure strategy Nash equilibrium. The set of weakly acyclic games contains games of identical payoffs as a special case, hence this stronger convergence result carries over to these games.

## STRATEGIC GAME REPRESENTATION OF DEC-POMDPs

When a horizon  $T$  is specified, a Dec-POMDP can be converted to a strategic form game with identical payoffs, where each possible complete conditional  $T$ -step plan of agent  $i$  (i.e., policy tree  $\pi_i$  as shown in Figure 1) in the former is a separate strategy,  $s_i$ , for that agent in the latter [7], and the common values (for all agents) of each joint policy tree gives the identical payoffs of all agents for that joint strategy. Then by [10], regret minimization in such a strategic form game should lead to a pure strategy Nash equilibrium, which corresponds to a set of policy trees – exactly one for each agent – as we would expect a Dec-POMDP solver to yield.

Unfortunately, the strategy space of each agent in such a strategic form game is of size  $O(|A_*|^{|\Omega_*|^T})$ , where  $A_*$ ,  $\Omega_*$  are the largest individual action and observation spaces. Even if dominated strategies are eliminated, there is no known result that ensures significant compaction. Therefore, direct regret minimization in the strategic form is infeasible. The main contribution of this paper is a more tractable alternative approach to regret minimization in a Dec-POMDP, that automatically accomplishes regret minimization in its strategic form.

## REGRET DECOMPOSITION IN A TREE

The main idea is to decompose the overall regret into independently controllable components, such that minimizing the component-regrets independently also minimizes the overall regret.

Recently, regret decomposition in an extensive form game tree in competitive alternate move games with imperfect information – called *counterfactual regret* (CFR) – has found successful application in variants of Poker [20]. In CFR, the overall regret is decomposed along the *information sets* of the game, which roughly translates to individual histories in a Dec-POMDP. [1] also uses history based regrets in their mixed integer linear programming (MILP) formulation of Dec-POMDPs. However, there are  $O(|A_*|^T |\Omega_*|^T)$  histories

in a Dec-POMDP of horizon  $T$ . We propose a more compact decomposition of overall regret in a Dec-POMDP along the *nodes of the policy tree*, of which there are  $O(|\Omega_*|^T)$ , leading to more efficient regret minimization. In order to anchor the notion of regret on the nodes of a tree and in the context of Dec-POMDPs, we will introduce a set of notations below.

We consider a stochastic policy tree  $\pi_i$  (of agent  $i$ ), where each node (contrast with Figure 1 right)  $n_i$  is labeled by a mixed strategy,  $\sigma_i(n_i)$ , instead of the special case of a pure strategy. A history leading to node  $n_i$  in a policy tree  $\pi_i$  is a sequence of pairs of mixed strategies and observations, notated as  $h^{\pi_i}(n_i)$ . Note that a node in  $\pi_i$  can be identified simply by traversing the sequence of observations in  $h^{\pi_i}(n_i)$ .

We define the utility function  $v(\pi, n, s)$  to represent the expected payoff given that the joint policy  $\pi$  is being played, that joint nodes  $n$  in the respective policy trees have been reached, and that the system state is  $s$ . If  $\pi(n)$  gives the (joint) policy subtrees rooted at the joint nodes  $n = (n_1, n_2, \dots, n_k)$ , then  $v$  evaluates the expected payoff of the joint subpolicy  $\pi(n)$ . Note that individual nodes  $n_i$  must be at the same level in all agents' policy trees, and that  $\pi(\text{root})$  is simply notated as  $\pi$ . Specifically,

$$v(\pi, n, s) = \sum_a \sigma(n, a) [R(s, a) + \sum_{s', \omega} P(s', \omega | s, a) v(\pi, n', s')],$$

where  $n'$  is the joint successor node from  $n$  following  $\omega$ , say  $n' = \text{succ}(n, \omega)$ .

Let  $D(n_i)$  be the set of nodes reachable from  $n_i$  in  $\pi_i$ , i.e., the set of nodes in the subtree rooted at (and including)  $n_i$ . Let  $\pi_i|_{D(n_i) \rightarrow \pi'_i}$  represent the policy which is identical to  $\pi_i$  except that the subtree rooted at  $n_i$  is the same as in another policy  $\pi'_i$ . Then, for agent  $i$ , the full regret for playing a sequence of subpolicies rooted at node  $n_i$ ,  $\{\pi_i^t(n_i)\}$ , is

$$R_{i, \text{full}}^\tau(n_i) = \frac{1}{\tau} \max_{\pi'_i} \sum_{t=1}^{t=\tau} \sum_{s, n_{-i}} P(s, h^{\pi_{-i}^t}(n_{-i}) | h^{\pi_i^t}(n_i)) \cdot [v(\pi_i^t|_{D(n_i) \rightarrow \pi'_i}, \pi_{-i}^t, n, s) - v(\pi^t, n, s)] \quad (3)$$

Consistent with the counterfactuality of CFR, equation 3 represents the average regret of  $i$  for not playing the subpolicy of  $\pi'_i$  rooted at  $n_i$  while the others played  $\pi_{-i}^t$ , but given that  $i$  played to reach  $n_i$ . Note that the path of observations leading to  $n_i$ , while determined by nature, is indirectly controlled by all agents via their actions. However, the contribution of agent  $i$  to reaching  $n_i$  is discounted by conditioning on  $h^{\pi_i^t}(n_i)$  to reflect this counterfactuality.

For the sake of notational consistency, verify that

$$\begin{aligned} R_{i, \text{full}}^\tau(\text{root}) &= \frac{1}{\tau} \max_{\pi'_i} \sum_{t, s} P(s, \emptyset | \emptyset) \cdot \\ &\quad [v(\pi_i^t|_{D(\text{root}) \rightarrow \pi'_i}, \pi_{-i}^t, \text{root}, s) - v(\pi^t, \text{root}, s)] \\ &= \frac{1}{\tau} \max_{\pi'_i} \sum_{t, s} b_0(s) \cdot \\ &\quad [v(\pi'_i, \pi_{-i}^t, \text{root}, s) - v(\pi^t, \text{root}, s)] \\ &= \frac{1}{\tau} \max_{\pi'_i} \sum_t (u(\pi'_i, \pi_{-i}^t) - u(\pi^t)) \\ &= R_i^\tau \text{ from equation 1} \end{aligned}$$

Given that a pure policy equilibrium always exists in Dec-POMDPs, i.e., where the node distributions  $\sigma_i$ s are pure, we

**Algorithm 1** REMIT

---

```

1: Initialize policy trees  $\pi_i^0, \forall i$ 
2: Initialize  $R_i^0(n_i, a_i) = 0, \forall i, n_i, a_i$ 
3:  $t \leftarrow 0$ 
4: repeat
5:   for each agent  $i$  do
6:     for each node  $n_i$  in  $\pi_i^t$  (breadth first order) do
7:        $\forall a_i \in A_i, R_i^{t+1}(n_i, a_i) \leftarrow \frac{t}{t+1} R_i^t(n_i, a_i) +$ 
          $\frac{1}{t+1} [\sum_{s, n_{-i}} P(s, h^{\pi_{-i}^t}(n_{-i}) | h^{\pi_i^t}(n_i)) \cdot$ 
          $[v(\pi_i^t |_{\sigma_i(n_i) \rightarrow a_i}, \pi_{-i}^t, n, s) - v(\pi^t, n, s)]]$ 
8:        $\forall a_i \in A_i$ , update  $\pi_i^{t+1}(n_i, a_i)$  to the following:
         
$$\begin{cases} \frac{[R_i^{t+1}(n_i, a_i)]_+}{\sum_{a_i} [R_i^{t+1}(n_i, a_i)]_+} & \text{if denominator} > 0 \\ \pi_i^t(n_i, a_i) & \text{otherwise} \end{cases}$$

9:     end for
10:   end for
11:    $t \leftarrow t + 1$ 
12: until termination_condition

```

---

can define *node regret* as

$$R_i^\tau(n_i) = \frac{1}{\tau} \max_{a_i} \sum_{t=1}^{\tau} \sum_{s, n_{-i}} P(s, h^{\pi_{-i}^t}(n_{-i}) | h^{\pi_i^t}(n_i)) \cdot [v(\pi_i^t |_{\sigma_i(n_i) \rightarrow a_i}, \pi_{-i}^t, n, s) - v(\pi^t, n, s)] \quad (4)$$

This gives the regret for not executing action  $a_i$  at node  $n_i$  while keeping the rest of the (possibly stochastic) subpolicy rooted at  $n_i$  unchanged.

As with CFR, the decomposition of regret along the nodes of a policy tree is also useful according to the following theorem (proof in appendix):

**Theorem 1.** *For each agent, the overall average regret is upper bounded by the sum of the positive node regrets, i.e.,*

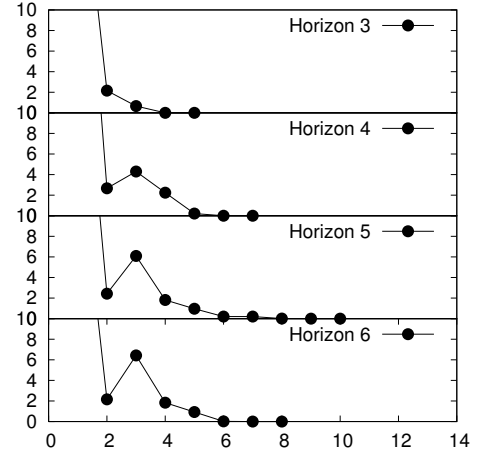
$$R_i^\tau \leq \sum_{n_i} [R_i^\tau(n_i)]_+, \forall i$$

Therefore, as the individual node regrets approach the nonpositive orthant by regret matching (i.e., individually become  $\leq 0$ , by Blackwell's approachability [4]), the overall average regret must also be minimized, i.e., become nonpositive. The key benefit of Theorem 1 is that the individual node regrets can be controlled independently as defined in equation 4. The algorithm, called REMIT (REgret MINimization on Trees), is shown as Algorithm 1. It initializes all agents' stochastic policy trees with the uniform distribution at each node. Then it traverses each agent's tree in a breadth first manner and computes the node regrets and the new node distribution based on regret matching. The complexity of the loop in lines 6–9 is  $O(|A||S|^2|\Omega|^T)$ . This is also effectively the complexity of the **for** loop in lines 5–10 over the set of agents, since this loop can be parallelized because  $i$ 's updates only depend on the previous joint policies. This loop is repeated until a termination condition is met. In this paper we study the following strong convergence criterion:

**Termination Condition.** *Set the termination condition in Algorithm 1 line 12, to*

$$R_i^t(n_i, a_i) = R_i^{t-1}(n_i, a_i) \leq 0, \forall i, n_i, a_i.$$

*This gives a strong convergence criterion that  $\exists t'$  such that  $\forall t \geq t', \pi^t = \pi^{t'}$ .*



**Figure 2:** Relative errors in policy values from known optimals in Dec-Tiger.

This convergence criterion is stronger than the almost sure convergence of [10]. However, neither of these criteria can be theoretically guaranteed for REMIT. This is because Marden et al.'s results apply to the strategic form, and rely on the policy trajectory following a better response path in weakly acyclic games [10]. In contrast, we deliberately avoid working in the strategic form for efficiency, as a result of which there is no guarantee that the sequence of policies produced by tree-regret minimization treads the better response path. In fact, our empirical results show that the policies do not have strictly improving payoffs, i.e., they are indeed not following the better response path. Note however, that even though the means differ, the end is common. That is, by minimizing the component regrets,  $R_i^\tau(n_i)$ , we indirectly minimize the overall regret by Theorem 1. Therefore it is not surprising that we can, in fact, guarantee that the policies  $\pi^t$  approach a Nash equilibrium. We show this with the help of the above termination condition as follows.

While the satisfaction of the above termination condition is not guaranteed, it can be shown that if it is indeed achieved, then the terminal policies do indeed form a Nash equilibrium (proof in appendix).

**Theorem 2.** *If the Termination Condition is satisfied at time  $\tau$ , then  $\pi^\tau$  is a Nash equilibrium.*

Note that REMIT performs regret matching at each node  $n_i$ , therefore by [6] we know that

$$R_i^\tau(n_i) \leq \frac{Z\sqrt{|A_i|}}{\sqrt{\tau}}$$

where  $Z = \max_{\pi, \pi'} (u(\pi) - u(\pi'))$ . In other words, the **Termination Condition** is indeed satisfied in the limit. Hence we have the following theorem:

**Theorem 3.** *Even if we set termination\_condition to 'false', REMIT converges to a Nash equilibrium in the limit.*

Our approach – which is singly exponential in  $T$  instead of doubly exponential – does not contradict the NEXP-hardness of Dec-POMDPs. Regret minimization converges to *some* Nash equilibrium joint policy in the Dec-POMDP, not necessarily the Pareto dominant one. On the other hand

the complexity of Dec-POMDPs reflect the cost of finding the Pareto optimal Nash equilibrium, which is clearly harder than finding just any Nash equilibrium. However, as our experiments show in the next section, REMIT does find (near) optimal equilibrium policies in a range of benchmark Dec-POMDP problems.

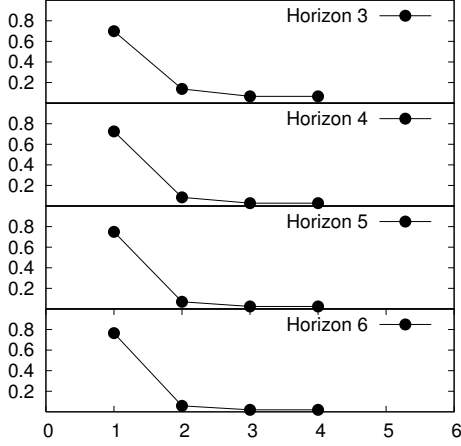


Figure 3: Relative errors in policy values from known optimals in Recycling-Robots.

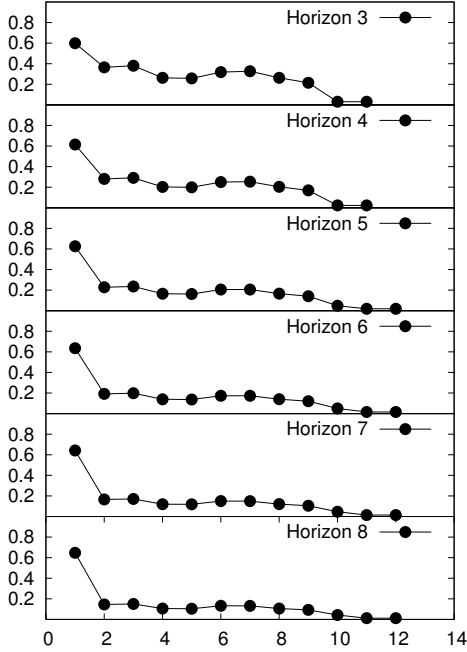


Figure 4: Relative errors in policy values from known optimals in BroadCast-Channel.

## EXPERIMENTAL RESULTS

For experiments, we make an enhancement to line 7 in Algorithm 1, where we replace the weights  $t/t+1$  and  $1/t+1$  by  $(1-\alpha)$  and  $\alpha$  respectively, setting  $\alpha = 0.7$ . This updates the

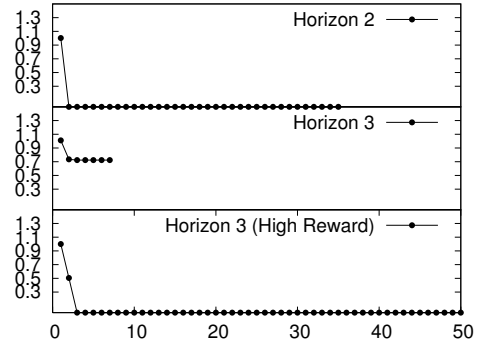


Figure 5: Relative errors in policy values from known optimals in Cooperative-Box-Pushing.

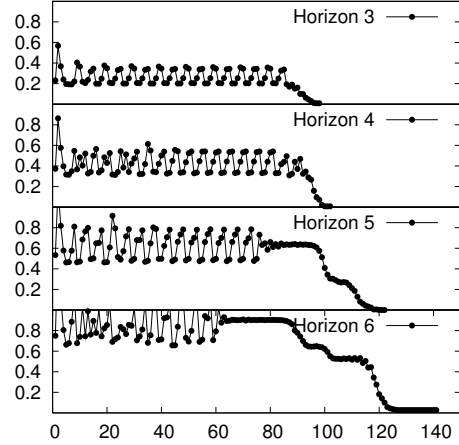


Figure 6: Relative errors in policy values from known optimals in Fire-Fighting.

regrets with *fading memory*. As used in [10]<sup>1</sup>, fading memory is particularly useful for coordination problems when we seek convergence of policies, instead of empirical play. We also use the **Termination Condition** as given before in our experiments. Although there is no guarantee that such a strong convergence condition will be met, we found it to be satisfied in every benchmark problem and horizon that we attempted.

We have experimented in 5 benchmark Dec-POMDP domains – DEC-TIGER, RECYCLING-ROBOTS, BROADCAST-CHANNEL, COOPERATIVE-BOX-PUSHING, and FIRE-FIGHTING (see [16] for details of these problems) – and found REMIT to converge to near-optimal equilibria in all except COOPERATIVE-BOX-PUSHING at  $T = 3$ . Even in this case, it turns out that the reason that REMIT finds a highly suboptimal equilibrium is that the optimal equilibrium is not risk-dominant [5]. As observed in [5], “One might expect that in a learning setting it would be unlikely for play to converge to a very risky equilibrium, even if the equilibrium is Pareto efficient.” To remedy this, we simply raised the payoff of both agents pushing the large box cooperatively from 100 to 10000, which turns out to be sufficient

<sup>1</sup>They also use inertia, which causes slower changes in policy. While this has a higher chance of reaching the optimal equilibrium, it also slows down convergence and has not been tried in the current experiments.



to make the optimal equilibrium risk dominant. REMIT converges to the optimal equilibrium in this case as well.

Figures 2–6 show the results of REMIT on the above 5 domains. The  $x$ -axis in each plot is the number of iterations of the **repeat** loop in Algorithm 1. The  $y$ -axis shows the relative error in policy value at the end of each iteration, compared to the known optimal for each domain and horizon [16]. The right-most plot point in each (sub)figure corresponds to the iteration number when **Termination Condition** became true. We see that in some cases, especially in COOPERATIVE-BOX-PUSHING, the policy values have converged long before this condition is met. If fading memory was not used, then the weight of the older regrets in the accumulated regrets would have been larger, meaning the initial regrets would have left a larger imprint in the accumulated regrets, exacerbating this effect. It is because the older regrets are weighed down sharply that the accumulated regrets can approach nonpositive values (and hence the termination condition) faster. We also see that REMIT attains optimal equilibria in DEC-TIGER (all horizons) and COOPERATIVE-BOX-PUSHING (horizon 3 in the modified version only, as described above). In RECYCLING-ROBOTS, BROADCAST-CHANNEL, and FIRE-FIGHTING, it attains nearly optimal equilibria. COOPERATIVE-BOX-PUSHING (horizon 3 original version) is the only problem (that we tried) where it converges to a highly suboptimal equilibrium. In Figure 6 we see an initial oscillatory pattern, which usually indicates the existence of counteracting attractors in the policy space, from which the REMIT trajectory eventually frees itself to achieve monotonically improving policy values. These attractors appear to be of progressively worse value for increasing horizons.

The main bottleneck for REMIT is the exponentially increasing size of policy trees with  $T$ . Currently, it is unable to exploit ideas such as history clustering [14] to compactly represent policies, and hence cannot solve problems for large  $T$ . However, it does not suffer from the memory and other time bottlenecks that many exact solvers face, and as a result can solve some problems beyond the known maximal horizons quite easily. E.g., in DEC-TIGER, REMIT can solve horizons 7 and 8 in matter of seconds, producing values 9.99357 and 12.2173. Since DEC-TIGER has only been solved up to horizon 6, we do not know if these are optimal, but these are very likely to be so (within rounding errors). This is because the policies produced by REMIT are basically concatenations of horizon 3 policy with horizon 4(5) policy which are very likely to be optimal for  $T = 7(8)$ , given the periodic structure of the problem. For reference, the optimal horizon 6 policy is the concatenation of two horizon 3 policies, leading to the value  $10.3816 = 2 \times 5.1908$ . We do not show plots for  $T$  beyond the known max in any problem, due to uncertainty about the optimal policy values.

The consistent satisfaction of the strong convergence criterion in a range of benchmark problems with different characteristics is surprising. It raises the intriguing possibility that it can be rigorously proven for general regret matching techniques (including REMIT) in identical payoff games. This would be a new result in game theory as well. We leave this for future investigation.

## CONCLUSIONS

We have presented a compact way to apply counterfactual regret minimization to Dec-POMDPs, with a per-node

decomposition of regret as opposed to per-history decomposition. We have proven that our algorithm, REMIT, converges to a Nash equilibrium policy. We have also shown empirically that it actually yields (near) optimal Nash equilibria in a finite number of iterations in a range of benchmark problems.

An immediate future direction is to test a sampling version of REMIT, where the node regrets are *estimated* on the basis of unbiased samples of the subpolicy values. We believe this will yield a more efficient reinforcement learning algorithm than what currently exists.

## ACKNOWLEDGMENT

We thank the reviewers for helpful comments and suggestions. This work was supported in part by the U.S. Army under grant #W911NF-11-1-0124.

## REFERENCES

- [1] R. Aras and A. Dutech. An investigation into mathematical programming for finite horizon decentralized POMDPs. *JAIR*, 37:329–396, 2010.
- [2] B. Banerjee, J. Lyle, L. Kraemer, and R. Yellamraju. Sample bounded distributed reinforcement learning for decentralized POMDPs. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*, pages 1256–1262, Toronto, Canada, July 2012.
- [3] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.
- [4] D. Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6:1–8, 1956.
- [5] D. Fudenberg and K. Levine. *The Theory of Learning in Games*. MIT Press, Cambridge, MA, 1998.
- [6] G. J. Gordon. No-regret algorithms for online convex programs. In *In Neural Information Processing Systems 19*, 2007.
- [7] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 709–715, San Jose, CA, 2004.
- [8] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68:1127–1150, 2000.
- [9] A. Kumar and S. Zilberstein. Dynamic programming approximations for partially observable stochastic games. In *Proceedings of The 22nd International FLAIRS Conference (FLAIRS-09)*, pages 547–552, 2009.
- [10] J. Marden, G. Arslan, and J. Shamma. Regret based dynamics: Convergence in weakly acyclic games. In *Proceedings of 6th Intl. Joint Conf. on Autonomous Agents and Multi-agent Systems*, pages 194–201, 2007.
- [11] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 705–711, Acapulco, Mexico, 2003.

- [12] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286 – 295, 1951.
- [13] F. A. Oliehoek, J. F. Kooij, and N. Vlassis. The cross-entropy method for policy search in decentralized POMDPs. *Informatica*, 32:341–357, 2008.
- [14] F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, pages 577–584, Budapest, Hungary, 2009.
- [15] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2009–2015, Hyderabad, India, 2007.
- [16] M. Spaan. Dec-POMDP problem domains and format. <http://users.isr.ist.utl.pt/~mtjspaan/decpomdp/>.
- [17] M. T. J. Spaan, F. A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 2027–2032, Barcelona, Spain, 2011.
- [18] D. Szer, F. Charpillet, and S. Zilberstein. MAA\*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 576–590, 2005.
- [19] H. P. Young. *Strategic Learning and its Limits*. Oxford University Press, 2004.
- [20] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *NIPS*, 2007.

## APPENDIX

### PROOF OF THEOREM 1

Suppose the state transitions from  $s$  to  $s'$  when the agents execute joint action  $a$  at joint history  $h$ , producing joint observation  $\omega$ . Let  $n_{h_i}$  represent the node at the end of history  $h_i$ , and  $h' = (h.a.\omega)$ , i.e., the next joint history. For brevity, we write the transition and observation probabilities jointly as  $P(s', \omega | s, a)$ . Although our proof of Theorem 1 is somewhat similar to that of Theorem 3 in [20], a key lemma that distinguishes our proof concerns the propagation of  $i$ 's conditional belief  $P(s, h_{-i} | h_i)$ , given next. There are also some original steps embedded in the proof of Lemma 5 which is the counterpart of Lemma 5 in [20].

**Lemma 4.** *The conditional belief propagation is given as*

$$P(s', h'_{-i} | h'_i) = \sum_s P(s', \omega | s, a) \pi_{-i}(n_{h_{-i}}, a_{-i}) \cdot P(s, h_{-i} | h_i) / P(\omega_i | a_i, h_i)$$

**Proof:** We observe

$$\begin{aligned} P(s', h'_{-i} | h'_i) &= \sum_s P(s, s', h, a, \omega) / P(h_i, a_i, \omega_i) \\ &= \sum_s P(s', \omega | s, a) P(s, h, a) / P(h_i, a_i, \omega_i) \\ &= \sum_s P(s', \omega | s, a) P(a_{-i} | h_{-i}) \cdot P(s, h, a_i) / P(h_i, a_i, \omega_i) \\ &= \sum_s P(s', \omega | s, a) \pi_{-i}(n_{h_{-i}}, a_{-i}) \cdot P(s, h_{-i} | h_i) P(h_i, a_i) / P(h_i, a_i, \omega_i) \\ &= \sum_s P(s', \omega | s, a) \pi_{-i}(n_{h_{-i}}, a_{-i}) \cdot P(s, h_{-i} | h_i) / P(\omega_i | a_i, h_i) \end{aligned}$$

**Lemma 5.** *Suppose the successor joint node after making the joint observation  $\omega$  at joint node  $n$  be  $n'$ . Then,*

$$R_{i,\text{full}}^\tau(n_i) \leq R_i^\tau(n_i) + \sum_{n'_i} [R_{i,\text{full}}^\tau(n'_i)]_+, \quad \forall i$$

**Proof:** We break  $R_{i,\text{full}}^\tau(n_i)$  into two components: the node regret at  $n_i$  and the full regret of the subtrees under  $n_i$ , starting at its definition in equation 3.

$$\begin{aligned} R_{i,\text{full}}^\tau(n_i) &= \frac{1}{\tau} \max_{\pi'_i} \sum_{t=1}^{\tau} \sum_{s, n_{-i}} P(s, h^{\pi^t_{-i}}(n_{-i}) | h^{\pi^t_i}(n_i)) \cdot [v(\pi^t_i |_{D(n_i) \rightarrow \pi'_i}, \pi^t_{-i}, n, s) - v(\pi^t, n, s)] \\ &= \frac{1}{\tau} \max_{a_i, \pi'_i} \sum_{t=1}^{\tau} \sum_{s, n_{-i}} P(s, h^{\pi^t_{-i}}(n_{-i}) | h^{\pi^t_i}(n_i)) \cdot [v(\pi^t_i |_{\sigma_i(n_i) \rightarrow a_i}, \pi^t_{-i}, n, s) - v(\pi^t, n, s) + \sum_{n', s', a_{-i}} P(n', s', a_{-i} | n, s, a_i) \cdot \{v(\pi^t_i |_{D(n'_i) \rightarrow \pi'_i}, \pi^t_{-i}, n', s') - v(\pi^t, n', s')\}] \\ &\leq R_i^\tau(n_i) + \max_{a_i, \pi'_i} \frac{1}{\tau} \sum_{t, s, n_{-i}} P(s, h^{\pi^t_{-i}}(n_{-i}) | h^{\pi^t_i}(n_i)) \cdot \sum_{n', s', a_{-i}} P(n', s', a_{-i} | n, s, a_i) \cdot [v(\pi^t_i |_{D(n'_i) \rightarrow \pi'_i}, \pi^t_{-i}, n', s') - v(\pi^t, n', s')] \\ &= R_i^\tau(n_i) + \max_{a_i, \pi'_i} \frac{1}{\tau} \sum_{t, s, n_{-i}} P(s, h^{\pi^t_{-i}}(n_{-i}) | h^{\pi^t_i}(n_i)) \cdot \sum_{\omega, s', a_{-i}} P(s', a_{-i}, \omega | s, h^{\pi^t}(n), a_i) \cdot [v(\pi^t_i |_{D(n'_i) \rightarrow \pi'_i}, \pi^t_{-i}, n', s') - v(\pi^t, n', s')] \\ &= R_i^\tau(n_i) + \max_{a_i, \pi'_i} \frac{1}{\tau} \sum_{t, \omega_i} \left\{ \sum_{s', n_{-i}, a_{-i}, \omega_{-i}} \sum_s P(s, h^{\pi^t_{-i}}(n_{-i}) | h^{\pi^t_i}(n_i)) P(s', a_{-i}, \omega | s, h^{\pi^t}(n), a_i) \cdot [v(\pi^t_i |_{D(n'_i) \rightarrow \pi'_i}, \pi^t_{-i}, n', s') - v(\pi^t, n', s')] \right\} \end{aligned}$$

Now,

$$\begin{aligned}
P(s', a_{-i}, \omega | s, h^{\pi^t}(n), a_i) &= P(s', \omega | s, a) P(a_{-i} | s, h^{\pi^t}(n), a_i) \\
&= P(s', \omega | s, a) P(a_{-i} | h^{\pi^t}(n_{-i})) \\
&= P(s', \omega | s, a) \pi_{-i}^t(n_{h_{-i}}, a_{-i})
\end{aligned}$$

Therefore, we can replace

$$\sum_s P(s, h^{\pi^t}(n_{-i}) | h^{\pi^t}(n_i)) P(s', a_{-i}, \omega | s, h^{\pi^t}(n), a_i)$$

in the last step of  $R_{i,\text{full}}^\tau(n_i)$  by

$$\sum_s P(s, h^{\pi^t}(n_{-i}) | h^{\pi^t}(n_i)) P(s', \omega | s, a) \pi_{-i}^t(n_{h_{-i}}, a_{-i}).$$

Then using Lemma 4, we can replace this by

$$P(s', h^{\pi^t}(n'_{-i}) | h^{\pi^t}(n'_i)) P(\omega_i | h^{\pi^t}(n_i), a_i)$$

Then continuing from the last step of  $R_{i,\text{full}}^\tau(n_i)$ ,

$$\begin{aligned}
R_{i,\text{full}}^\tau(n_i) &\leq R_i^\tau(n_i) + \max_{a_i, \pi_i'} \frac{1}{\tau} \sum_{t, \omega_i} \left\{ \sum_{s', n'_{-i}, a_{-i}, \omega_{-i}} \right. \\
&\quad P(s', h^{\pi^t}(n'_{-i}) | h^{\pi^t}(n'_i)) P(\omega_i | h^{\pi^t}(n_i), a_i) \cdot \\
&\quad \left. [v(\pi_i^t |_{D(n'_i) \rightarrow \pi'_i}, \pi_{-i}^t, n', s') - v(\pi^t, n', s')] \right\} \\
&= R_i^\tau(n_i) + \max_{a_i, \pi_i'} \frac{1}{\tau} \sum_{t, \omega_i} \left\{ P(\omega_i | h^{\pi^t}(n_i), a_i) \cdot \right. \\
&\quad \sum_{s', n'_{-i}} P(s', h^{\pi^t}(n'_{-i}) | h^{\pi^t}(n'_i)) \cdot \\
&\quad \left. [v(\pi_i^t |_{D(n'_i) \rightarrow \pi'_i}, \pi_{-i}^t, n', s') - v(\pi^t, n', s')] \right\} \\
&= R_i^\tau(n_i) + \max_{a_i} \sum_{\omega_i} \left\{ P(\omega_i | h^{\pi^t}(n_i), a_i) R_{i,\text{full}}^\tau(n'_i) \right\} \\
&\leq R_i^\tau(n_i) + \sum_{\omega_i} R_{i,\text{full}}^\tau(n'_i) \\
&= R_i^\tau(n_i) + \sum_{n'_i} R_{i,\text{full}}^\tau(n'_i) \\
&\leq R_i^\tau(n_i) + \sum_{n'_i} [R_{i,\text{full}}^\tau(n'_i)]_+
\end{aligned}$$

This concludes the proof of lemma 5.  $\square$

The rest of the proof of Theorem 1 is by inductive application of Lemma 5 from the root to the leaves of  $i$ 's policy, as in [20].

## PROOF OF THEOREM 2

We make the standard assumption that all payoffs are bounded, so that there exists a well defined minimum value of joint policies:

$$m \triangleq \min_{\pi} u(\pi)$$

Now if the convergence criterion is satisfied,  $\pi^\tau$  remains unchanged for all  $t > \tau$  if REMIT keeps running beyond  $\tau$ . Suppose,  $\pi^\tau$  is not a Nash equilibrium; then there exists a policy  $\pi_i^*$  for some agent  $i$  such that the joint payoff strictly improves, i.e., improves by at least  $\epsilon > 0$ . That is

$$u(\pi_i^*, \pi_{-i}^\tau) \geq u(\pi^\tau) + \epsilon.$$

Let us fix some  $\tau'$  beyond  $\tau$  such that

$$\tau' > \tau(1 + |m|/\epsilon)$$

Now the overall regret at the policy root for agent  $i$  over  $\tau'$  steps is

$$\begin{aligned}
R_i^{\tau'} &= \frac{1}{\tau'} \max_{\pi_i'} \sum_{t=1}^{\tau'} (u(\pi_i', \pi_{-i}^t) - u(\pi^t)) \\
&\geq \frac{1}{\tau'} \sum_{t=1}^{\tau'} (u(\pi_i^*, \pi_{-i}^t) - u(\pi^t)) \\
&= \frac{1}{\tau'} \left[ \sum_{t=1}^{\tau} (u(\pi_i^*, \pi_{-i}^t) - u(\pi^t)) + \right. \\
&\quad \left. \sum_{t=\tau+1}^{\tau'} (u(\pi_i^*, \pi_{-i}^t) - u(\pi^t)) \right] \\
&= \frac{1}{\tau'} \left[ \sum_{t=1}^{\tau} (u(\pi_i^*, \pi_{-i}^t) - u(\pi^t)) + \right. \\
&\quad \left. (\tau' - \tau)(u(\pi_i^*, \pi_{-i}^\tau) - u(\pi^\tau)) \right] \\
&\geq \frac{1}{\tau'} \left[ \sum_{t=1}^{\tau} (u(\pi_i^*, \pi_{-i}^t) - u(\pi^t)) + (\tau' - \tau)\epsilon \right] \\
&\geq \frac{1}{\tau'} [-|m|\tau + (\tau' - \tau)\epsilon] \\
&> 0, \text{ by the choice of } \tau'
\end{aligned}$$

However, since all node regrets of all agents are  $\leq 0$  at  $\tau'$ , in particular for agent  $i$ , by Theorem 1 we know that  $R_i^{\tau'} \leq 0$  – a contradiction. Therefore  $\pi^\tau$  must be a Nash equilibrium.

# Automated Generation of Interaction Graphs for Value-Factored Decentralized POMDPs

William Yeoh  
Department of Computer Science  
New Mexico State University  
Las Cruces, NM 88011, USA  
wyeoh@cs.nmsu.edu

Akshat Kumar  
Analytics and Optimization Group  
IBM Research  
New Delhi, India 110 070  
akshat.kumar@gmail.com

Shlomo Zilberstein  
School of Computer Science  
University of Massachusetts  
Amherst, MA 01003, USA  
shlomo@cs.umass.edu

## ABSTRACT

The Decentralized Partially Observable Markov Decision Process (Dec-POMDP) is a powerful model for multi-agent planning under uncertainty, but its applicability is hindered by its high complexity – solving Dec-POMDPs optimally is NEXP-hard. Recently, Kumar *et al.* introduced the Value Factorization (VF) framework, which exploits decomposable value functions that can be factored into subfunctions. This framework has been shown to be a generalization of several specialized models such as TI-Dec-MDPs, ND-POMDPs and TD-POMDPs, which leverage different forms of sparse agent interactions to improve the scalability of planning. Existing algorithms for these models assume that the interaction graph of the problem is given. So far, no studies have addressed the generation of interaction graphs. In this paper, we address this gap by introducing three algorithms to automatically generate interaction graphs for models within the VF framework and establish lower and upper bounds on the expected reward of an optimal joint policy. We illustrate experimentally the benefits of these techniques for sensor placement in a decentralized tracking application.

## 1. INTRODUCTION

Markov Decision Processes (MDPs) and Partially Observable MDPs (POMDPs) have been shown to be popular models for modeling problems where a single agent needs to plan under uncertainty. As a result, Decentralized POMDPs (Dec-POMDPs) [2] have emerged as a natural extension for modeling problems where a team of agents need to plan under uncertainty. Unfortunately, scalability is an issue for Dec-POMDP algorithms because not only is finding optimal solutions NEXP-hard [2], but finding constant factor approximations is also NEXP-hard [13].

In general, researchers have taken two approaches to address this issue. The first approach is motivated by the observation that many multi-agent planning problems, like our sensor network problem, have *sparse agent interactions*, that is, each agent only interacts with a *limited* number of other agents. Thus, researchers have introduced specialized models such as ND-POMDPs [10], TD-POMDPs [17] and DPCL [16], which exploit the sparsity in these interactions to increase scalability.

In the second approach, researchers assume that the problem can be factored into smaller factors. For example, Oliehoek *et al.* introduced models where the state space and reward function are factored into subspaces and subfunctions [12, 11]. More recently, Kumar *et al.* introduced

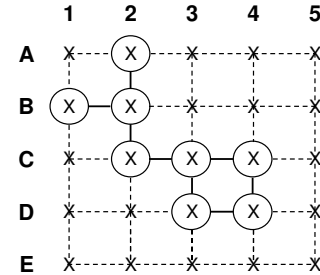


Figure 1: Example Sensor Network

the *Value Factorization* (VF) framework, where the value functions are factored into subfunctions [8]. This framework is appealing as it has been shown to represent several sparse-interaction models like TI-Dec-MDPs [1], ND-POMDPs [10], TD-POMDPs [17]. We thus describe our work in the context of the VF framework and use ND-POMDPs as one example model within this framework.

All the value-factored Dec-POMDP algorithms developed thus far assume that the interaction graph of the problem, that is, the number of agents and their possible interactions, is given. However, there have been no studies on the automated generation of interaction graphs, which is unfortunate because not only is the interaction graph often unspecified in many multi-agent applications, but the choice of interaction graph is often a design decision that needs to be optimized. For example, the placement of sensors (agents) to form a sensor network (interaction graph) is often unspecified in a decentralized tracking application, and the choice of interaction graph can directly affect the expected rewards of joint policies computed for that problem. Therefore, in this paper, we introduce three algorithms to automatically generate interaction graphs, which increase the size and density of the interaction graph only when the increase is believed to be beneficial. We also show how one can calculate lower and upper bounds on the expected reward of an optimal joint policy across all possible interaction graphs.

## 2. MOTIVATING EXAMPLE: SENSOR PLACEMENT

We motivate the work in this paper with sensor network problems, where multiple sensors need to coordinate with each other to track non-adversarial targets that are moving in an area. Examples of such problems include the tracking of vehicle movements [9] and the tracking of weather phe-

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with *AAMAS*, May 2013, St. Paul, Minnesota, USA.

nomena such as tornadoes [5]. We assume that the sensors are immobile and at least two sensors are required to scan a potential target location. We also assume that the interaction graph, that is, the number of sensors, their locations and their possible interactions, is not given and, thus, the generation of the interaction graph is part of the problem. However, the possible sensor placement locations are given and each sensor can only interact with its neighboring sensor to scan the location between them. Lastly, we assume that we have an insufficient number of sensors to place at every location. (The interaction graph is otherwise trivial.) The objective is to place the sensors and find their joint policy such that the expected reward of that joint policy is maximized. For example, Figure 1 illustrates a sensor network, where the crosses denote the 25 possible sensor placement locations, the circles denote the 8 placed sensor locations and the lines denote the possible target locations. The circles and solid lines form nodes and edges of the interaction graph, respectively.

### 3. SPARSE-INTERACTION MODELS

We now describe the VF framework and ND-POMDP model, which we use to model the problems in this paper.

#### 3.1 Value Factorization Framework

The *Value Factorization* (VF) framework assumes that each joint state  $s$  can be factored such that  $s = (s^1, \dots, s^m)$ , which is true in several multi-agent planning models such as TI-Dec-MDPs [1], ND-POMDPs [10] and TD-POMDPs [17]. Without making further (conditional independence) assumptions on the problem structure, a general Dec-POMDP requires exact inference in the full corresponding (finite-time) DBNs, which would be exponential in the number of state variables and agents. The value factorization approach relies on a general, simplifying property of agent interaction, which can be shown to be consistent with many of the existing multi-agent planning models [8, 18]. We next summarize key ideas behind this framework.

Given a Dec-(PO)MDP defined by the set of agents  $N$ , joint states  $S$  and joint actions  $A$ , a *value factor*  $f$  defines a subset of agents  $N_f \subseteq N$ , joint states  $S_f \subseteq S$ , and joint actions  $A_f \subseteq A$ . A multi-agent planning problem satisfies *value factorization* if the joint-policy value function can be decomposed into a sum over value factors:

$$V(s, \theta) = \sum_{f \in F} V_f(s^f, \theta^f), \quad (1)$$

where  $F$  is a set of value factors,  $\theta^f \equiv \theta^{N_f}$  is the collection of parameters of the agents of factor  $f$ , and  $s^f \equiv s^{S_f}$  is the collection of state variables of this factor.

Even when the value factorization property holds, planning in such models is still highly coupled because factors may overlap. That is, an agent can appear in multiple factors as can state variables. Therefore, a value factor cannot be optimized *independently*. But, it has been shown that such structured agent interactions lead to tractable planning algorithms [8]. Such additive value functions have also been used to solve large factored MDPs [4].

#### 3.2 Networked Distributed POMDPs

For concrete illustrations and evaluation of the results in this paper, we use *Network Distributed POMDPs* (ND-POMDPs) – one of the most commonly used model that

satisfies the value factorization property. Formally, an ND-POMDP is defined as a tuple  $\langle S, A, \Omega, P, O, R, b, H \rangle$ , where

$S$  is the set of joint states.  $S = \times_{1 \leq i \leq n} S_i \times S_u$ , where  $S_i$  is the set of local states of agent  $i$  and  $S_u$  is the set of uncontrollable states that are independent of the actions of the agents. Each joint state  $s \in S$  is defined by  $\langle s_u, s_1, \dots, s_n \rangle$ , where  $s_i \in S_i$  and  $s_u \in S_u$ .

$A$  is the set of joint actions.  $A = \times_{1 \leq i \leq n} A_i$ , where  $A_i$  is the set of actions of agent  $i$ . Each joint action  $a \in A$  is defined by  $\langle a_1, \dots, a_n \rangle$ , where  $a_i \in A_i$ .

$\Omega$  is the set of joint observations.  $\Omega = \times_{1 \leq i \leq n} \Omega_i$ , where  $\Omega_i$  is the set of observations of agent  $i$ . Each joint observation  $\omega \in \Omega$  is defined by  $\langle \omega_1, \dots, \omega_n \rangle$ , where  $\omega_i \in \Omega_i$ .

$P$  is the set of joint transition probabilities that assume conditional transition independence.  $P = \times_{s' \in S, s \in S, a \in A} P(s'|s, a)$ , where  $P(s'|s, a) = P_u(s'_u|s_u) \cdot \prod_{1 \leq i \leq n} P_i(s'_i|s_i, s_u, a_i)$  is the probability of transitioning to joint state  $s'$  after taking joint action  $a$  in joint state  $s$ .

$O$  is the set of joint observation probabilities that assume conditional observation independence.  $O = \times_{\omega \in \Omega, s \in S, a \in A} O(\omega|s, a)$ , where  $O(\omega|s, a) = \prod_{1 \leq i \leq n} O_i(\omega_i|s_i, s_u, a_i)$  is the probability of jointly observing  $\omega$  after taking joint action  $a$  in joint state  $s$ .

$R$  is the set of joint reward functions that are decomposable among the agent subgroups  $e = \{e_1, \dots, e_k\}$ .  $R = \times_{s \in S, a \in A} R(s, a)$ , where  $R(s, a) = \sum_e R_e(s_e, s_u, a_e)$  is the reward of taking joint action  $a$  in joint state  $s$ .  $R_e(s_e, s_u, a_e)$  is the reward of taking joint action  $a_e$ , which is defined by  $\langle a_{e_1}, \dots, a_{e_k} \rangle$ , in joint states  $s_e$ , which is defined by  $\langle s_{e_1}, \dots, s_{e_k} \rangle$ , and  $s_u$ .  $a_{e_i}$  and  $s_{e_i}$  is the action and state of agent  $e_i \in e$ , respectively.

$b$  is the belief over the initial joint state.  $b = \times_{s \in S} b(s)$ , where  $b(s) = b(s_u) \cdot \prod_{1 \leq i \leq n} b(s_i)$  is the belief for joint state  $s$ .

$H$  is the horizon of the problem. In this paper, we address finite-horizon problems.

ND-POMDPs can be represented by the VF framework by associating each agent subgroup  $e$  as a value factor. For our sensor network problem, each agent subgroup corresponds to an edge in the graph.

### 4. AUTOMATED INTERACTION GRAPH GENERATION

All the value-factored Dec-POMDP algorithms developed thus far assume that the interaction graph of the problem is given. However, there have been no studies on the automated generation of interaction graphs, which is unfortunate because not only is the interaction graph often unspecified in many multi-agent applications, but the choice of interaction graph is often a design decision that needs to be optimized. For example, when there is an insufficient number of sensors to place in every possible location, the placement of sensors to form a sensor network can directly affect the expected rewards of joint policies computed for that problem. Additionally, the choice of interaction graph can also affect the time and space complexities of ND-POMDP algorithms. For example, the time and space complexities of CBDP, a current state-of-the-art ND-POMDP algorithm, are expo-

nential in the induced width of the interaction graph [7]. Therefore, in this paper, we formalize the problem of finding an optimal interaction graph and introduce three algorithms to automatically generate interaction graphs based on contribution estimates of edges to the expected reward: two greedy algorithms and a mixed integer linear programming-based algorithm.

## 4.1 Problem Statement

Given a fixed number of available homogeneous agents and a set of feasible value factors (which is the set of all feasible edges in our sensor network) and their transition, observation and reward functions, a solution is a subset of value factors (which together define an interaction graph and accompanying Dec-POMDP) that satisfy some feasibility constraints (e.g., the number of agents involved in the value factors is no more than the number of available agents). The quality of a solution is the expected reward of an optimal Dec-POMDP policy for that solution. An optimal solution is a solution with the best quality. While our approaches only apply to problems with homogeneous agents, they can be extended to work with heterogeneous agents with additional constraints in the MILPs.

## 4.2 Greedy Algorithm

Since not all the candidate edges that can be added to the interaction graph are equally important, a natural starting point would be to consider a greedy algorithm that generates an interaction graph by incrementally adding edges based on their contribution, or estimates of their contributions, to the expected reward. This idea is similar to how the algorithm presented in [6] incrementally places sensors based on the additional information that they provide about the unsensed locations. We now introduce two variants of this greedy algorithm.

### 4.2.1 Naive Greedy Algorithm

This algorithm greedily adds value factors based on their actual contribution to the expected reward. In each iteration, it repeatedly loops over all candidate value factors, that is, unchosen value factors that does not require more agents than available; computes a joint policy with each candidate value factor when it is added to the interaction graph; and chooses the value factor with the largest positive gain in expected reward to be added to the interaction graph. This process continues until no candidate value factor results in a positive gain or there are no remaining candidate value factors to consider.

### 4.2.2 Heuristic Greedy Algorithm

The computation of the joint policy for each candidate value factor can be inefficient. For example, the time and space complexities of a current state-of-the-art ND-POMDP algorithm is exponential in the induced width of the interaction graph [7]. Thus, we also introduce a variant of the greedy algorithm that uses contribution estimates of each value factor to the expected reward to greedily select value factors.

We estimate the contribution of each value factor as the sum of expected joint rewards gained by agents defined for that value factor from coordinating with each other across all time steps. More precisely, we calculate the estimated contribution  $w_f$  of each value factor  $f$ :

$$w_f = \sum_{t=1}^H \max_{\vec{a} \in A_f} w_f^{\vec{a},t} \quad (2)$$

$$w_f^{\vec{a},t} = \sum_{s \in S} b'(s,t) \cdot R_f(s, \vec{a}) \quad (3)$$

where  $A_f$  is the set of joint actions for value factor  $f$ ;  $b'(s,t)$  is the belief at state  $s$  and time step  $t$  calculated using MDP-based sampling starting from the initial belief  $b$  [15, 14]; and  $R_f(s, \vec{a})$  is the reward of the value factor  $f$  given state  $s$  and joint action  $\vec{a}$ . For ND-POMDPs, value factors correspond to edges. Thus, the equations are:

$$w_e = \sum_{t=1}^H \max_{\vec{a} \in A_e} w_e^{\vec{a},t} \quad (4)$$

$$w_e^{\vec{a},t} = \sum_{s \in S} b'(s,t) \cdot R_e(s, \vec{a}) \quad (5)$$

We expect this version of the greedy algorithm to run significantly faster since it only needs to compute the joint policy *once* at the end of the algorithm instead of *each time* it evaluates a candidate value factor.

## 4.3 MILP-based Algorithm

While the Heuristic Greedy algorithm can efficiently generate interaction graphs, it uses heuristic values that assume that an agent involved in multiple VFs can take different actions for each VF, which is an incorrect assumption. Thus, we introduce a mixed integer linear program (MILP) that assumes that each agent must choose the same action for all VFs that it is involved in. This MILP finds an *open loop* joint policy<sup>1</sup> that is optimal across all possible interaction graphs and returns the interaction graph that joint policy is operating on. Figure 2 shows the MILP, where

$B$  is the maximum number of available agents. It is an input parameter.

$w_f^{\vec{a},t}$  is the normalized estimate of the expected joint rewards of agents in value factor  $f$  taking joint action  $\vec{a}$  at time step  $t$ . It is the same input parameter described earlier in Eq. 3, except that it is now normalized.

$n_i$  is a boolean variable indicating if agent  $i$  is chosen for the interaction graph (Line 9). The constraint on Line 2 ensures that the number of agents chosen does not exceed the maximum number of available agents.

$fac_f$  is a boolean variable indicating if value factor  $f$  is chosen for the interaction graph (Line 10). The constraint on Line 3 ensures a value factor is chosen only if all agents involved in that factor are chosen.

$act_i^{a,t}$  is a boolean variable indicating if agent  $i$  is taking action  $a$  at time step  $t$  (Line 9). The constraint on Line 4 ensures that an agent can take at most one action in each time step.

$act_f^{\vec{a},t}$  is a boolean variable indicating if all the agents involved in value factor  $f$  are taking joint action  $\vec{a}$  at time step  $t$  (Line 10). Note that we use the vector notation to represent joint actions and regular notations to represent individual actions. The constraint on Line 5 ensures that the joint action  $\vec{a} \in A_f$  is taken

<sup>1</sup>An open loop joint policy is a policy that is independent of agent observations. In other words, the joint policy is a sequence of  $H$  actions for each agent, where  $H$  is the horizon of the problem.

<b>Maximize</b>	$\sum_{t,f,\vec{a} \in A_f} obj_f^{\vec{a},t}$	(Line 1)
<b>Subject to</b>		
	$\sum_i n_i \leq B$	(Line 2)
	$fac_f \leq n_i \quad \forall f, i \in f$	(Line 3)
	$\sum_{a \in A_i} act_i^{a,t} \leq 1 \quad \forall t, i$	(Line 4)
	$act_f^{\vec{a},t} \leq fac_f \quad \forall t, f, \vec{a} \in A_f$	(Line 5)
	$act_f^{\vec{a},t} \leq act_i^{a,t} \quad \forall t, f, i \in f, \vec{a} \in A_f, a \in A_i \cap \vec{a}$	(Line 6)
	$obj_f^{\vec{a},t} \leq act_f^{\vec{a},t} \quad \forall t, f, \vec{a} \in A_f$	(Line 7)
	$obj_f^{\vec{a},t} \leq w_f^{\vec{a},t} \quad \forall t, f, \vec{a} \in A_f$	(Line 8)
	$n_i, act_i^{a,t} \in \{0, 1\} \quad \forall t, i, a \in A_i$	(Line 9)
	$fac_f, act_f^{\vec{a},t} \in \{0, 1\} \quad \forall t, f, \vec{a} \in A_f$	(Line 10)
	$obj_f^{\vec{a},t} \in [0, 1] \quad \forall t, f, \vec{a} \in A_f$	(Line 11)

Figure 2: MILP for the VF Framework

only if value factor  $f$  is chosen and the constraint on Line 6 ensures that the joint action  $\vec{a}$  is taken only if all the individual actions  $a \in \vec{a}$  are taken.

$obj_f^{\vec{a},t}$  is the objective variable whose sum is maximized by the MILP (Lines 1 and 11). The constraints on Lines 7 and 8 ensure that  $obj_f^{\vec{a},t}$  equals  $w_f^{\vec{a},t}$  if  $act_f^{\vec{a},t}$  is 1 and equals 0 otherwise. Thus, maximizing the objective variables over all time steps, value factors and joint actions maximizes the expected reward of the open loop joint policy.

Once we solve the MILP, the interaction graph is formed by exactly those agents  $i$  and value factors  $f$  whose boolean variables  $n_i$  and  $fac_f$ , respectively, equals 1. One can then use it to compute a *closed loop* joint policy.<sup>2</sup>

#### 4.3.1 Optimizations for ND-POMDPs

For ND-POMDPs, a positive reward is typically obtained for only *one* joint action in each edge (value factor). For example, in our sensor network problem, a reward is obtained only if two agents sharing an edge coordinates with each other to scan the area between them (denoted by that edge). If either agent does not scan that area, then neither agent gets any reward. As such, one can optimize the general MILP for the VF framework to a specialized MILP for ND-POMDPs by removing the superscripts  $\vec{a}$  (since each edge maps to exactly one useful joint action) resulting in a significant reduction in the number of variables and constraints.

Figure 3 shows this MILP, where subscripts  $i$  and  $j$  denote agent IDs and  $ij$  denotes the edge between agents  $i$  and  $j$ . The constraints on Lines 13 and 14 correspond to those on Lines 2 and 3, respectively. The constraints on Line 15 are to ensure symmetry. The constraints on Lines 16 and 17 correspond to those on Lines 4 and 5, and the constraints on Line 18 correspond to those on Lines 7 and 8.

#### 4.3.2 Complexity Discussions

<sup>2</sup>A closed loop joint policy is a regular Dec-POMDP joint policy that is dependent on agent observations.

<b>Maximize</b>	$\sum_{t,i,j} obj_{ij}^t$	(Line 12)
<b>Subject to</b>		
	$\sum_i n_i \leq B$	(Line 13)
	$fac_{ij} \leq n_i \quad fac_{ij} \leq n_j$	(Line 14)
	$fac_{ij} = fac_{ji} \quad act_{ij}^t = act_{ji}^t$	(Line 15)
	$\sum_i act_{ij}^t \leq 1 \quad act_{ij}^t \leq fac_{ji}$	(Line 16)
	$act_{ij}^t = 0$ for all $i$ and $j$ that are not neighbors	(Line 17)
	$obj_{ij}^t \leq act_{ij}^t \quad obj_{ij}^t \leq w_{ij}^t$	(Line 18)
	$n_i \in \{0, 1\} \quad fac_{ij} \in \{0, 1\}$	(Line 19)
	$act_{ij}^t \in \{0, 1\} \quad obj_{ij}^t \in [0, 1]$	(Line 20)

Figure 3: MILP for ND-POMDPs

The MILP requires  $O(H \cdot n \cdot |\hat{A}_i|)$  variables and  $O(H \cdot |F| \cdot |\hat{f}| \cdot |\hat{A}_f| \cdot |\hat{A}_i|)$  constraints, where  $H$  is the horizon,  $n$  is the number of agents,  $|\hat{A}_i|$  is the maximum number of actions per agent,  $|F|$  is the number of value factors,  $|\hat{f}|$  is the maximum number of agents in a value factor and  $|\hat{A}_f|$  is the maximum number of joint actions in a value factor. The dominating terms for the number of variables and constraints are the number of  $act_f^{a,t}$  variables and the constraints on Line 6, respectively.

While the number of constraints might appear intractably large, bear in mind that one of the main motivations of sparse-interaction models is that the number of agents and joint actions in a value factor is *small* such that they can be exploited for scalability.

## 4.4 Lower Bounds

The open loop joint policy found by the MILP can be extracted from its result by taking the union of all actions  $a$  of agent  $i$  at time step  $t$  whose boolean variable  $act_i^{a,t}$  equals 1. One can then evaluate that policy to get an expected reward, which will form a lower bound on the optimal expected reward. For ND-POMDPs, evaluating an open loop joint policy  $\pi$  to get the expected reward  $V^\pi(b)$  for the initial belief  $b$  can be done:

$$V^\pi(b) = \sum_{s \in S} b(s) \cdot \sum_{e \in E} V_e^\pi(s, 0) \quad (6)$$

$$V_e^\pi(s, t) = R_e(s, \pi(t)) + \sum_{s' \in S} P(s'|s, \pi(t)) \cdot \sum_{\omega \in \Omega} O(\omega|s', \pi(t)) \cdot V_e^\pi(s', t+1) \quad (7)$$

$$= R_e(s, \pi(t)) + \sum_{s' \in S} P(s'|s, \pi(t)) \cdot V_e^\pi(s', t+1) \quad (8)$$

where  $E$  is the set of edges in the interaction graph and  $\pi(t)$  is the joint action of the agents at time step  $t$  according to joint policy  $\pi$ . Eq. 7 simplifies to Eq. 8 because  $\pi$  is independent of the observations received.

## 4.5 Upper Bounds

To obtain an upper bound on the optimal expected reward, one can solve the underlying MDP for each possible interaction graph and take the largest expected reward. For ND-POMDPs with a given interaction graph, one can calculate the upper bound  $V(b)$  for initial belief  $b$  using the expected rewards  $V(s)$  for starting states  $s$ :

(a) Random Trajectories			
Max Sensors	Naive Greedy	Heuristic Greedy	MILP
5	4.0	4.0	4.0
7	6.0	7.0	6.0
9	8.5	10.0	8.5
11	10.0	13.5	11.0
13	10.0	17.0	11.5
15	11.0	18.5	13.0

(b) Fixed Trajectories			
Max Sensors	Naive Greedy	Heuristic Greedy	MILP
5	4.0	4.0	5.0
7	7.0	7.0	8.0
9	9.0	10.0	10.0
11	12.5	12.5	12.0
13	15.5	14.5	14.5
15	16.5	16.0	16.0

**Table 1: Number of Edges in the Interaction Graphs**

$$V(b) = \sum_{s \in S} b(s) \cdot V(s) \quad (9)$$

$$V(s) = \max_{a \in A} \left\{ R(s, a) + \sum_{s' \in S} P(s'|s, a) \cdot V(s') \right\} \quad (10)$$

$$= \max_{a \in A} \left\{ \sum_{e \in E} R_e(s, a) \right\} + \sum_{s' \in S} P(s'|s) \cdot V(s') \quad (11)$$

where  $E$  is the set of edges in the given interaction graph. Eq. 10 simplifies to Eq. 11 because all the states in our problem are uncontrollable states that are independent of the action of agents. If not all states are uncontrollable states, then one can still calculate an upper bound on the expected reward of an optimal joint policy for the MDP [7] or use existing techniques to solve factored MDPs [3].

## 5. EXPERIMENTAL EVALUATION

For our experiments, we compare the greedy and MILP-based algorithms on a problem with 25 possible sensor placement locations arranged in a 5x5 grid as in Figure 1. We model the problem as an ND-POMDP as one example model within the VF framework. We vary the maximum number of available sensors from 5 to 15 to investigate the scalability of the algorithms. We set the number of targets to track to 2 and experiment with two types of target trajectories – fixed trajectories, where each target can stay at its current location with probability 0.1 and move to the next location in its trajectory with probability 0.9, and random trajectories, where each target can stay at its current location or move to any neighboring location with equal probability – to simulate problems where target trajectories are known and unknown, respectively.

We use CDBP [7], a state-of-the-art ND-POMDP algorithm, as a subroutine in the greedy and MILP-based algorithms to compute joint policies. We set the horizon to 10 and the number of samples to 5. We conduct our experiments on a quad core machine with 16GB of RAM and 3.1GHz CPU, and set a cut-off time limit of 40 hours. Figures 4 and 5 show the results for problems with random and fixed target trajectories, respectively, and Table 1 shows the size of interaction graphs built.

### 5.1 Runtime Discussions

The runtimes are similar for both problem types. The only exception is that the upper bound computations are

one order of magnitude slower on random trajectory problems than on fixed trajectory problems. This result is to be expected since there is a larger number of state transitions with non-zero probabilities in random trajectory problems and, as such, more computation is necessary.

For both problem types, Naive Greedy runs up to one order of magnitude longer than the Heuristic Greedy and MILP-based algorithms. The reason is that Naive Greedy runs CDBP multiple times each time it adds an edge to the interaction graph. (It runs CDBP each time it evaluates a candidate edge.) On the other hand, the Heuristic Greedy and MILP-based algorithms run CDBP only once to compute the joint policy for its interaction graph.

For random trajectory problems, the MILP-based algorithm is faster than the Heuristic Greedy algorithm except for problems that are very small (problems with 5 sensors). The reason for this behavior is the following: In these problems, there is a large number of edges with non-zero probabilities that a target will be at those edges since the targets are performing random walks. Additionally, the Heuristic Greedy algorithm uses heuristic values (in Eq. 4) that assume that the sensors at locations connected by an edge *will* coordinate with each other to get the reward at that edge (since we are taking the maximum over all joint actions in that edge). Therefore, the algorithm will add a large number of edges with non-zero probabilities to its interaction graph as long as adding those edges do not require more sensors than available.

On the other hand, the MILP-based algorithm takes into account that sensors can only coordinate with at most one neighboring sensor at each time step in choosing its edges. (Each agent must have the same action for all value factors.) Thus, the number of edges in the MILP interaction graph is smaller than that in the Greedy Heuristic interaction graph, as shown in Table 1(a). Thus, the runtime of CDBP on the MILP interaction graph is also smaller than on the Greedy Heuristic interaction graph.

For fixed trajectory problems, the Heuristic Greedy algorithm is faster than the MILP-based algorithm. The reason for this behavior is the following: In these problems, there is a small number of edges with non-zero probabilities that a target will be at those edges since the targets are always in one of the edges in their respective trajectories. Therefore, the Heuristic Greedy algorithm will add a small number of edges to its interaction graph. The MILP-based algorithm is also able to exploit this property and only include a small number of edges to its interaction graph. Thus, the number of edges in the MILP interaction graph is similar to that in the Greedy Heuristic interaction graph, as shown in Table 1(b). Thus, the runtime of CDBP on both interaction graphs are also similar.

However, on the computational effort in generating the interaction graph, the Heuristic Greedy algorithm is more efficient than the MILP-based algorithm – Heuristic Greedy takes approximately 0.5 seconds while the MILP-based algorithm takes approximately 25 seconds. Therefore, this difference is more noticeable when the problems are small (as the runtimes of CDBP are small) and diminishes as the problems become larger.

### 5.2 Expected Reward Discussions

The expected rewards of joint policies found for random trajectory problems are smaller than those found for fixed



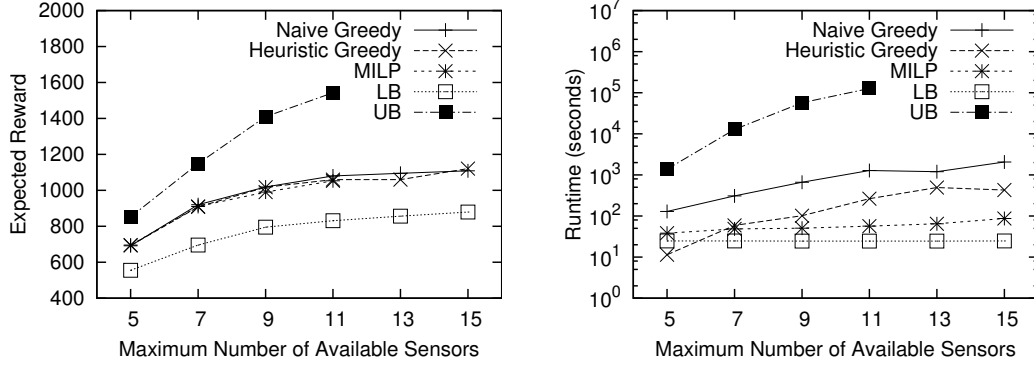


Figure 4: Results for Random Trajectories

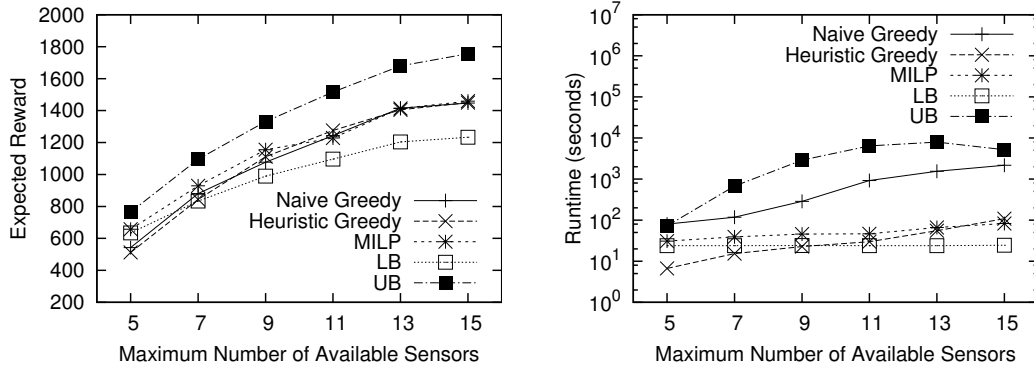


Figure 5: Results for Fixed Trajectories

trajectory problems, which is to be expected since there is a larger entropy in the random trajectory problems.

For both problem types, all three algorithms find comparable joint policies except for problems with 5 maximum available sensors. In these problems, both greedy algorithms found joint policies with expected rewards that are about 20% smaller than the expected rewards of joint policies found by the MILP-based algorithm. The reason is that the first two edges chosen by the greedy algorithm typically correspond to the starting edges (locations) of the two targets. These edges are typically disjoint, and four sensors are thus placed just to track the targets along these two edges. On the other hand, the MILP-based algorithm typically places the first four sensors more efficiently by placing them in a square, and they can thus track targets along the four edges of the square. Thus, the joint policies found by the MILP-based algorithm have larger expected rewards than those found by the greedy algorithm. Table 1(b) shows this behavior, where the MILP interaction graph has 25% more edges than the Naive Greedy and Heuristic Greedy interaction graphs on problems with 5 sensors.

For both problem types, the lower bounds are at most 20% smaller than the expected rewards of the joint policies found by the MILP-based algorithm, which are at most 35% smaller than the upper bounds. The lower bounds have smaller expected rewards since they are expected rewards of open loop joint policies. The upper bounds have larger expected rewards since they are expected rewards of joint policies on fully observable problems. These bounds are tighter

in fixed trajectory problems than in random trajectory problems since fixed trajectory problems are simpler.

Lastly, we also exhaustively enumerated all possible interaction graphs for small problems where it is possible to do so, and in all of those cases, both the optimal and MILP-based graphs are *very* similar.

In summary, our experimental results show that the Heuristic Greedy and MILP-based algorithms are good ways to automatically generate interaction graphs and find joint policies that are within reasonable error bounds. The Heuristic Greedy algorithm is better suited for problems with little transition uncertainty and the MILP-based algorithm is better suited for problems with more transition uncertainty. Furthermore, one can use the *open loop* joint policies if there is an insufficient amount of time to compute better *closed loop* joint policies.

## 6. CONCLUSIONS

The VF framework has been shown to be a general framework that subsumes many sparse-interaction Dec-POMDP models including ND-POMDPs. Existing algorithms for these models assume that the interaction graph of the problem is given. However, there have been no studies on the automated generation of interaction graphs. In this paper, we introduced two greedy algorithms and a MILP-based algorithm to automatically generate interaction graphs and establish lower and upper bounds on the expected reward of an optimal joint policy. The greedy algorithms greedily adds value factors (or edges) to the interaction graph based

on their actual or estimated contribution to the expected reward. The MILP-based algorithm generates an interaction graph by choosing value factors to form an interaction graph such that an optimal open loop joint policy on that interaction graph is optimal across all possible interaction graphs.

Our experimental results show that they find reasonable joint policies (their expected rewards are at least 65% of a loose upper bound). The Heuristic Greedy algorithm is faster than the MILP-based algorithm in problems with less transition uncertainty and vice versa in problems with more transition uncertainty. In conclusion, we examined the challenge of automatically generating interaction graphs and offered several general methods that performed well in a sensor network coordination testbed. These methods offer a foundation for further exploration of this area.

## 7. REFERENCES

- [1] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004.
- [2] D. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.
- [3] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1523–1530, 2001.
- [4] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1332–1339, 1999.
- [5] M. Krainin, B. An, and V. Lesser. An application of automated negotiation to distributed task allocation. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pages 138–145, 2007.
- [6] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284, 2008.
- [7] A. Kumar and S. Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 561–568, 2009.
- [8] A. Kumar, S. Zilberstein, and M. Toussaint. Scalable multiagent planning using probabilistic inference. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2140–2146, 2011.
- [9] V. Lesser and D. Corkill. The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, 1983.
- [10] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 133–139, 2005.
- [11] F. Oliehoek. *Value-Based Planning for Teams of Agents in Stochastic Partially Observable Environments*. PhD thesis, University of Amsterdam, Amsterdam (Netherlands), 2010.
- [12] F. Oliehoek, M. Spaan, S. Whiteson, and N. Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 517–524, 2008.
- [13] Z. Rabinovich, C. Goldman, and J. Rosenschein. The complexity of multiagent systems: The price of silence. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1102–1103, 2003.
- [14] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2009–2015, 2007.
- [15] D. Szer and F. Charpillet. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1233–1238, 2006.
- [16] P. Velagapudi, P. Varakantham, P. Scerri, and K. Sycara. Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 955–962, 2011.
- [17] S. Witwicki and E. Durfee. From policies to influences: A framework for nonlocal abstraction in transition-dependent Dec-POMDP agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1397–1398, 2010.
- [18] S. Witwicki and E. Durfee. Towards a unifying characterization for quantifying weak coupling in Dec-POMDPs. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 29–36, 2011.

# Opponent modeling and planning against non-stationary strategies

Pablo Hernandez-Leal, Enrique Munoz de Cote and L. Enrique Sucar

Instituto Nacional de Astrofísica, Óptica y Electrónica

Luis Enrique Erro #1

Sta. Maria Tonantzintla, Puebla, México

{pablohl,jemc,esucar}@ccc.inaoep.mx

## ABSTRACT

In multiagent systems, in order to make the best decisions, each agent has to take into account not only the strategy used by other agents but also how those strategies might change in the future. This is further exacerbated in open environments where strategies cannot be assumed to be rational. This paper studies repeated interactions between an agent and an opponent that changes its strategy over time (it is non-stationary). Our main contribution is a framework for fast learning changing non-stationary strategies. The agent uses decision trees to learn the most up to date opponent's strategy. The agent's learned model is continuously re-evaluated to assess strategy switches. Our method detects such switches by measuring tree similarities. Aside from its fast learning process, decision trees can provide an easy interpretation of the opponent model. A second contribution is with regards to computing a policy against the opponent making the best use of the generated opponent model. For this, we propose a novel approach of transforming a decision tree into a Markov Decision Process. We evaluated the proposed approach in the iterated prisoner's dilemma, outperforming state of the art algorithms in predictive accuracy when facing non-stationary strategies.

## Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Multiagent Systems

## General Terms

Algorithms

## Keywords

Opponent modelling; Decision Trees; Markov Decision Process; Iterated Prisoner's Dilemma

## 1. INTRODUCTION

When agents operate in open environments they should be able to interact effectively with an unknown agent. In order to deal with such uncertainty, it is common practice to assume that the counterpart's strategy is rational and stationary. However, is not a reasonable assumption and must be revisited.

---

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with *AAMAS*, May 2013, St. Paul, Minnesota, USA.

Game theory's approaches typically rely on full rationality assumptions [7]. Other approaches assume that the opponent is stationary or will converge to a stationary policy [2]. Furthermore, most of the current approaches need long sequences of interactions for learning a model. All these are important limitations when facing real-world scenarios or when interacting with a human.

In this work we consider repeated interactions between an agent and an opponent that changes its strategy over time (it is non-stationary) and propose a framework for fast learning changing non-stationary strategies. The approach is a two-phase process, first it learns decision trees to assess the opponent's strategy. Second, it uses a novel approach in which the assessment (decision tree) is transformed into a Markov Decision Process (MDP), whose solution prescribes the optimal way of playing against the learned strategy. The agent's learned model is continuously re-evaluated to detect strategy switches. The method detects strategy switches by measuring tree similarities revealing whether the opponent has changed its strategy and a new model has to be learned.

Our contribution is twofold: a framework for fast learning changing non-stationary opponent model strategies and a novel way of transforming a decision tree into a MDP. Additionally, because the learned models of the opponent are decision trees, they provide an easily interpretation of the model, which can be useful if a compact representation of strategies is needed. We evaluate the performance of this approach in the iterated prisoner's dilemma, outperforming state of the art algorithms in predictive accuracy against non-stationary strategies.

The paper is organized as follows. In Section 2 we present the state of the art algorithms that have a direct relation with the problem of learning opponent models facing non-stationary strategies. In Section 3 we formally describe the problem setting. In Section 4 we introduce our proposed framework. Following this, in Section 5 we present the setting for our experiments and the obtained results. In Section 6 we present conclusions and directions for future research.

## 2. RELATED WORK

Opponent modelling is a problem that lies in the intersection of different areas like game theory, learning and planning.

From the area of game theory, *fictitious play* [4] was one of the first approaches for learning a model of the opponent. The model maintains a count of plays by the opponent in the past to assess the opponent's strategy and best responds to that estimate. FP assumes its opponent plays a stationary

strategy and the observed frequencies are taken to represent the opponent’s mixed strategy.

Approaches from reinforcement learning have been extended to multi agent scenarios [5]. *R-max* [3] is a model based reinforcement learning algorithm that always maintains a complete but possibly inaccurate model. The model is initialized in an optimistic fashion by assuming all actions return the maximum possible reward. In repeated games there is only one parameter needed. A threshold on the number of times a pair (*state*, *action*) has to occur to count that (*state*, *action*) as known. R-max is an example of a model based RL algorithm that makes good use of its model to guide exploration, therefore needs less interactions with the environment to converge to an optimal policy. Although R-max addresses the problem of faster convergence, its basic limitation is that it is designed for two agents in a competitive environment.

A different approach is presented in [9] where the authors propose two strategies that influence the best response of a follower agent in repeated games. The first strategy is a deterministic, state-free policy called *Bully* that chooses the action that maximizes its reward given that its opponent is playing a best response. The second strategy called *Godfather* is a generalization of Tit-for-tat that offers the opponent a situation where it can obtain a high reward. If the opponent does not accept the offer, Godfather forces the opponent to obtain a low reward.

The Fast Adaptive Learner (FAL) [6] approach is a recently proposed approach composed of two parts, the first one is a predictive model called Entropy Learning Pruned Hypotheses (ELPH) [8]. ELPH maintains a set of hypotheses according to a history of observations, these hypotheses increment exponentially in the number of possible observations and history length, thus a pruning step is performed. This prediction is used as a model of the other agent. To obtain a strategy to be used against the opponent, a modified version of Godfather is used. One limitation is the exponential increase in the number of hypotheses; another is that the Godfather strategy is not a general strategy that can be used against any opponent.

Another recent work is presented in [1]. The attacked problem is one where an agent is forced to work in a team with other three unknown agents in order to complete a specific task. Therefore, the agent needs to build a model of its teammates to plan its future behavior. To learn the models, supervised learning algorithms were used and to perform planning a Monte Carlo tree search approach is used. A disadvantage is that the learning of models is performed in an off-line fashion, only the belief’s update is online.

Our approach has certain similarities with the previous approaches. As R-max, FAL and Barret et al.’s approach, we also divide the task in two parts: the former will construct a model of the other agent, and the latter will use that model to perform planning to obtain decisions on how to act. Our framework differs from others in that to build a model we use decision trees, based on C4.5 [12]. This has the advantage of being a fast approach that can detect strategy switches with a transparent representation. For the planning phase we transform the tree into a MDP, which is a general model that guarantees an optimal policy.

### 3. PROBLEM DESCRIPTION

We consider two players (A and B), that face each other

**Table 1: A common bimatrix game known as the prisoner’s dilemma. Each cell represent the utilities given for the agents (first the row player and then the column player)**

	$b_1$	$b_2$
$a_1$	3,3	4,0
$a_2$	0,4	1,1

and repeatedly play a *bimatrix game*.

**DEFINITION 1.** A *bimatrix game* is a two player simultaneous-move game defined by the tuple  $\Gamma = \langle \mathcal{A}, \mathcal{B}, R_A, R_B \rangle$ , where

- $\mathcal{A}$  and  $\mathcal{B}$  are the set of possible actions for player A and B respectively.
- $R_i$  is the reward matrix of size  $|\mathcal{A}| \times |\mathcal{B}|$  for each agent  $i \in \{A, B\}$ , where the payoff to the  $i$ th agent for the joint action  $(a, b) \in \mathcal{A} \times \mathcal{B}$  is given by the entry  $R_i(a, b)$ ,  $\forall (a, b) \in \mathcal{A} \times \mathcal{B}$ ,  $\forall i \in \{A, B\}$ .

In our setting, a bimatrix game is called the *stage game* and it is the building block of a *repeated game*. At stage one, each agent simultaneously chooses an action and a pair  $(a, b) \in \mathcal{A} \times \mathcal{B}$  is formed, announced to all agent, then each agent  $i$  receives a reward from  $R_i$ . The process then repeats for all following stages. An example of a bimatrix representing the prisoner’s dilemma is presented in Table 1.

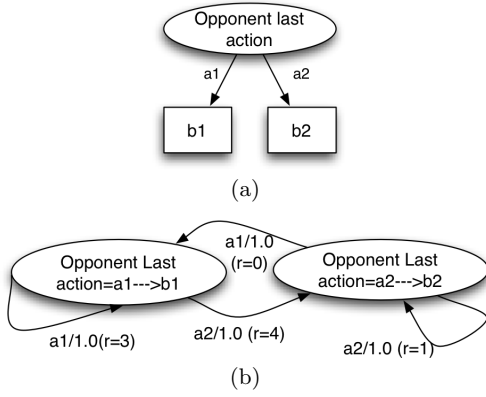
## 4. MDP4.5

We propose a framework (coined MDP4.5) for learning switching non-stationary strategies in repeated games using a novel approach based on decision trees and MDPs. In a nutshell, we use decision trees to model opponent strategies then transform these into a MDP to obtain an optimal strategy against that opponent. Since we care for opponents that change their strategy, we propose a way of detecting strategy switches through the game and recompute a strategy to counteract the switch.

### 4.1 Learning: opponent strategy assessment

Consider the case where agent A is the modeling agent and agent B is an opponent that use a non-stationary strategy throughout the game. To promote exploration, agent A will start playing a random strategy to learn a model of the opponent with a decision tree (using C4.5).

A decision tree  $D = (V, E)$  specifies some test to be carried out on each attribute, with one branch and one subtree for each possible outcome of the test. From the set of vertices  $V$ , we use the inner vertices (not leafs) for the set of attributes  $O$ , which will be assumed to be given by an expert and use the set of opponent actions  $\mathcal{B}$  for leafs. We denote as  $class(l)$  to the value in leaf  $l$ . Additionally, each leaf has two associated values,  $classified(l)$  is the number of correctly classified instances and  $misclassified(l)$  is the number of incorrectly classified instances. Each path  $p_l$  traversing decision nodes and arriving into a leaf  $l$  is a unique decision rule. Thus, there is a one-to-one correspondence between a path  $p_l$  and the leaf  $l$ . In Figure 1 (a) a decision tree with just one decision node and two leaves is depicted. The leaves



**Figure 1:** (a) A decision tree that corresponds to a model of an agent. It contains one decision node *Opponent Last action* and two leaves that correspond to the actions  $b_1$  and  $b_2$ . (b) The MDP obtained from decision tree in (a) using the game matrix of Table 1. It is composed of two states (ovals). The arrows represent the transition probabilities using action  $a_x$ , with the immediate reward in parenthesis.

of the tree represent the opponent’s next action and edges represent the decision rules (later used to plan a strategy).

The learned tree describes the opponent strategy based on the window of interactions between agents. The designer is free to include any attribute for learning the tree. For the iterated prisoner’s dilemma we opted for the previous plays from both agents and the round number, the class is the opponent’s next play.

## 4.2 Planning: Mapping decision trees to MDPs

Once we have learned a model of the opponent, we need to be able to play against such strategy. To do so, we propose a novel way to transform decision trees into MDPs. By so doing, we can now compute an optimal policy by solving the MDP model<sup>1</sup> of the opponent. A decision tree that contains as decision nodes the last  $n$ -actions induces a MDP (later we show why this is true). Surely, if the model correctly represents the opponent’s strategy, then the solution will produce an optimal strategy against that opponent.

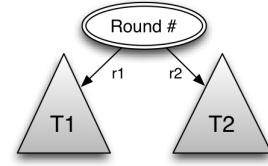
In more detail the induced MDP (by a decision tree  $D$  and a bimatrix game  $\Gamma$ ) is a tuple  $\langle S, \mathcal{A}, T, R_A \rangle$  where,

- $\mathcal{A}$  and  $R_A$  are heritage from  $\Gamma$ ,
- the set of states  $S := P \times \mathcal{B}$ , i.e. each state is formed by a path and an action of the opponent,
- the transition function  $T : S \times \mathcal{A} \rightarrow S$  is generated as follows, for each  $s \in S$  and  $a \in \mathcal{A}$ ,

$$T(s, a, s') = \begin{cases} \text{classified}(l) & \text{if } s' \in P \\ \frac{\text{misclassified}(l)}{|\mathcal{B}|-1} & \text{other case} \end{cases}$$

As an example, the tree of Figure 1 (a) represents the model of an opponent with two actions  $b_1, b_2$ . Converting this tree and using the classic matrix of the prisoner’s

<sup>1</sup>Solving an MDP can be done with any off the shelf dynamic programming algorithm, like value iteration [11].



**Figure 3:** A decision tree that has *Round #* as decision node. Here, each branch (T1 and T2) will be transformed into two independent MDPs.

dilemma (Table 1) will yield the MDP depicted in Figure 1 (b). The decision tree in (a) can be augmented to include new leaves (corresponding to misclassified instances) as depicted in Figure 2 (a). These new leaves share the path from the original leaf but replace the class with the other possible values. In the example, the (original) left leaf has the value  $b_1$  and this leaf classified correctly 90% of the times. The other 10% corresponds to a different value, in this case  $b_2$ , therefore this new leaf is added to the tree and presented in dotted arrows. Notice that the original leaves correspond to the set  $P$  and new leaves correspond to the set  $P'$ . Converting this augmented tree and using the classic matrix of the prisoner’s dilemma (Table 1) will yield the MDP depicted in Figure 2 (b).

## 4.3 Large state space MDPs

It can be the case that an opponent is wired to switch its strategy at specific times and the learned decision node will contain a feature like *round* (see Figure 3) that can not be mapped easily to an MDP because it would require increasing the number of states for each possible outcome of the attribute. When this happen we use a hybrid approach. Since each branch of the tree is also a tree, then the branches of the *round* node are treated as separate trees and converted into independent MDPs. Therefore a learned decision tree has the potential to induce a set of MDPs, where each MDP represents a different strategy used by the opponent and the strategy switch happens around the time step in decision node “round”.

With this approach we can tackle a number of different domains since it is a general approach and optimal, given that the learned model is correct.

## 4.4 Assessing strategy switches

As said earlier, we use a two phase approach. The objective in the first phase is to assess the strategy used by the opponent and use a random strategy throughout this phase. On the second phase the modeler computes a strategy of play, therefore switching its previously used (random) strategy to the one that optimizes against that opponent. However, the switch might trigger a response from the opponent and in order to detect responses as strategy switches we need to be able to detect changes in the learned model.

With this in mind, we keep two decision trees learning concurrently. One of these (tree  $a$ ) will learn for  $w$  steps and will be reset afterwards, while the other (tree  $b$ ) is never reset (keeping the entire history). Trees  $a$  and  $b$  are compared every  $w$  steps (at steps  $k.w$  for  $k = 2, 3, \dots$ ) to evaluate their *similarity*. For this, we use a distance metric presented in [10] which let us balance out the structure (the attributes of the nodes) and predictive (the predicted classes) similarities.

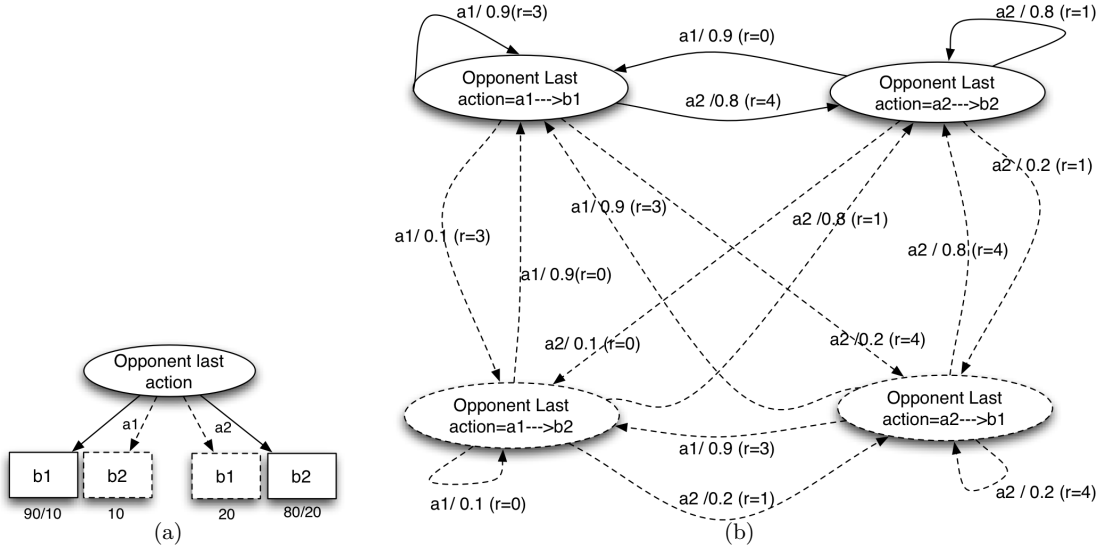


Figure 2: (a) The augmented decision tree of Figure 1 (a) that contains in dotted lines two added leaves representing the classification errors. (b) The MDP obtained from (a) using the game matrix of Table 1. It is composed of four states. The dotted arrows and ovals correspond to the added actions depicted in (a).

If the *distance* between these trees is greater than a threshold  $\delta$ , the opponent has changed strategy and the modeling agent must restart the learning phase, resetting both trees and starting from scratch. Otherwise, it means that the opponent has not switched strategies and  $k$  is incremented waiting for a new batch of interactions.

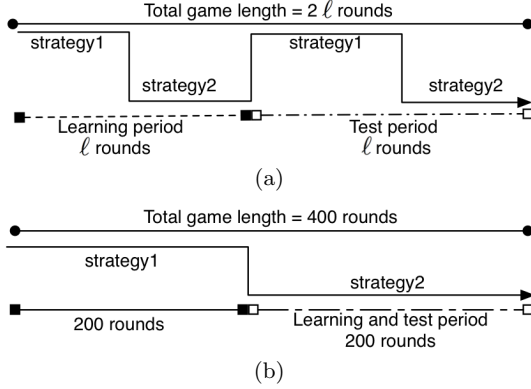


Figure 4: Graphical representation of the experiments. In (a) a deterministic switching opponent, in (b) a non-deterministic switching opponent.

## 5. EXPERIMENTS AND RESULTS

Because the first phase (model assessment) is crucial for the success of any model based strategy, these first experiments are only focused on evaluating how the first phase of the approach fares in direct comparison with state of the art algorithms like R-max and ELPH. If the model is accurate the planning is guaranteed to be optimal. In Section 5.4 we present experiments with involving the two phases, learning and planning.

Now, R-max was selected because is a model-based reinforcement learning algorithm renowned by its efficient use of information to guide its exploration. ELPH was selected because beside reporting to have fast learning rates it also constructs a model of the opponent. The experiments were performed under the iterated prisoner's dilemma scenario, since it is a largely studied problem with many reported strategies that fare well against arbitrary opponents. MDP4.5, R-max and ELPH all generate a model that can be used to predict the opponent's next action. It is this predictive accuracy that will be used as a measure of the quality of the learned model. The predictive accuracy will be measured as the ratio between the number of correct predicted opponent actions and the total number of interactions in a test period.

For MDP4.5, the attributes of the tree are the last step joint action, and the round number of interaction. For ELPH, the algorithm constructs a set of hypotheses and the most probable one is the one we select as the predicted one. For its parameters, history size was set to 1 and the observations were the opponent and agent last action, no pruning

was performed. For R-max, we used the last round's joint action and the actions, rewards and  $Rmax$  value were obtained from the game matrix. We evaluated different values for parameter  $m$  that controls the number of times each state-action pair must be experienced before updating the model and used the one that performed best. For R-max, the transition from one state to another gives the prediction of the next action.

Because we are only evaluating model learning the three algorithms will use random actions throughout their interaction to promote fair exploration. Each tested algorithm was paired against common strategies such as Tit-for-Tat (TFT), Pavlov, Tit-for-two-tats (TFTT) and the Bully strategy. TFT, Pavlov and Bully use one step of interaction to compute its play, in contrast to TFTT which keeps two steps of interaction. This means that all algorithms should be able to correctly model all opponents but TFTT. Furthermore, all the opponents include a .03 chance of choosing a random action, not following the prescribed one.

### 5.1 Offline learning: predicting when strategies will switch

The first set of experiments evaluate the algorithms' ability to identify strategy switches, plus predicting when will these switches take place. To achieve this, the opponent starts with a strategy drawn from the set {TFT, Pavlov, TFTT} and will deterministically (at some specific time step) switch to a strategy from the same set but without repetition. A graphical representation of the experiment is depicted in Figure 4 (a).

There are two stages of interactions: the first one is the *learning stage* where the learning algorithms interact with the opponent over  $\ell$  steps. After this, the *test stage* is designed to evaluate learned opponent models. Both phases share the same number of interaction steps, and the strategy switch occurs at the same time step  $d$ , where  $0 < d < \ell$ . Then, we tested different windows of interaction  $\ell$  from 10 to 100 in increments of 10. In particular, the switch was wired to happen at time step  $d = \ell/2$ . For example with  $\ell = 50$  and against the TFT-Pavlov opponent, the TFT strategy is used the first 25 interactions and Pavlov strategy is used from round 26-50.

In Table 2 we can observe the predictive accuracy of each tested algorithm against all the combinations of pairs of strategies (without repetition). In this setting, in all cases, R-max (using  $m = 2$ ) needed longer sequences of steps to improve its scores but never reached the best score. MDP4.5 has the best predictive scores after a period of 20 or 30 steps. ELPH obtained the best scores between 10-20 steps, however their scores do not improve throughout. In contrast MDP4.5 and R-max have a similar learning rate however MDP4.5 obtained better scores.

### 5.2 Online learning: detecting when strategies switch

We presented experiments against switching deterministic opponents. Now we study non-deterministic switches. The objective of this setting is to assess how fast an algorithm can detect a strategy switch when this happens without prior knowledge. This is specially useful for situations where the opponent changes it strategy only once in a lifetime (not changing back to the previous strategy). While this appears as a very simple behavior, it presents serious difficulties for

**Table 3: Average predictive accuracy scores for the three compared approaches against a non-deterministic switching strategy. Each row is the average of 100 games**

	MDP4.5, $\delta = 0.0$		Rmax	ELPH
	$w = 25$	$w = 30$	$m = 4$	
TFT-Pavlov	<b>0.779</b>	0.753	0.503	0.541
TFT-TFTT	0.581	0.593	0.736	<b>0.743</b>
Pavlov-TFTT	0.566	<b>0.590</b>	0.501	0.508
Pavlov-TFT	0.798	<b>0.806</b>	0.514	0.536
TFTT-TFT	0.828	0.796	0.795	<b>0.863</b>
TFTT-Pavlov	<b>0.763</b>	0.758	0.344	0.617
TFT-Bully	0.803	<b>0.807</b>	0.504	0.736
Pavlov-Bully	0.801	<b>0.806</b>	0.505	0.734
TFTT-Bully	0.803	0.806	0.478	<b>0.840</b>
Bully-TFTT	0.501	0.560	0.610	<b>0.702</b>
Bully-Pavlov	<b>0.760</b>	0.759	0.646	0.731
Bully-TFT	<b>0.796</b>	0.787	0.652	0.728
Average	0.731	<b>0.735</b>	0.566	0.690

most learning algorithms.

A graphical representation of the experiment is depicted in Figure 4 (b). Just as before, the opponent has two strategies to be drawn without repetition from the set {TFT, Pavlov, TFTT, Bully}. The game consisted of  $\ell = 400$  rounds and in time step  $d = \ell/2$ , the opponent switched its strategy to a different one. In contrast to the previous setting, there is no offline learning phase. Instead, the process of determining the switch and learning the model is all done online. Thus, the evaluation of the predictive score begins in the 200 round, when the strategy switches.

In Table 3 we present the predictive scores for the different opponents for the three algorithms. Each row is the average of 100 games. In particular for MDP4.5 we present different results setting  $w$  to 25 and 30 as the window used for learning the tree, as these values seemed to work best. For R-max, we experimented with different values of  $m$  from 2 to 5 and selected  $m = 4$  since it obtained the best scores.

From the table we can see that MDP4.5 with any of two window sizes obtained better scores than R-max and ELPH in most cases and in average. We also evaluated the parameter  $\delta$  from 0.0 to 0.4 in increments of 0.1. The best scores were obtained with 0.0 and the results showed that increasing  $\delta$  decreased the predictive score since the difference between trees has to be larger to determine a change in the opponent strategy. Increasing the size of the window from 25 to 30 had similar results. This happens because, the learned models with 25 rounds are almost as good as the ones with 30 rounds, therefore our framework is capable of detecting the switch in strategies and learn the new one. It is important to note that there are some tradeoffs to consider in our framework. On one side, the larger the window, the better the learned model; however, the time to determine changes between models also increases. On the other side, the size of the threshold is related to the sensitivity in determining strategy switches. A extremely small value may detect not existing changes, while a large value may not detect real changes. R-max obtained very similar results while varying the parameter  $m$ , however they were lower and far from those obtained by MDP4.5 and ELPH. Finally, ELPH obtained better scores when the strategies do

Table 2: Offline learning: average predictive accuracy scores for the three compared approaches against a deterministic switching strategy.  $\ell$  represents the size of the learning and test phase. Each row is the average of 100 games.

	TFT-Pavlov			TFT-TFTT			Pavlov-TFTT		
$\ell$	MDP4.5	R-max	ELPH	MDP4.5	R-max	ELPH	MDP4.5	R-max	ELPH
10	61.700	38.600	<b>72.800</b>	68.600	22.600	<b>75.600</b>	55.800	43.450	<b>64.900</b>
20	<b>77.450</b>	62.175	73.150	81.600	45.433	<b>85.200</b>	64.050	60.925	<b>70.100</b>
30	<b>87.300</b>	71.367	75.333	<b>84.767</b>	58.500	82.867	<b>73.133</b>	67.500	72.000
40	<b>91.750</b>	72.625	74.325	<b>86.525</b>	66.560	86.000	<b>77.375</b>	71.288	72.000
50	<b>93.860</b>	72.910	73.740	<b>87.920</b>	69.900	85.660	<b>82.640</b>	71.750	72.460
60	<b>93.833</b>	73.267	73.317	<b>88.183</b>	71.921	85.733	<b>85.650</b>	73.275	72.217
70	<b>95.043</b>	74.507	73.700	<b>88.657</b>	73.806	86.186	<b>85.529</b>	72.143	72.343
80	<b>95.938</b>	73.894	74.763	<b>89.750</b>	75.544	85.725	<b>87.338</b>	73.556	72.163
90	<b>96.422</b>	73.439	74.467	<b>90.156</b>	76.940	86.244	<b>87.200</b>	73.428	72.600
100	<b>96.150</b>	73.255	74.000	<b>90.570</b>	85.135	86.440	<b>88.790</b>	73.820	73.240
Average	<b>88.945</b>	68.604	73.959	<b>85.673</b>	64.634	84.566	<b>78.750</b>	68.113	71.402

	Pavlov-TFT			TFTT-TFT			TFTT-Pavlov		
$\ell$	MDP4.5	R-max	ELPH	MDP4.5	R-max	ELPH	MDP4.5	R-max	ELPH
10	65.500	40.150	<b>71.600</b>	69.900	43.850	<b>76.600</b>	57.900	43.350	<b>69.900</b>
20	<b>77.100</b>	63.375	74.950	<b>84.900</b>	65.925	82.500	67.950	55.375	<b>70.250</b>
30	<b>85.833</b>	72.050	75.067	<b>85.633</b>	74.233	85.167	<b>76.833</b>	61.683	70.833
40	<b>90.150</b>	73.525	74.450	<b>86.475</b>	79.125	85.600	<b>83.200</b>	61.438	71.925
50	<b>92.740</b>	73.640	73.300	<b>85.800</b>	78.760	85.620	<b>85.020</b>	62.690	72.460
60	<b>92.983</b>	73.233	74.267	<b>88.683</b>	80.467	86.200	<b>85.117</b>	60.633	71.867
70	<b>95.657</b>	74.100	73.486	<b>88.371</b>	80.750	85.986	<b>86.929</b>	61.214	71.971
80	<b>96.125</b>	73.719	75.125	<b>90.450</b>	80.444	85.925	<b>87.863</b>	60.200	72.888
90	<b>96.344</b>	72.983	74.667	<b>90.756</b>	79.628	86.289	<b>88.111</b>	59.378	73.067
100	<b>96.320</b>	73.460	74.710	<b>90.500</b>	80.180	86.650	<b>88.950</b>	58.250	73.410
Average	<b>88.875</b>	69.024	74.162	<b>86.147</b>	74.336	84.654	<b>80.787</b>	58.421	71.857

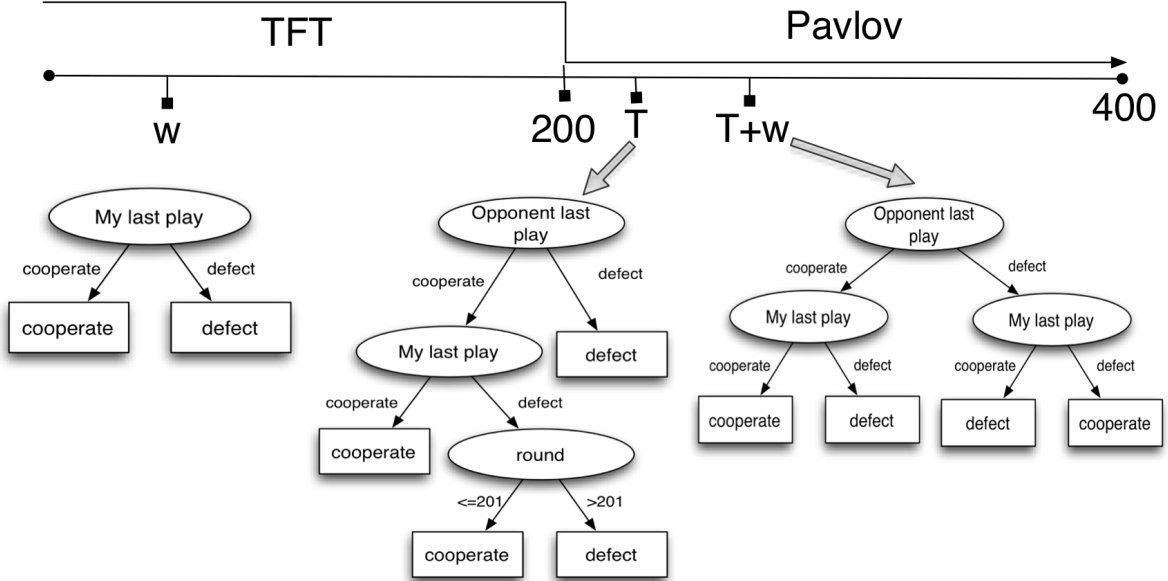


Figure 5: A set of trees learned against the opponent TFT-Pavlov. From left to right, the first learned tree represents the TFT strategy. The second tree is represents when the opponent is switching strategies. The last tree, represents the Pavlov strategy.



not differ much like TFT-TFTT and TFTT-TFT or when the change involved Bully and TFTT, this may be attributed to the other algorithms not being able to learn the TFTT strategy (since it uses two steps of history) and thus it is more difficult to detect a switch in strategy.

### 5.3 Exemplifying the learned trees

Now we present how the learned models can be of use by an expert to construct better representations. Specifically, we present a set of learned trees against the opponent TFT-Pavlov obtained from the experiments presented in the previous section. These trees exemplify three important periods and they are depicted in Figure 5. The first one (from left to right) is the initial learned tree at step  $w$  and represents the TFT strategy. In step 200, the opponent switches from TFT to Pavlov and our approach, learns a tree at step  $T$ , that detects the switch, this can be noticed in the *round* decision node in the second tree. Since the previous two thirds result in a distance greater than the specified threshold then the exploration phase restarts. The third tree is the learned model after the switch (at step  $T+w$ ) and represents the Pavlov strategy.

### 5.4 Learning and planning

In the previous experiments the algorithms learned a model of the opponent but played randomly. In this section we present results using the two phases of the approach. The algorithms will learn a model and will use to perform planning and act accordingly.

In this setting we evaluated the average reward obtained by MDP4.5 and R-max in the iPD against non-stationary opponents. For MDP4.5 we set  $\delta = 0.0$  and  $w = 30$ . When solving the models we used value iteration with  $\gamma = 0.9$  and  $\epsilon = 0.0001$ . The game length was 400 steps. As in section 5.2 the opponent has two strategies and will change from one to another in the middle of the game.

In Table 4 we present the average rewards obtained for MDP4.5 and R-max against a switching opponent in a game of 400 steps. For the first experiment the rewards are the ones presented in Table 1, *cooperate* = 3, *defect* = 1, *sucker* = 0 and *temptation* = 4 for the second experiment we changed *temptation* = 5 to evaluate if the policies obtained by the MDPs were different.

From the results we note that against Pavlov-TFT our approach obtains higher scores but it is important to analyze also the reward obtained by the opponent. When  $T = 4$  both agents against obtain similar rewards, in contrast when  $T = 5$ , the MDP4.5 agent obtains a higher reward and the opponent a lower one. This happens because when  $T = 4$ , the best thing to do is to cooperate, therefore the agents stay in C-C. If  $T = 5$  then, the best thing to do against a Pavlov strategy is to defect always, this results in a loop between D-C and D-D. This behavior will occur in all the cases when facing a Pavlov strategy, therefore the results can be different only due to the effect of the rewards. The same situation happens with R-max, when  $T = 4$ , it learns the Pavlov strategy and learns to cooperate, however when  $T = 5$  the selected action is to defect in all cases, this explains the change in opponent average from 2.398 to 0.846.

In the case against TFT-Pavlov, R-max learns the TFT strategy then it cooperates all the time. However, when the opponent changes to Pavlov, they change from C-C to C-D since R-max does not detect the switch. MDP4.5 learns

to cooperate with TFT and when the opponent changes to Pavlov, it detects the change and updates its model, thus stopping the C-D loop and changing to C-C (when  $T = 4$ ) or defecting (when  $T = 5$ ).

Another interesting case is against TFT-Bully. When facing a TFT, the result is the C-C joint action, but when the opponent changes to Bully, our approach detects this and therefore it changes to a defect action, resulting in the D-D joint action. In contrast, R-max does not adapt its model when the opponent changes to Bully. Thus, it keeps playing cooperate and it stuck in the C-D joint action, which result in 0 reward for the agent. As consequence MDP4.5 obtains higher average reward than the obtained by R-max.

When facing Bully-TFT the results are near 1 (for both agents) in all cases. This happens because R-max and MDP4.5 easily determine the Bully strategy and act accordingly with a defect action. When the opponent changes to TFT, the agents are stuck on D-D. This result in not detecting a switch and keep playing the D-D action. Something similar appears against TFT-TFTT. When the agents learn they are facing TFT they cooperate. When the opponent changes to TFTT, they do not notice this change and stay in the C-C joint action. These behaviors appears with both MDP4.5 and R-max.

When the first strategy is TFTT, MDP4.5 learns an incomplete model (since it only uses 1 step of memory) this results in a suboptimal strategy against the opponent.

In average MDP4.5 obtained better results than R-max. Since each game has 400 steps and the difference between average reward is 0.079 ( $T = 4$ ) and 0.126 ( $T = 5$ ) this results in a difference of 31 ( $T = 4$ ) and 50 ( $T = 5$ ) for a complete game.

## 6. CONCLUSIONS

We contribute to the state of the art by introducing a framework for fast learning changing non-stationary strategies. It uses decision trees to learn the most up to date opponent's strategy. The agent's learned model is continuously re-evaluated to assess strategy switches. We also contribute with a novel approach of transforming the learned decision tree and the game matrix into a MDP. Solving this MDP yields an optimal policy against that opponent. In order to evaluate whether the opponent changes its strategy, different trees are learned and compared through the game. We evaluated the both parts of the approach against state of the art algorithms. Our approach obtained the best results facing a non-deterministic opponent and it was able to efficiently detect strategy changes while learning a new strategy obtaining the best predictive scores. As future research we propose to perform a larger set of experiments on different domains.

## 7. ACKNOWLEDGMENTS

This work was partly supported by the National Council of Science and Technology of Mexico (CONACyT) scholarship grant 234507 to Pablo Hernandez-Leal .

## 8. REFERENCES

- [1] S. Barrett, P. Stone, S. Kraus, and A. Rosenfeld. Learning Teammate Models for Ad Hoc Teamwork. In *AAMAS Workshop Autonomous Learning Agents 2012*, Dec. 2012.

**Table 4: Average reward obtained for MDP4.5 and R-max against a switching opponent in 400 steps. Each row is the average of 100 games**

$C = 3, D = 1, S = 0, T = 4$								
	MDP4.5		R-max ( $m = 2, 3, 4$ )					
	Agent	Opponent	Agent	Opponent	Agent	Opponent	Agent	Opponent
Pavlov-TFT	<b>2.620</b>	2.577	2.446	2.398	2.496	2.486	2.455	2.437
Pavlov-TFTT	2.531	2.228	2.687	2.384	<b>2.688</b>	2.415	2.656	2.350
TFT-Pavlov	<b>2.646</b>	2.807	2.313	3.126	2.250	3.132	2.207	3.144
TFT-TFTT	2.865	2.922	<b>2.906</b>	2.901	2.886	2.923	2.871	2.922
TFTT-TFT	2.565	2.253	2.603	1.795	<b>2.624</b>	1.826	2.613	1.822
Bully-TFT	1.007	1.098	<b>1.037</b>	1.117	1.013	1.217	1.018	1.221
TFT-Bully	<b>1.868</b>	2.107	1.451	3.436	1.436	3.411	1.422	3.406
Pavlov-Bully	<b>1.721</b>	1.957	1.697	2.653	1.672	2.599	1.664	2.600
Bully-Pavlov	<b>1.744</b>	1.804	1.711	0.957	1.670	1.025	1.606	1.118
Average	<b>2.174</b>	2.195	2.095	2.307	2.082	2.337	2.057	2.335
$C = 3, D = 1, S = 0, T = 5$								
Pavlov-TFT	<b>2.797</b>	1.840	2.018	0.846	2.003	0.870	1.982	0.897
Pavlov-TFTT	<b>2.769</b>	1.208	2.042	0.843	2.034	0.876	2.053	0.944
TFT-Pavlov	<b>2.727</b>	2.474	2.289	3.330	2.281	3.344	2.277	3.354
TFT-TFTT	2.277	3.354	2.936	2.924	<b>2.937</b>	2.887	2.912	2.906
TFTT-TFT	2.641	2.114	<b>3.077</b>	2.039	3.069	2.029	3.068	2.043
Bully-TFT	1.018	1.138	1.074	1.214	<b>1.077</b>	1.305	1.057	1.455
TFT-Bully	<b>1.888</b>	2.184	1.473	3.876	1.455	3.883	1.433	3.908
Pavlov-Bully	1.935	1.071	<b>2.003</b>	0.825	1.969	0.962	1.955	1.002
Bully-Pavlov	1.916	1.109	<b>1.923</b>	1.011	1.915	1.090	1.784	1.366
Average	<b>2.219</b>	1.832	2.093	1.879	2.082	1.916	2.058	1.986

- [2] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.
- [3] R. I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.
- [4] G. W. Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.
- [5] L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, Mar. 2008.
- [6] M. Elidrisi, N. Johnson, and M. Gini. Fast Learning against Adaptive Adversarial Opponents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 1–8, Valencia, Spain, Nov. 2012.
- [7] D. Fudenberg and J. Tirole. *Game Theory*. The MIT Press, Aug. 1991.
- [8] S. Jensen, D. Boley, M. Gini, and P. Schrater. Rapid on-line temporal sequence prediction by an adaptive agent. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. ACM, July 2005.
- [9] M. L. Littman and P. Stone. Implicit Negotiation in Repeated Games. *ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, Aug. 2001.
- [10] R. Miglio and G. Soffritti. The comparison between classification trees through proximity measures. *Computational Statistics and Data Analysis*, 45(3):577–593, Apr. 2004.
- [11] M. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.
- [12] J. Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann, 1993.

# Qualitative Planning under Partial Observability in Multi-Agent Domains

Ronen I. Brafman  
Department of Computer  
Science  
Ben-Gurion University  
brafman@cs.bgu.ac.il

Guy Shani  
Information Systems  
Engineering  
Ben-Gurion University  
shanigu@bgu.ac.il

Shlomo Zilberstein  
Department of Computer  
Science  
University of Massachusetts  
shlomo@cs.umass.edu

## ABSTRACT

Decentralized POMDPs (Dec-POMDPs) provide a rich, attractive model for planning under uncertainty and partial observability in cooperative multi-agent domains with a growing body of research. In this paper we formulate a qualitative, propositional model for multi-agent planning under uncertainty with partial observability, which we call QDec-POMDP. We show that the worst-case complexity of planning in QDec-POMDPs is similar to that of Dec-POMDPs. Still, because the model is more “classical” in nature, it is more compact and easier to specify. Furthermore, it eases the adaptation of methods used in classical and contingent planning to solve problems to which Dec-POMDPs solution techniques cannot scale up. In particular, in this paper we describe a method based on compilation to classical planning, which handles multi-agent planning problems significantly larger than those handled by current Dec-POMDP algorithms.

## 1. INTRODUCTION

Many problems of practical importance call for the use of multiple autonomous agents that work together to achieve a common goal. For example, disaster response teams typically consist of multiple agents that have multiple tasks to perform, some of which require or can benefit from the cooperation of multiple agents. In such domains, agents typically have partial information, as they can sense their immediate surroundings only. And because agents are often located in different positions and may even have different sensing abilities, their runtime information states differ. Sometimes, this can be overcome using communication, but communication infrastructure can be damaged, and even if it exists, communication may be costly (in terms of time and resources) and should be reasoned about explicitly.

Decentralized POMDPs (Dec-POMDPs) offer a rich model for capturing such multi-agent planning problem [2002, 2008]. Dec-POMDPs extend the single agent POMDP model to account for multiple agents with possibly different information states. But the complexity of the Dec-POMDP model has limited its applicability. With a few exceptions [2008], little work has focused on exploiting factored models and no algorithms have been proposed for propositional STRIPS-style action models. In this paper we define and study a conceptually simpler model for multi-agent planning that extends the single-agent contingent planning model. We call this new model *Qualitative Dec-POMDP* (QDec-POMDP). We begin with defining a qualitative model that supports non-deterministic actions, but later restrict the model to have only deterministic ac-

tions.

In terms of worst-case complexity, we show that QDec-POMDPs are no easier than Dec-POMDPs. Nevertheless, a multi-agent contingent planning formalism offers two main advantages. First, being geared to propositional (a.k.a. factored) state models, it allows for more convenient model specification, as opposed to flat state models that characterize much of the work on Dec-POMDPs. Second, much like contingent planning, it is more amenable to the use of current classical planning methods, which are quite powerful. Thus, it could allow us to solve much larger problems. Indeed, one of our main contributions is a compilation method from QDec-POMDPs to classical planning, allowing us to tackle domains larger than those that can be solved by current Dec-POMDP algorithms.

Of course, the qualitative contingent planning model is less expressive in that it specifies the possible outcome states without their likelihood. But this is an advantage in cases where it is difficult to specify a richer, quantitative model, or when such models are too complex to solve. Furthermore, a solution to a qualitative model can provide guidance and heuristics for methods that operate on the quantitative model. Alternatively, one could use information from the quantitative model to bias choices made by a qualitative counterpart, e.g., when state sampling techniques are used, thus gradually moving from qualitative to quantitative.

In the next section we introduce the formal QDec-POMDP model. We start with an analysis of the complexity of solving a flat state-space qualitative model. This makes clear the impact of the move from a quantitative Dec-POMDP model, for which complexity results exist in that form. Next, we take a closer look at the issue of belief state representation, which is much more complex than in the single agent case. Here we still consider a flat state space model for semantic clarity. Next, we introduce a factored model, in the spirit of contingent planning models. Focusing on a deterministic variant of this model, we suggest an offline compilation method for its solution, and describe its empirical performance. The earlier discussion of belief states will help us understand an essential simplification made by this model. We end by discussing some of the challenges faced in designing an online algorithm.

## 2. MODEL DEFINITION

**DEFINITION 1.** *A qualitative decentralized partially observable Markov decision process (QDec-POMDP) is a tuple*

$$\mathcal{Q} = \langle I, S, b_0, \{A_i\}, \delta, \{\Omega_i\}, O, G, T \rangle$$

where

- $I$  is a finite set of agents indexed  $1, \dots, m$ .
- $S$  is a finite set of states.

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with AAMAS, May 2013, St. Paul, Minnesota, USA.

- $b_0 \subset S$  is the set of states initially possible.
- $A_i$  is a finite set of actions available to agent  $i$  and  $\vec{A} = \otimes_{i \in I} A_i$  is the set of joint actions, where  $\vec{a} = \langle a_1, \dots, a_m \rangle$  denotes a particular joint action.
- $\delta : S \times \vec{A} \rightarrow 2^S$  is a non-deterministic Markovian transition function.  $\delta(s, \vec{a})$  denotes the set of possible outcome states after taking joint action  $\vec{a}$  in state  $s$ .
- $\Omega_i$  is a finite set of observations available to agent  $i$  and  $\vec{\Omega} = \otimes_{i \in I} \Omega_i$  is the set of joint observation, where  $\vec{o} = \langle o_1, \dots, o_m \rangle$  denotes a particular joint observation.
- $\omega : \vec{A} \times S \rightarrow \vec{\Omega}$  is a deterministic observation function.  $\omega(\vec{a}, s)$  denotes the joint observation  $\vec{o}$  given that joint action  $\vec{a}$  was taken and led to outcome state  $s$ . Here  $s \in S$ ,  $\vec{a} \in \vec{A}$ ,  $\vec{o} \in \vec{\Omega}$ .
- $G \subset S$  is a set of goal states.
- $T$  is a positive integer representing the horizon

Note that all agents have a shared, initial belief state.

### 3. COMPLEXITY OF QDEC-POMDP

We represent the local plan of each agent using a *policy tree*  $q$ , which is a tree with branching factor  $|\Omega|$  and depth  $T$ . Each node of the tree is labeled with an action and each branch is labeled with an observation. To execute the plan, each agent performs the action at the root of the tree and then uses the subtree labeled with the observation it obtains for future action selection. If  $q_i$  is a policy tree for agent  $i$  and  $o_i$  is a possible observation for agent  $i$ , then  $q_{i, o_i}$  denotes the subtree that corresponds to the branch labeled by  $o_i$ .

Let  $\vec{q} = \langle q_1, q_2, \dots, q_m \rangle$  be a vector of policy trees.  $\vec{q}$  is also called a *joint policy*. We denote the joint action at the root of  $\vec{q}$  by  $\vec{a}_{\vec{q}}$ , and for an observation vector  $\vec{o} = o_1, \dots, o_m$ , we define  $\vec{q}_{\vec{o}} = \langle q_{1, o_1}, \dots, q_{m, o_m} \rangle$ .

Next, we characterize the set of states reachable via a joint policy  $\vec{q}$ . Intuitively, if all states reached by time  $T$  are goal states, the joint policy is a solution to the QDec-POMDP. To do so, we define the set of pairs of the form (global state, policy trees) that are reachable. The base case,  $\beta^0$ , corresponds to the initially possible state and the full depth- $T$  joint policy  $\vec{q}$ :  $\beta^0 = \{(s, \vec{q}) | s \in b_0\}$ . We define  $\beta^{t+1}$  inductively:

$$\beta^{t+1} = \{(s', \vec{q}_{\vec{o}}) | (s, \vec{q}) \in \beta^t, s' \in \delta(s, \vec{a}_{\vec{q}}), \vec{o} = \omega(\vec{a}_{\vec{q}}, s')\}$$

**DEFINITION 2.** A given depth- $T$  joint policy  $\vec{q}$  is a solution to a QDec-POMDP  $Q$  iff  $\forall s : (s, \emptyset) \in \beta^T \Rightarrow s \in G$ .

Note that at time  $T$  the remaining policy trees are empty.

**DEFINITION 3.** Let QDec-POMDP $_m$  denote the problem of finding a joint policy  $\vec{q}$  that is a solution of a given  $m$ -agent QDec-POMDP  $Q = \langle I, S, \{A_i\}, \delta, \{\Omega_i\}, O, G, T \rangle$  (i.e.,  $|I| = m$ ).

**THEOREM 1.** For all  $m \geq 2$ , if  $|T| \leq S$ , then QDec-POMDP $_m \in \text{NEXP}$ .

**PROOF.** We show that a nondeterministic machine can solve an instance of QDec-POMDP $_m$  using at most exponential time. To start, we guess a joint policy  $\vec{q}$ . A joint policy includes  $m$  policy trees, each of size  $O(|\Omega|^T)$ . Overall, the size is  $O(m|\Omega|^T)$ , and because  $T < |S|$ , the joint policy can be generated in exponential time. Given a joint policy, the update of the belief state  $\beta^t$  can be performed in exponential time:  $\beta^{t+1}$  can be larger than  $\beta^t$  by at

most a multiplicative factor of  $|S|$ , and the update takes polynomial time in the size  $\beta^{t+1}$ . Thus repeating this process  $T$  times may require at most exponential time. Finally, all we need is to verify that  $\forall s : (s, \emptyset) \in \beta^T \Rightarrow s \in G$ .  $\square$

**THEOREM 2.** For all  $m \geq 2$ , QDec-POMDP $_m$  is NEXP-Hard.

**PROOF.** The proof is similar to the one presented by [2002] for Dec-POMDPs. It follows a reduction of the TILING problem [1978, 1994], which is NEXP-complete, to the QDec-POMDP $_2$  problem. We only sketch the argument here.

TILING involves a given board size  $n$  (represented in binary), a set of tile types  $L = \{tile_0, \dots, tile_k\}$ , and a set of binary horizontal and vertical compatibility relations  $H, V \in L \times L$ . A tiling  $f$  is consistent iff (a)  $f(0, 0) = tile_0$ , and (b) for all  $x, y$   $\langle f(x, y), f(x + 1, y) \rangle \in H$  and  $\langle f(x, y), f(x, y + 1) \rangle \in V$ . That is, adjacent tiles satisfy the compatibility relations. The decision problem is to determine, given  $n, L, H, V$ , whether a consistent tiling exists.

The basic idea is to create a two-agent QDec-POMDP that randomly selects two tiling locations bit by bit, informing one agent of the first location and the other agent of the second location. The agents' local policies are observation-history based, so the agents can base their future actions on the tiling locations given to them. After generating the locations, the agents are simultaneously queried to place tiles at some locations. The QDec-POMDP problem is designed such that the agents reach the goal iff their answers to the query are based on some agreed upon solution of the tiling problem. Here is a brief discussion of the phases of the original proof from [2002] and the relevant changes needed for the QDec-POMDP model.

**Select Phase** Using nondeterminism, the system generates two random bit positions and values. They are memorized as part of the state and not observed by the agents.

**Generate Phase** Using nondeterminism, the system generates two tile locations and reveals one to each agent via their observation streams.

**Query Phase** Each agent is queried for a tile type to place in the location specified to it.

**Echo Phase** The agents are now required to echo their tile locations. Only one location (not known to the agents) is verified by the system per observation stream. Making an error in the echo phase leads to a dead-end, from which the goal cannot be reached. During the echo phase, the system tracks the adjacency relationship between the tile locations.

**Test Phase** At this point the system checks whether the tile types provided in the query phase come from a single consistent tiling. If the tile types violate any of the TILING problem constraints, a dead-end state is reached. Otherwise, the goal is reached.

Similar to the original proof, if there exists a consistent tiling, then there must exist a joint policy for the constructed QDec-POMDP $_2$  that reaches the goal state. Likewise, there is no way to *guarantee* goal reachability without the agents being faithful to a single consistent tiling.  $\square$

Note that the QDec-POMDP constructed for the proof is in fact a QDec-MDP (i.e., the observations of the two agents combined provide full information on the state of the system). Therefore, QDec-MDP $_2$  is NEXP-Hard as well.

**COROLLARY 1.** For all  $m \geq 2$ , both QDec-POMDP $_m$  and QDec-MDP $_m$  are NEXP-complete.

It is somewhat surprising that the qualitative model with its different objective (goal reachability versus maximizing expected reward) has the same complexity as the standard Dec-POMDP model. In some sense, this confirms the intuition that the main source of complexity is decentralized operation with partial information, not stochastic actions.

## 4. BELIEF STATES AND JOINT POLICIES

The notion of the agent’s belief state plays a central role in algorithms for solving problems with partial observability. In this section, we explore types of belief states that could be used in QDec-POMDPs and then describe an alternative representation for a joint policy.

### 4.1 Online Local Belief States

We begin with a definition of a local belief state of an agent in the context of a known joint-policy tree. This definition is useful for reasoning about the information state of an agent online. However, it cannot be used for the generation of a joint-policy, as our definition assumes a given policy.

Each agent can maintain a belief state  $\beta_i^t$  at time  $t$  that reflects its own experience. The belief state includes all the possible pairs of *system state* and *policy trees for all the agents*. Agent  $i$  knows its own policy tree, so all the vectors of policy trees included in its belief state must agree on its own policy tree being the actual one.

The initial belief state of agent  $i$  is  $\beta_i^0 = \{(s_0, \vec{q}) | s_0 \in b_0\}$  where  $\vec{q}$  is the initial vector of policy trees for all the agents. Let  $a_i^t$  be the action agent  $i$  executes at time  $t$ , and  $o_i^t$  is the observation it obtains. We define  $\beta^t$  inductively as follows:

$$\begin{aligned} \beta_i^t = \{ & (s_t, \vec{q}) \mid (s_{t-1}, \vec{q}) \in \beta_i^{t-1}, \\ & s_t \in \delta(s_{t-1}, \vec{a}_t), \\ & \vec{o} = \omega(\vec{a}_t, s_t), \vec{o}[i] = o_i^t \} \end{aligned} \quad (1)$$

The only difference between the global update of  $\beta^t$  and the local update is the added condition  $\vec{o}[i] = o_i^t$ , which means that we only include outcome states  $s_t$  that produce the actual observation that agent  $i$  obtained. That is, we use the local information of agent  $i$  to filter states that are inconsistent with its knowledge.

This belief state update scheme is valid when the joint policy is fixed in advance in the form of policy trees. But if we want to have policies that depend on these local belief states we run into a problem. The actions of the other agents depend on their beliefs that in turn depend on their actions. Without resolving this circularity, it is hard to generate plans conditioned on local beliefs.

### 4.2 Offline, Policy Independent Belief States

Most existing methods for planning under partial observability rely on a “nice-to-manage”, policy-independent notion of belief state. These methods include, for example, search in belief state space, the computation of a real-valued function over belief states, as in POMDPs, and the generation of a policy that maps belief states to actions.

In the multi-agent case there is no longer a single belief state, but we can replace that with the notion of a history. A *history* is a sequence of states and actions, of the form  $(s_0, a_1, s_1, \dots, a_n, s_n)$ , denoting the initial state, the initial action, the resulting state, etc. If  $h = (s_0, a_1, s_1, \dots, a_n, s_n)$ , let  $h_s(k) = s_k$  and  $h_a(k) = a_k$ . Initially, every agent’s belief state is  $\beta_i^0 = \{(s_0) | s_0 \in b_0\}$ . We define  $\beta_{\beta^{t-1}, a_i, o_i}^t$  the new belief state of agent  $i$  at time  $t$  after executing  $a_i$  and observing  $o_i$  in belief state  $\beta^{t-1}$ , as follows:

$$\begin{aligned} \beta_{\beta^{t-1}, a_i, o_i}^t = \{ & (h \circ (\vec{a}_t, s_t)) \mid h \in \beta^{t-1}, \\ & s_t \in \delta(h_s(t-1), \vec{a}_t), \vec{a}_t[i] = a_i, \\ & \vec{o} = \omega(\vec{a}_t, s_t), \vec{o}[i] = o_i \} \end{aligned} \quad (2)$$

That is, those histories that extend current histories with a joint action that is consistent with the joint action executed by the agent, and with a state which is the result of applying that joint action to the last state of the history, and that the last state and action would induce a joint-observation consistent with agent’s local observation.

Unfortunately, to the best of our knowledge, it is not possible to compile the history based representation to a state-based representation. The reason is that different histories that led one agent to the same state, lead to different states of information for other agents. Consequently, the set of current last states is not a sufficient statistics, but only an approximation, which we will employ later on to obtain a planning algorithm, and refer to as the set-of-possible-states approximation.

Initially, every agent’s belief state is  $\beta_i^0 = \{(s_0) | s_0 \in b_0\}$ . The estimated set of possible states for agent  $i$  at time  $t$  given the estimated belief state at time  $t-1$ , action  $a_i$  by the agent, and observation  $o_i$  is defined as follows:

$$\begin{aligned} \beta_{\beta^{t-1}, a_i, o_i}^t = \{ & s_t : s_{t-1} \in \beta^{t-1}, \\ & s_t \in \delta(s_{t-1}, \vec{a}_t), \vec{a}_t[i] = a_i, \\ & \vec{o} = \omega(\vec{a}_t, s_t), \vec{o}[i] = o_i \} \end{aligned} \quad (3)$$

### 4.3 Global Policy Tree

To describe a joint policy, we used a vector  $\vec{q}$  of individual policy trees. An alternative description is a global policy tree, which we denote by  $q_g$ . Its definition is identical to that of an individual policy tree, except that nodes are labeled by *joint* actions, and edges are labeled by *joint* observations.

Unfortunately, some general policy trees do not correspond to any joint policy. If two nodes in the global policy tree correspond to branches that would yield the same history for agent  $i$ , i.e., agent  $i$  cannot distinguish between these branches, the action assigned to  $i$  in these nodes must be identical.

Thus, let  $q_g$  be a policy tree, and let  $b_0$  be the initial belief state. For every node  $n$ , let  $\vec{b}(n) = b_1(n), \dots, b_m(n)$  be the vector of agents’ belief states given the history that corresponds to the path to this node.  $q_g$  is *executable* if for every agent  $i = 1, \dots, m$  and every two nodes  $n, n'$  in  $q_g$ , if  $b_i(n) = b_i(n')$  then the  $i^{th}$  component of the joint actions associated with  $n$  and  $n'$  must be identical.

Although joint policies are easier to execute — they contain an explicit policy for each agent — global policy trees are a better fit for the compilation approach that we describe below, because they are closer in form to (single-agent) classical plans over joint actions: Instead of generating joint policy trees consisting of  $m$  local policies, our translation method will seek a single *executable* global policy tree. To ensure that the global tree is executable, we will enforce the constraint described above while using the set-of-possible-states approximation for agents’ state of knowledge. Because this approximation is sound, i.e., two histories that the agent cannot distinguish with will always yield two identical sets of possible states (but not vice versa), we are guaranteed that the global policy tree is indeed executable.

## 5. FACTORED REPRESENTATION OF QDEC-POMDP

We now describe a factored specification of a QDecPOMDP model, motivated by the classical STRIPS and PDDL languages. We will slightly abuse notation, overloading terms defined in previous sections.

DEFINITION 4. A factored QDec-POMDP is a tuple

$$\langle I, P, \vec{A}, Pre, Eff, Obs, b_0, G \rangle$$

where  $I$  is a set of agents,  $P$  is a set of propositions,  $\vec{A}$  is a vector of individual action sets,  $Pre$  is the precondition function,  $Eff$  is the effects function,  $b_0$  is the set of initially possible states, and  $G$  is a set (conjunction) of goal propositions. The state space  $S$  consists of all truth assignments to  $P$ , and each state can be viewed as a set of literals.

The precondition function  $Pre$  maps each individual action  $a_i \in A_i$  to its set of preconditions, i.e., a set of literals that must hold whenever agent  $i$  executes  $a_i$ . Preconditions are local, i.e., defined over  $a_i$  rather than  $\vec{a}$ , because each agent must ensure that the relevant preconditions hold prior to executing its part of the joint action. To execute an action when the agent does not know the current state, the agent must ensure that the preconditions hold for all the states in its current belief. We extend  $Pre$  to be defined over joint actions  $\{\vec{a} = \langle a_1, \dots, a_m \rangle : a_i \in A_i\}$  (where  $m = |I|$ ):  $Pre(\langle a_1, \dots, a_m \rangle) = \cup_i Pre(a_i)$ .

The effects function  $Eff$  maps joint actions into a set of pairs  $(c, e)$  of conditional effects, where  $c$  is a conjunction of literals and  $e$  is a single literal, such that if  $c$  holds before the execution of the action  $e$  holds after its execution. Thus, effects are a function of the *joint* action rather than of the local actions, as can be expected, due to possible interactions between local actions. For the sake of semantic clarity, we assume that if  $(c, e)$  and  $(c', e')$  are conditional effects of the same joint action, then  $c$  and  $c'$  are inconsistent. Here we focus on deterministic effects, but one can model non-deterministic effects simply by allowing for multiple pairs of the form  $(c, e), (c, e')$  representing alternative outcomes of the action under the same conditions. The preconditions and effects functions, taken together, define the transition function from one state to another given actions.

For every joint action  $\vec{a}$  and agent  $i$ ,  $Obs(\vec{a}, i) = \{p_1, \dots, p_k\}$ , where  $p_1, \dots, p_k$  are the propositions whose value agent  $i$  observes after the joint execution of  $\vec{a}$ . The observation is private – i.e., each agent may observe different aspects of the world, and we assume that the observed value is correct and corresponds to the post-action value of these variables.

A solution to the factored model is identical to that used for the flat model. We can use joint policy trees or executable global policy trees, as discussed earlier.

EXAMPLE 1. We now illustrate the factored QDec-POMDP model using a simple box pushing domain. In this example there is one dimensional grid of size 3, with cells marked 1-3, and two agents, starting in cells 1 and 3. In each cell there may be a box, which needs to be pushed upwards. The left and right boxes are light, and a single agent may push them alone. The middle box is heavy, and requires that the two agents push it together.

We can hence define  $I = \{1, 2\}$  and  $P = \{AgentAt_{i,pos}, BoxAt_{j,pos}\}$  where  $pos \in \{1, 2, 3\}$  is a possible position in the grid,  $i \in \{1, 2\}$  is the agent index, and  $j \in \{1, 2, 3\}$  is a box index. In the initial state each box may or may not be in its corresponding cell —  $b_0 = AgentAt_{1,1} \wedge AgentAt_{2,3} \wedge (BoxAt_{1,j} \vee \neg BoxAt_{1,j})$  for  $j = 1, 2, 3$ . There are therefore 6 possible initial states.

The allowed actions for the agents are to move left and right, and to push a give box up. There are no preconditions for moving left and right, i.e.  $Pre(Left) = Pre(Right) = \phi$ . To push up box  $j$ , agent  $i$  must be in the same place as the box. That is,  $Pre(PushUp_{i,j}) = \{AgentAt_{i,j}, BoxAt_{j,j}\}$ . The moving actions transition the agent from one position to the other, and are independent of the effects of other agent actions, e.g.,  $Right_i = \{(AgentAt_{i,1}, \neg AgentAt_{i,1} \wedge AgentAt_{i,2}), (AgentAt_{i,2}, \neg AgentAt_{i,2} \wedge AgentAt_{i,3})\}$ .

The only truly joint effect is for the actions that contain a component  $PushUp_{i,2}$ , where box 2 is the heavy box —  $Eff(PushUp_{1,2}, a_2)$  where  $a_2$  is some other action, are identical to the independent effects of action  $a_2$ , while

$Eff(PushUp_{1,2}, PushUp_{2,2}) = \{(\phi, \neg BoxAt_{2,2})\}$ , that is, if and only if the two agents push the heavy box jointly, it (unconditionally) gets moved out of the grid.

We define sensing actions for boxes —  $SenseBox_{i,j}$ , with precondition  $Pre(SenseBox_{i,j}) = AgentAt_{i,j}$ , no effects, and  $Obs(SenseBox_{i,j}) = BoxAt_{j,j}$ . The goal is to move all boxes out of the grid, i.e.,  $\bigwedge_j \neg BoxAt_{j,j}$ .

## 6. COMPILATION-BASED METHOD

We now present a method for solving QDec-POMDP problems using a compilation approach to classical planning. Our approach generates a planning problem whose solution corresponds to an executable global plan tree, branching on agent observations. It relies on the approximate, sound, but incomplete notion of belief state, as discussed earlier.

The compilation method is currently designed for deterministic QDec-POMDPs, i.e., ones where actions have deterministic effects. The method could in principal be expanded to handle non-determinism by embedding the uncertainty of action effects into the uncertainty of the initial belief [2008], but this will clearly impact the solution time and size. We hence leave discussion of efficient handling of non-deterministic effects to future research.

A classical planning problem is a tuple  $\pi = \langle P, A, s_0, G \rangle$  where  $P$  is a set of propositions,  $A$  is a set of actions,  $s_0$  is the initial state, and  $G$  is a set of goal propositions. We use a translation method inspired by the MPRS translation method [2012]. An important concept in this translation is *distinguishability* between states. We say that we can distinguish at runtime between two states  $s, s'$ , denoted  $\neg s/s'$ , if we observed the value of some proposition  $p$  which is true in  $s$  and false in  $s'$ . In our translation we have two types of distinguishability — when a single agent can distinguish between states based on its own observations, denoted  $\neg s/s' | i$ , and when the combined observations of the agents can distinguish between states, denoted  $\neg s/s'$ , as in MPRS.

Given a factored QDec-POMDP problem

$$\pi = \langle I, P, \vec{A} = \{A_i\}, Pre, Eff, Obs, b_0, G \rangle$$

defined as in the previous section we create the following classical planning problem  $\pi_c = \langle P_c, A_c, s_{0,c}, G_c \rangle$ :

**Propositions**  $P_c = \{p/s : p \in P, s \in S\} \cup \{\neg s/s' : s, s' \in S\} \cup \{\neg s/s' | i : s, s' \in S, i \in I\}$ . Propositions of the form  $p/s$  capture the value at run time of  $p$  when  $s$  is the true initial state. Propositions of the form  $\neg s'/s | i$  denote that at run-time, if  $s$  is the true initial state, then agent  $i$  has gathered sufficient data to conclude that  $s'$  cannot be the true initial state, i.e., to distinguish between state  $s$  and  $s'$ . These propositions allow us to define the agent-specific belief state during execution. These will be used later to enforce the constraint on actions at the same level explained in the previous section. Propositions of the form  $\neg s'/s$  allow us to

distinguish between states that at least one of the agents can distinguish between. These propositions allow us to define the joint belief state during plan construction.

**Actions** For every joint action  $\vec{a}$  and every subset of  $S' \subseteq b_0$ ,  $A_c$  contains an action  $a_{S'}$ . This action denotes the execution of  $\vec{a}$  when the set of possible states is  $S'$ .  $a_{S'}$  has no effect on states outside  $S'$ . It is defined as follows:

$pre(\vec{a}_{S'}) = \{p/s : s \in S', p \in pre(\vec{a})\} \cup \{\neg s'/s : s' \in S', s \in b_0 \setminus S'\}$ . That is, the preconditions must hold prior to applying the action in all states for which this action applies, and the joint knowledge of the agents must be sufficient to distinguish between any state in  $S'$  and every state *not* in  $S'$ . Thus, the plan can choose action  $a_{S'}$  only when the current belief state is  $S'$ , and all action preconditions are known to hold in  $S'$ .

For every  $(c, e) \in effects(a)$ ,  $effects(a_{S'})$  contains the following conditional effects:

- For each  $s \in S'$ ,  $(c/s, e/s)$  — the effect applied to every state in  $S'$ .
- $\{(p/s \wedge \neg p/s', \neg s/s' | i)\}$  — for every  $p$  observable by agent  $i$ , and every two states  $s, s' \in S'$ , if the states disagree on  $p$ , then agent  $i$  can distinguish between the states following the observation of the value of  $p$ .

**Initial State**  $s_{0c} = \bigwedge_{s \in b_0, s \models l} l/s$  — for every literal we specify its value in all possible initial states.

**Goal**  $G_c = \{\bigwedge_{s \in b_0} G/s\}$  — we require that the goal will be achieved in all states.

In addition we must explicitly enforce the constraints on nodes at the same depth or level, as explained in the previous section. To avoid the dependency on the depth, which is a numeric variable, unsupported by the planners that we use, we enforce the plan construction to proceed in a breadth-first-search (BFS). That is, each level in the tree must be fully constructed before the next level can be started. To achieve that we add for each state  $s$  in the initial belief a proposition  $LevelDone_s$ . For each compiled action  $\vec{a}_{S'}$  we add preconditions  $\bigwedge_{s \in S'} \neg LevelDone_s$ , and unconditional effects  $\bigwedge_{s \in S'} LevelDone_s$ . Thus, once a state has been handled at the current level of the tree, no action that applies to it can be executed at the current level. To move to the next level, we add an action  $ResetLevel$  with preconditions  $\bigwedge_{s \in b_0} LevelDone_s$  and unconditional effects  $\bigwedge_{s \in b_0} \neg LevelDone_s$ . That is, once all states have been taken care for the current level, the  $LevelDone$  propositions are reset and the next level begins.

After enforcing a BSF plan construction, we enforce that all agent actions at the current level in different states can be different only if the agent can distinguish between the states. As the ability to distinguish between states is a result of a different observation, this achieves the validity constraint required for global policy trees to become executable, as discussed in the previous section. We add for each agent  $i$  and action  $a_i \in \{A_i\}$  predicates  $constraint_{a_i, s}$ , modeling which states are constrained on  $a_i$ . For every action  $\vec{a}_{S'}$  we add preconditions:

$$\bigwedge_{i \in I, s \notin S'} \neg LevelDone_s \wedge (constraint_{a_i, s} \vee (\bigwedge_{s' \in S'} \neg s/s' | i))$$

where  $a_i$  is the action assigned to agent  $i$  in  $\vec{a}_{S'}$ . That is, for each agent  $i$  and state  $s$  which is not handled by the action, either  $s$  has not yet been handled by any other action, and is hence unconstrained, or there is a constraint of  $s$  and it matches  $a_i$ , or we can distinguish between  $s$  and any other state  $s' \in S'$  for which the action does apply. We also add unconditional effects  $\bigwedge_{i \in I, s \in S'} constraint_{a_i, s}$ , specifying the new constraint induced

**Table 1: Execution time (seconds) for different box pushing domains, comparing our translation-based QDec-POMDP approach, and two Dec-POMDP solvers, IPG and GMAA-ICE with the  $Q_{MDP}$  heuristic. A model is defined by its width (W), length (L), and number of boxes (B). Average depth denotes the average depth of leaves in the policy tree. Expected cost was reported by the GMAA-ICE solver.**

Domain W, L, B	S	b <sub>0</sub>	QDec- POMDP	Avg depth	IPG IPG	GMAA- ICE	Expected cost
2, 2, 2	256	4	12.79	2	450	15.32	2
2, 3, 2	1296	4	25.39	2	×	59.67	2
2, 3, 3	7776	8	48.42	5	×	732.59	5
3, 3, 3	59049	8	66.47	6	×	×	×

when selecting the joint action. When a level is done, we remove all constraints in the  $ResetLevel$  action, i.e., we add to  $ResetLevel$  unconditional effects  $\bigwedge_{i \in I, a_i \in A_i, s \in b_0} \neg constraint_{a_i, s}$ .

The solution to the classical problem above is a linearization of the joint plan tree [2012].

**EXAMPLE 2.** We now describe a portion of the compilation of the box pushing domain described in Example 1. The set of possible initial state can be described as  $s_{b_1 b_2 b_3}$  where  $b_i$  denotes whether  $b_i$  is initially in the grid and must be pushed up. For example,  $s_{tft}$  denotes that box 1 and 3 are in the grid, and box 2 is not. The propositions are conditioned on the initial states, and we thus have, e.g.,  $BoxAt_{j, pos}/s_{tft}$ , and  $AgentAt_{i, pos}/s_{tft}$ .

For each subset of states we define one instance of an action. For example, for  $S' = \{s_{ttt}, s_{fff}\}$ , and action  $Left_i$  we will define an action  $Left_{i, S'}$  with preconditions  $\bigwedge_{s \in S'} \neg s_{ttt}/s \wedge \neg s_{fff}/s$ . We also need to ensure the BFS expansion, by adding  $\neg LevelDone_{s_{ttt}} \wedge \neg LevelDone_{s_{fff}}$ . Finally, we ensure that the proper execution tree structure holds by adding

$$\bigwedge_{s \notin \{s_{ttt}, s_{fff}\}} constraint_{Left_{i, S'}, s} \vee (\neg s_{ttt}/s' | i \wedge \neg s_{fff}/s' | i)$$

The effects of the action are specified only for  $s_{ttt}$  and  $s_{fff}$ :  
 $(AgentAt_{i, 3}/s_{ttt}, \neg AgentAt_{i, 3}/s_{ttt} \wedge AgentAt_{i, 3}/s_{ttt})$ ,  
 $(AgentAt_{i, 3}/s_{fff}, \neg AgentAt_{i, 3}/s_{fff} \wedge AgentAt_{i, 3}/s_{fff})$ .

In addition, we add to the effects  $LevelDone_{s_{ttt}} \wedge LevelDone_{s_{fff}}$  so that these states will not be handled again at the current depth. Next, we add the resulting constraint effect  $constraint_{Left_{i, S'}, s_{ttt}} \wedge constraint_{Left_{i, S'}, s_{fff}}$  ensuring that all states undistinguishable from  $\{s_{ttt}, s_{fff}\}$  must also use  $Left_i$  at the current tree depth.

## 7. EXPERIMENTAL RESULTS

We now provide some proof-of-concept experimental results showing that our algorithm can solve considerable size QDec-POMDP problems. We experiment with a variant of the box pushing problem [2007] where a set of boxes are spread in a grid, and the agents must push each box to a designated location at the edge of the grid (the end of the column it appears in). Each box may be either in a pre-specified location, or at its goal location to begin with, and the agent must be in the same location as the box in order to observe where it is. Agents may move in the 4 primary directions, and can push boxes in these 4 primary directions, if they occupy the same location as the box. Some boxes are heavy and must be pushed by a few agents jointly (in our example, heavy boxes are pushed by 2 agents). Agents can also only observe the location of other agents when they are in the same location. All transitions and observations are deterministic.

We experimented with four box pushing domains. The smallest example that we tried was a  $2 \times 2$  grid, with 2 boxes and 2 agents and the largest had a  $3 \times 3$  grid with 3 boxes. Each  $A_i$  has 11 possible actions (4 move actions, 4 push actions, observing the other agent, and observing each box), and hence there are 121 joint actions. We ran two Dec-POMDP solvers on this fully deterministic Dec-POMDP problem — the GMAA-ICE algorithm with the  $Q_{MDP}$  search heuristic [2008] using the MADP package<sup>1</sup>, and Incremental Policy Generation (IPG) [2009]. The results are presented in Table 1. Our compilation approach solves all the problems using the Fast Downward (FD) classical planner [2006], while IPG solves only the smallest instance, and GMAA-ICE solves the smaller instances but not the larger one. Manually observing the trees, we saw that the planner computed the intuitive plan tree.

We acknowledge that this comparison is not entirely fair, because Dec-POMDP solvers try to optimize the solution, whereas we only seek a satisfying solution. Thus, Dec-POMDP solvers may need to explore many more branches of the search graph, at a much greater computational cost. That being said, our experiments clearly demonstrate that our approach can provide solutions to decentralized problems beyond the reach of current Dec-POMDP solvers.

Another interesting aspect of our approach is the ability to compactly represent huge problems. For example, the  $3 \times 3$  box pushing example that we describe above, required a model size of over 1GB (specifying only non-zero probabilities) in the traditional Cassandra format for Dec-POMDPs, while our factored representation required less than 15KB.

## 8. CONCLUSION

We presented a new model for multi-agent planning problems, called QDec-POMDP, which emphasizes valid, rather than optimal solutions, that achieve a given goal, in the spirit of classical and contingent planning. We analyzed the complexity of the new model, concluding that it is as hard as the standard Dec-POMDP model for a given horizon. Then, we presented a factored version of this model, motivated by similar representations used in classical and contingent planning. Our representation is compact and can describe models with tens of thousands of states and about 150 joint actions using file sizes of less than 15KB, although we will investigate more compact specification of the effects of joint action. Next, we described a solution method for deterministic QDec-POMDPs, based on a compilation approach to classical planning. Our method creates a classical planning problem whose solution is a linearized joint plan tree. We demonstrated the advantage of this compilation method over Dec-POMDP solvers using a number of examples. Our approach solves small problems much faster and scales to larger problems compared to existing Dec-POMDP solvers.

In this paper, our focus was on providing an exposition of the model, its properties, and potential. Of course, this is only the first step towards developing more scalable solvers for QDec-POMDP domains. In particular, we know well from contingent planning that it is much harder to scale up offline solution methods. Hence, we intend to explore online planning in QDec-POMDPs. This raises some non-trivial challenges as we will need some mechanism that will allow different agents with different belief states to jointly plan [2011], unlike the offline case in which a global plan is generated for a group of agents that share an initial belief state. The advantage, however, is that agents can focus on the relevant part of the state space at each planning phase, requiring smaller encodings

and smaller plans. In addition, online methods are likely to better deal with non-deterministic effects. A second possible direction for scaling up would allow agents to plan independently, enforcing certain constraints on the joint solution.

## 9. REFERENCES

- Amato, C.; Dibangoye, J. S.; and Zilberstein, S. 2009. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2–9.
- Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27:819–840.
- Brafman, R. I., and Shani, G. 2012. A multi-path compilation approach to contingent planning. In *Proceedings of the 26th Conference on Artificial Intelligence*, 1868–1874.
- Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.
- Lewis, H. R. 1978. Complexity of solvable cases of the decision problem for the predicate calculus. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, 35–47.
- Oliehoek, F. A.; Spaan, M. T. J.; Whiteson, S.; and Vlassis, N. 2008. Exploiting locality of interaction in factored DEC-POMDPs. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 517–524.
- Oliehoek, F. A.; Spaan, M. T. J.; and Vlassis, N. A. 2008. Optimal and approximate q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research* 32:289–353.
- Papadimitriou, C. H. 1994. *Computational Complexity*. Reading, MA: Addison-Wesley.
- Seuken, S., and Zilberstein, S. 2007. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 344–351.
- Seuken, S., and Zilberstein, S. 2008. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems* 17(2):190–250.
- Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175(2):487–511.
- Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd Conference on Artificial Intelligence*, 1010–1016.

<sup>1</sup>staff.science.uva.nl/~faolieho/madp



# Solving Dec-POMDPs by Genetic Algorithms: Robot Soccer Case Study

Okan Aşık  
Boğaziçi University,  
Department of Computer Engineering  
34342, Bebek, İstanbul, Turkey  
okan.asik@boun.edu.tr

H. Levent Akın  
Boğaziçi University,  
Department of Computer Engineering  
34342, Bebek, İstanbul, Turkey  
akin@boun.edu.tr

## ABSTRACT

Decentralized Partially Observable Markov Decision Process (Dec-POMDP) is one of the well defined formalizations for studying the multi-agent decision making problem. However, it lacks real-world practical applications. In this work, we model multi-agent decision making in 2D robot soccer as Dec-POMDP and solve it by using genetic algorithms developed by Eker and Akın [6]. We define both a discrete Dec-POMDP model and a continuous Dec-POMDP model. Finite state controller policy representation is used for the discrete Dec-POMDP model and neural network is used for the continuous Dec-POMDP model. The model has four agents and every agent has different policies. The algorithm searches the joint policy space with genetic algorithms. To calculate the fitness of policies we use the *TeamBots* simulation environment. The fitness of the policies are calculated as the score difference. To have good fitness estimates, we need to run many trials. We show that we can model decision making in 2D robot soccer as Dec-POMDP and have satisfactory results with an approximate Dec-POMDP algorithm. The trained policies win most of the games against the standard *TeamBots* teams and against teams trained with reinforcement learning method.

## Keywords

Dec-POMDP, genetic algorithms, robot soccer, simulation, high-level planning

## 1. INTRODUCTION

Robots are embodied entities which achieve their designated goals using their sensors and actuators. The sensors provide information about the external world and also about the robot itself. Different sensors have different capabilities and limitations. For example, an RGB camera can only provide visual information at a certain view angle, but a laser scanner provides depth information in 1D for a certain view angle. In addition to the limited sensor modalities, the information provided by the sensors is noisy. Therefore, robots should use robust methods to be able to deal with such sensor limitations.

A robot acts on its environment via its actuators. Different actuators have different capabilities and limitations. For example, while wheels are most appropriate for indoor navigation, legged navigation is better for rugged terrain. Also, each actuator type has its own limitations and uncertainties. For wheeled navigation, while we expect a robot to travel for certain amount of distance, because

of slippage, it may not be able to travel the desired distance. In order to achieve certain task performance, we should use methods which deal with uncertainties in the actuators.

A robot develops an understanding of the world via its sensors and acts on the world by its actuators. However, there is a need for a method to produce certain actuator commands according to the sensor inputs. Decision making is the central method which maps sensor inputs to actuator commands. Therefore, the success of the robot is directly related to the success of decision making. However, the success of decision making is dependent on many factors such as the complexity of decision making method, and the formulation of the problem.

Markov Decision Process (MDP) is a formalism which is used to model the uncertainties in actions. Partially Observable Markov Processes (POMDP) is an extension of MDP to allow partial observability and uncertainty in observations. To represent robot decision making as POMDP, sensor inputs can be modeled as observations and primitive behaviors as actions. There are two limitations of such models. Firstly, their success is highly dependent on the detail of the model. Secondly, for certain problems, it requires detailed knowledge of the dynamics of the process which may not be available for every problem. Also, as the detail of models increases the problem complexity increases.

Some tasks, such as foraging, coverage, search, cooperative manipulation, and soccer, require the cooperation of robots. In addition to the problems a single robot has to solve, it has to achieve coordination with other robots. At each time step, each robot takes an action. The environment changes its state based on those actions. The robots get reward according to the joint action and the current environment state. Finally, the robots get new observations from the environment based on the current state of the environment. Decision making algorithms are expected to find the optimal action which results in the maximization of the reward. Decentralized Partially Observable Markov Process (Dec-POMDP) is an extension of POMDP to the multi-robot processes. However, the optimal solution to Dec-POMDP is intractable for real-world problems. Therefore, most of the research efforts are focused on approximate solution methods.

In this work, we model 2D robot soccer as Dec-POMDP and solve it by using genetic algorithms based approach developed by Eker and Akın [6, 5]. We use the finite state controller policy representation for the discrete robot soccer Dec-POMDP model, and neural network representation for the continuous robot soccer Dec-POMDP model. The genetic algorithm searches the joint policy space to find a good policy. The fitness of the policies are calculated as the score difference by using the *TeamBots* simulation environment. The trained policies win most of the games against the standard *TeamBots* teams and against teams trained by reinforce-

---

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with AAMAS, May 2013, St. Paul, Minnesota, USA.

ment learning method [12]. The main contribution of this work is that we can solve large-scale real world Dec-POMDP problems.

The organization of the rest of the paper is as follows. Section 2 introduces related work. Section 3 presents the background and the algorithm we used to solve Dec-POMDP problem. Section 4 introduces our experiments and results. We present our conclusions and intended future work in Section 5.

## 2. RELATED WORK

As it has been shown by Bernstein and *et al.* [3], optimal solution to Dec-POMDP is NEXP-complete. Therefore, optimal algorithms can only solve toy problems and recent research focus on the development of approximate algorithms. Dynamic programming approach is one of the widely used methods in approximate Dec-POMDP algorithms [17]. A heuristic identifies reachable belief states and the most probable observations, then a dynamic algorithm finds the appropriate joint policies [15]. Another widely used approach is policy iteration [17]. Policies are represented as finite state controllers and are iteratively improved by optimization methods such as linear programming [4]. One of the recent approaches is genetic algorithms [5, 6]. Since the algorithm represents policies as finite state controllers, it is similar to the policy iteration approach, but it searches the joint policy space with genetic algorithms.

In the literature, there are very limited applications of Dec-POMDPs. Sensor networks [10] and robot teams [16, 7, 22] are some of the application domains of Dec-POMDP.

Robot soccer is one of the good testbeds for studying Dec-POMDP algorithms. Wu and Chen [22] propose the correlation device to achieve the coordination between individual agents. The correlation device is represented as MDP and operates independent of the Dec-POMDP process. Dynamic programming is used to calculate the correlation device. They compare their results with the memory-bounded dynamic programming algorithm [4]. They report improvement in running time while maintaining the same performance. However, since they implement their algorithm to improve a primitive defense behavior, we cannot directly attribute the success of the team to the algorithm.

Since robot soccer is complex problem, Keepaway soccer has been proposed as a sub-problem of robot soccer to be used as a machine learning benchmark. Keepaway problem is modeled as Semi-Markov Decision Process (SMDP) and various reinforcement learning methods have been studied [19, 20, 21, 18]. Di Pietro and *et al.* [14] uses evolutionary algorithms to learn a policy which achieves the coordination of the Keepaway players. The policy is a decision tree formulation of local observations such as distance to the ball and distance to the closest opponent. The evolutionary algorithm searches for the optimal decision tree to keep the ball as long as possible which is the goal of the keeper team.

Most of the work which studies multi-agent coordination generally solves only a subset of the soccer problem. Therefore, we cannot identify the success of the algorithm. In this work, we propose the full robot soccer Dec-POMDP model and solve it by an approximate Dec-POMDP algorithm [5, 6] using genetic algorithms.

## 3. METHOD

We model 2D robot soccer as Dec-POMDP and solve it using a genetic algorithm based approach. The genetic algorithm searches for a good policy in the joint policy space which is dependent on the policy representation. Although, the generation of the full observation-action tree encompasses all possible policies, it is not tractable due to combinatorial explosion. In this work we use the

FSC policy representation, and the neural network representation.

### 3.1 Dec-POMDP Model

The *Decentralized Partially Observable Markov Decision Process (DEC-POMDP)* [3] model consists of the 7-tuple

$$(n, S, A, T, \Omega, O, R)$$

where:

- $n$  is the number of agents.
- $S$  is a finite set of states.
- $A$  is the set of joint actions which is the Cartesian product of  $A_i$  ( $i = 1, 2, \dots, n$ ) i.e. the set of actions available to  $agent_i$ .
- $T(s, a, s')$  is the state transition function which determines the probabilities of the possible next states given the current state  $s$  and the current joint action  $a$ . Here,  $s, s' \in S$  and  $a \in A$ .
- $\Omega$  is the set of joint observations which is the Cartesian product of  $\Omega_i$  ( $i = 1, 2, \dots, n$ ) i.e. the set of observations available to  $agent_i$ . At any time step the agents receive a joint observation  $o = (o_1, o_2 \times \dots \times o_n)$  from the environment.
- $O(s, a, o)$  is the observation function which specifies the probability of receiving the joint observation  $o$  given the current state  $s$  and the current joint action  $a$ . Here,  $s \in S$ ,  $a \in A$ , and  $o \in \Omega$ .
- $R(s, a)$  is the immediate reward function specifying the reward taken by the multiagent team given the current state  $s$  and the joint action  $a$ . Here,  $s \in S$ , and  $a \in A$ .

#### 3.1.1 Dec-POMDP Model of Robot Soccer

We give the formal Dec-POMDP model of robot soccer as implemented by *TeamBots* [2] robot soccer simulator:

- $n = 10$ . There are 10 robot players.
- $S = \{S_1 \times S_2 \times \dots \times S_{11}\}$  where  $S_i$  is a tuple  $\langle x, y, h \rangle$ . The state of robot soccer consists of the positions and the headings of the 10 players and the position of the ball. The tuple  $\langle x, y, h \rangle$  represents the configuration of agents where  $x$  represents the position in  $x$  direction,  $y$  represents the position in  $y$  direction, and  $h$  represents the heading of players.
- $A = \{A_1 \times A_2 \times \dots \times A_{10}\}$  where  $A_i$  is a tuple  $\langle x, y, kick \rangle$ . The action of the robots consists of movement vectors in the  $x$  and  $y$  direction, and the kick action. If the value of  $kick$  is greater than 0, the robot makes a kick with the given kick value. Here,  $x \in \mathbb{R}$ ,  $y \in \mathbb{R}$ , and  $kick \in \mathbb{R}$ .

- $T(S, A, S') = move(S, A) + \mathcal{N}(0, error)$ .

The transition probability to change the state from  $S$  to  $S'$  with joint action  $A$  is determined by the *move* function and an added artificial Gaussian noise. The *move* function calculates the next position of every robot by adding the given action vectors to the current positions of robot. If the desired heading and the current heading is not the same, the robots turn to the desired heading. If the difference between the desired heading and the current heading is greater than some threshold value, the robot turns in the direction of the desired heading with the threshold value. Then the robot is moved

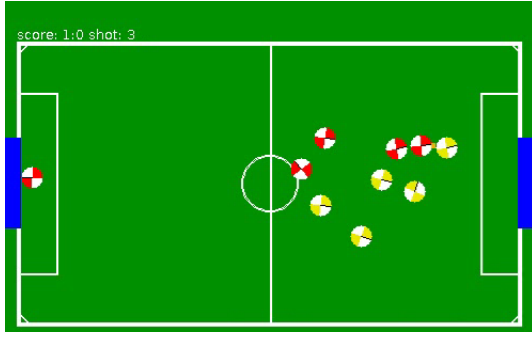


Figure 1: TeamBots Robot Soccer Simulator

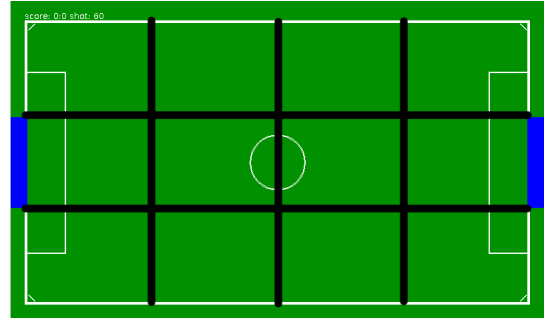


Figure 2: TeamBots Field

for the calculated heading for a certain amount. *error* determines the standard deviation of the Gaussian noise in the robot actions. Each and every robots' transition is calculated one by one. If the calculated position for an robot is not available, it does not change its state.

- $\Omega = \{S_1 \times S_2 \times \dots \times S_{10}\}$

The observation set  $\Omega$  is the cross product of the state  $S_i$  of every robot and the ball.

- $Obs(S', A, Z) = viewAngle_{1:10}(S') + \mathcal{N}(0, error)$

The observation probability at state  $S$  is defined by the current state, *viewAngle* function and a Gaussian artificial noise term. *viewAngle* function filters the full state by returning the state of robots which are in the view angle of the robot  $i$ . There is also a noise term to add artificial Gaussian noise with standard deviation *error*. By default, the robots have full view of the field.

- $R(S, A) = \begin{cases} 1 & \text{if } S_{ball} \geq OppGoal \\ -1 & \text{if } S_{ball} \geq OwnGoal \\ 0 & \text{otherwise} \end{cases}$

If the position of the ball is greater than that of the opponent goal line and also it is between the two goal posts, a goal is scored and the function returns 1. If the position of the ball is greater than that of the own goal line and also it is between two goal posts, a goal is conceded and the function returns -1. Otherwise, a 0 reward is given.

### 3.1.2 Discrete Dec-POMDP Model

To model robot soccer as a discrete Dec-POMDP model, we define a new finite set of actions and a finite set of observations. Other Dec-POMDP definitions such as the transition function, the reward function and the state set are not needed because they will be carried out by the simulator, but it is possible to redefine them according to these new action and observation sets.

We define five high level behaviors. All of the behaviors are implemented using the potential field method which represents the objects as a point. All objects are modeled as either having attractive forces or repulsive forces. The target object has attractive forces and all other objects have repulsive forces. Overall motion of the robot is determined by the summation of all active forces. More details can be found in [13]. The finite set of actions is as follows:

- $A = \{Go\ to\ ball,$   
*Go to support position,*  
*Go to defense position,*  
*Pass to the closest teammate,*  
*Pass to the teammate closest to the opponent goal}\}*

The finite set of observations is as follows: The *TeamBots* field is divided with 2 equally spaced lines from the narrow edge and 3 equally spaced lines from the wide edge. In total there are 12 grid cells as seen in Figure 2. The *Location* information is based on this grid.

We define two observation metrics in those grid cells. The first observation metric called *Dominance* has three possible values based on the number of players in the cell the ball resides:

- *Equal number of players,*
- *The opponent team has more players, and*
- *Our team has more players.*

The other observation metric is called *Closeness*. It also has three possible values which are based on which player is the closest to the ball:

- *An opponent player is the closest,*
- *A teammate is the closest, and*
- *The robot itself is the closest.*

Therefore, the observation set includes three critical pieces of information about the environment: The location of the ball in the grid, the closest player to the ball, and the team which is dominant in the cell where the ball resides.

$$Observation = Location \times Closeness \times Dominance$$

### 3.1.3 Continuous Dec-POMDP Model

For the continuous Dec-POMDP Model of robot soccer, we can directly use the observation set and action set. However, to be able to compare the performances of the two models and their respective solution methods, we keep the set of actions the same as that of the discrete Dec-POMDP model. We use the observation set defined by the original robot soccer Dec-POMDP model. Therefore, every robot will have access to the position of the robots and the ball which are in the view angle of the robot.

### 3.2 Dec-POMDP Policies

A policy is a set of rules which determines which action to choose at a particular time step given the current observations. In Dec-POMDP, a policy is a mapping function which takes observations as input and returns actions as output. For finite horizons, we can enumerate all possible observation-action selections in a tree structure. However, the complexity of the policy tree increases exponentially by the number of observations. Therefore, we need more compact solutions for policy representation. For the discrete robot soccer model, we use the finite state controller representation and for the continuous robot soccer model, we use the neural network representation. However, every policy representation has its own limitations and advantages. While finite state controller can handle observation history, it cannot be used for continuous Dec-POMDP model. On the other hand, a neural network can handle continuous Dec-POMDP, but it does not maintain observation-action history.

#### 3.2.1 Finite State Controller Representation

Finite state controllers (FSC) are widely used in Dec-POMDP to represent policies. For finite horizon Dec-POMDPs, finite state controllers provide a compact policy representation. A FSC is a special finite state machine which consists of a set of states and transitions. The states of FSC are different than the states of Dec-POMDP model. For clarity, FSC states will be called as *nodes*. Every FSC node will be associated with an action. At every time step, the agent executes the action of the current node. The transition of the current node is executed based on the current observations. An example FSC can be seen in Figure 3. This FSC has two observations and three actions. A FSC always has a starting node. If the starting node is  $S1$ , at first,  $A1$  action is executed. If the agent gets an observation  $O2$ , it updates its current node  $S1$  to  $S2$  and executes the action  $A2$ . The action selections and node transitions are executed until the end of the episode.

A FSC can be represented as a 5-tuple  $(Q_i, A_i, \Omega_i, \psi_i, \eta_i)$  where:

- $Q_i$  is a finite set of FSC nodes
- $A_i$  is a finite set of actions
- $\Omega_i$  is a finite set of observations
- $\psi_i : Q_i \rightarrow A_i$  is node-action mapping function
- $\eta_i : Q_i \times \Omega_i \rightarrow Q_i$  is transition function

As it is obvious from the definition, with sufficient number of nodes, a FSC can represent all possible observation-action trees. The choice of the number of nodes is the only parameter of a FSC policy. We can have FSCs having arbitrary number of nodes. If the policy will be able to execute all available actions, the number of nodes should be at least the same as the number of actions. However, increasing the number of nodes, increases the policy complexity. Since having greater number of nodes than the number of actions does not improve the performance of the algorithm [6], we keep the number of FSC nodes equal to the number of actions.

#### 3.2.2 Neural Network Representation

A neural network is a network of interconnected artificial neurons. In this study we use the multi-layer artificial neural network model [8]. Every artificial neuron has a weight value for every incoming connection. A neuron activates the outgoing connected neurons if the weighted sum of the incoming connected neuron values are greater than the threshold value. There are three types of layers of neurons:

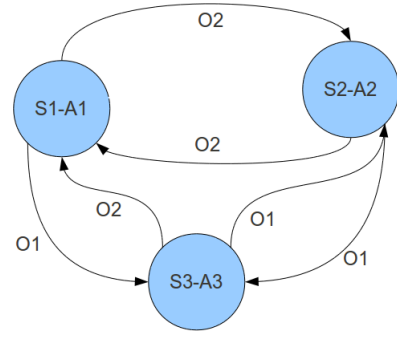


Figure 3: An Example Finite State Controller

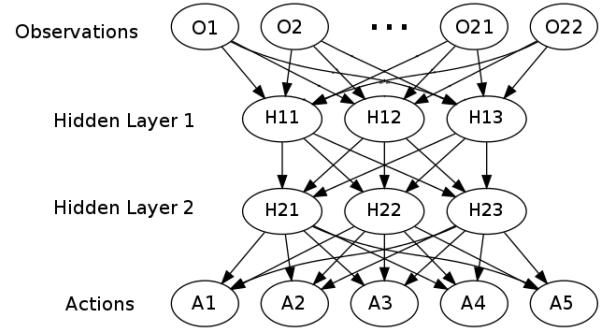


Figure 4: An Example Neural Network

- The **input layer** consists of neurons which are activated by system inputs.
- A **hidden layer** is the layer between input and output layers. There can be more than one hidden layer.
- The **output layer** consists of neurons which provide the solution of the neural network.

A fully connected neural network example can be seen in Figure 4.

A neural network can take observations as inputs and return outputs as actions. When we use the neural network policy representation with a Dec-POMDP having discrete action set, we need to discretize real-valued outputs. There are as many output neurons as the number of actions. Every output neuron corresponds to an action. The algorithm chooses the action whose output value is the greatest. With these extensions, if the architecture of the neural network is fixed, the problem turns into the optimization of the parameters of a neural network. In other words, the problem becomes an optimization problem where the weights of the artificial neurons are optimized.

One of the most significant advantages of the neural network representation is that there is no need for the discretization of observations and actions. The neural network can take real numbers as inputs and provide real number outputs. This is a very powerful feature because many real life problems, by nature, are continuous and there is information loss when they are discretized. However, a neural network can represent memoryless policies. It assumes the Markov property for observations which is not the case for many problems. Generally, the sequence of observations depends on each other.

### 3.3 Genetic Algorithm Based Approach

In a genetic algorithm [9], a solution to a given problem is represented as a chromosome. A population is constructed by a set of chromosomes. The genetic algorithm evolves the population to achieve better solutions by evolutionary operators. The goodness of a solution is determined by the fitness calculation. Selection, crossover, and mutation are commonly used evolutionary operators. A new population is generated by the application of evolutionary operators to the current population. When a new population is created, the algorithm decides whether it should terminate the evolution. The evolution ends either when the solution converges or evolution reaches to the maximum number of generations. To solve a search problem with genetic algorithm, we should be able to encode candidate solutions as chromosomes and calculate their fitness.

The selection operator chooses the parents which will take part in the creation of the new generation. The most intuitive idea is choosing the best chromosomes. However, this prevents having diversity in the population. We use roulette wheel selection operator. Every chromosome can be chosen proportional to its fitness value.

Once the selection operator chooses parents, other evolutionary operators are applied to the parent chromosomes. The mutation operator makes random changes to a random gene of a chromosome with the mutation probability.

The crossover operator is used to create new children from the parent chromosomes. It combines two chromosomes to create new offspring by changing the parts of chromosomes from a randomly chosen point.

The fitness of a policy is calculated by running many simulations. Although fitness have been shown to stabilize after 1000 simulations for the Dec-POMDP benchmark problems [6], the number of simulations for stable fitness calculation is problem dependent. As the number of simulations increases the accuracy of the fitness estimation increases. For the genetic algorithm, it is sufficient to know the general characteristic of the policy.

To deal with the uncertainty in the fitness calculation, we use a three stage genetic algorithm by maintaining a best chromosome list. The main difference between the stages is the precision of the fitness calculation. If we use a simple genetic algorithm with a very high precision fitness calculation, we can get similar results. However, in this case the algorithm's running time becomes infeasible. With such a three stage genetic algorithm, we calculate the fitness of the candidate solution in a sufficiently precise manner to achieve the expected functionality.

The stages of the algorithm are as follows:

- *Pre-Evolution Stage:* Initially  $k$  best chromosomes are calculated and put into the best chromosomes list. In this phase a higher precision fitness calculation is used because the success of the genetic algorithm is dependent on the initial population. If there are good solutions at the beginning, the next generations will be reproduced from them.

For some problems, very good solutions may exist in the population, but their fitnesses may be calculated to be very low. By having a high precision calculation, we guarantee not to miss such a solution [6]. Therefore, at the beginning of the evolution and at the end of every evolution cycle, good chromosomes' fitnesses are recomputed. If they are still good to be in the list of best chromosomes, they are added to the best chromosomes list.

- *Evolution Stage:* After each generation, 10 best chromosomes of the current *population* are compared with the chromosomes which are in the best chromosomes list. If one of

A1	A2	A3	S3	S2	S3	S1	S2	S1
Best Action at S1	Best Action at S2	Best Action at S3	O1 is taken at S1	O2 is taken at S1	O1 is taken at S2	O2 is taken at S2	O1 is taken at S3	O2 is taken at S3

Figure 5: An Example FSC Encoding

the chromosomes is good enough to be in the best chromosomes list, its fitness is calculated with higher precision. If it is still good enough to be in the best chromosomes list, it is added to the list. This phase has the least precise fitness calculation because the fitness of the chromosomes will be used only to rank the chromosomes. Once the chromosomes are ranked, basic genetic operators are applied. In order to converge to a good solution, it is enough to identify bad solutions, good solutions, and the intermediate ones.

- *Post Evolution Stage:* This phase has the highest precision calculation because it is the final phase of the algorithm. The evolution ends if one of two conditions holds, either the generation count reaches a prespecified maximum number or the fitness of the best chromosome does not change for some generations. When the termination criteria is satisfied, the evolution is terminated. The best solution will be calculated from the best chromosomes list. The fitness of the solution should be calculated as precisely as possible so that the algorithm does not choose a moderate chromosome as the final solution while there is a better solution in the list.

The training of the Dec-POMDP teams is carried out against standard *TeamBots* teams which have different strengths. Therefore, in the beginning of the training, most of chromosomes have negative fitness value against strong teams. We use an iterative genetic algorithm which starts training against the easiest team and continues with stronger teams. It changes the opponent team with a stronger team when the algorithm cannot improve the best chromosome or the maximum generation number is reached.

#### 3.3.1 FSC Encoding

FSC represents the policies of discrete Dec-POMDP models. In this study, the encoding of a FSC is as follows: the first  $n$  genes represent node-action mapping and their values are between 1 and the number of actions ( $A$ ). Then, for each node, there is an observation-node mapping which denotes the transition when an observation is taken as seen in Figure 5. The value of this range is between 1 and  $S$  which represents the number of nodes. The whole chromosome of the Dec-POMDP policy is constructed by concatenating each and every robot's policy.

#### 3.3.2 Neural Network Encoding

Neural network representation is used to represent policies of continuous Dec-POMDP models. In this study, the architecture of the neural network is fixed. In other words, the number of neurons in the input layer, hidden layer, and output layer are determined before the optimization process. In addition to the number of neurons, the connections between neurons are also fixed. We use fully connected feedforward neural networks. Therefore, it is sufficient to encode only the weights of the connections. An example encoding





Figure 6: An Example Neural Network Encoding.

Table 1: Parameters of the Genetic Algorithm

Parameter	Value
Population Size	30
Mutation Rate	0.2
Crossover Rate	0.5
$N_B$ : Number of Simulations Before Evolution	10
$N_D$ : Number of Simulations During Evolution	5
$N_A$ : Number of Simulations After Evolution	50
Fitness Metric	Score
Maximum Number of Generations	50
Convergence Limit	20

can be seen in Figure 6. Every value corresponds to the weight of the connection between two neurons, which are identified in Figure 4. The weights are concatenated starting from the weights of input layer. The real valued chromosome is the concatenation of each and every robot’s policy.

## 4. EXPERIMENTS AND RESULTS

Genetic algorithm is implemented using the *JGAP* genetic algorithms package [11]. We use *TeamBots* simulation environment to train the Dec-POMDP policies. The *TeamBots* simulator comes with four standard teams. They are in the order of increasing power: *BrianTeam*, *Kechze*, *SibHeteroG*, *AIKHomoG*. We also add an immobile *NullTeam* which is used to learn very basic behaviors. Games are played with teams of five players for 6000 time steps. Training starts against the weakest team, i.e. *NullTeam*, and continues with the next weakest team *BrianTeam* to the strongest team *AIKHomoG*.

Training is carried out in stages. We first train against the *NullTeam*, then against the other standard *TeamBots* teams, in the order of increasing difficulty. The population of a previous team is used for the next team except the *NullTeam* whose population is randomly initialized.

The algorithm performance is highly dependent on the parameters which are generally problem dependent. However, genetic algorithm parameters do not change from problem to problem. We determine the genetic algorithm parameters empirically. Another important parameter is the architecture of the neural network which we use to represent policies. Also, the architecture of the neural network is determined empirically. We use two hidden layers each having 3 nodes and whole neural network is fully connected. There are 22 nodes in input layer and 5 nodes in output nodes. Input layer takes 22 positions of 10 robots and a ball. There are 5 primitive behaviors. Every output node corresponds to one behavior and the action is selected as the node having the greatest value.

One of the most important genetic algorithm parameters is the number of simulation runs. The fitness of a policy is calculated by running many simulations and taking their mean. There is a trade-off between the number of simulation runs and the running time of the algorithm. Table 1 gives the list of genetic algorithm parameters.

### 4.1 Fitness Calculation

There are two important problems in fitness calculation. The pre-

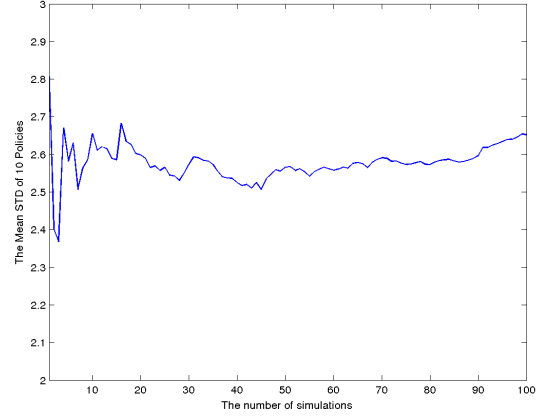


Figure 7: The change of the mean standard deviation of 10 random policies with the number of simulations.

cision of the fitness calculation is determined by the number simulations. Therefore, we need to determine the number of simulations for different stages of the genetic algorithm. Another issue is the choice of fitness value. Although, game score is the best indicator of the success of a policy, it is a sparse-reward. In our previous work [1], we compare performances of different fitness calculation methods. We showed that the score is the best fitness method when used with iterative genetic algorithm.

As we increase the number of simulations, the precision of fitness calculation increases. However, we need to run sufficient number of simulation runs to have the best running time performance. In Figure 7, we can see the change of the mean standard deviation of 10 random policies with the number of simulations. We assume that policies have different mean fitness values, but same standard deviation. Therefore, we expect oscillations at first, but finally stabilization. Those oscillations are in small range, 0.2, and gets smaller as the number of simulations increases. Therefore, we choose 10 for *before evolution* stage, 5 for *evolution* stage, and 50 for *post-evolution* stage as seen in Table 1.

### 4.2 Performance of the Dec-POMDP Team

In Figure 8, we see the evolution of the neural network based policy representation. As expected, the neural network policy quickly learns against easy teams such as *NullTeam* and *BrianTeam*. However, as the strength of the team increases its performance decreases. Although the policy is trained against *SibHeteroG* for 50 generations, it makes little improvements in its fitness score. Therefore, as the complexity of the problem increases, we should increase the number of generations to train better policies.

In Table 2, we report the result of 500 evaluation runs. After training, the best policy plays 500 games against the standard *TeamBots* teams. The neural network based approach outperforms the reinforcement learning based approach since it has positive average scores against strong teams such as *SibHeteroG* and *AIKHomoG*. When we compare the neural network based approach with the FSC based approach, the neural network based approach performs better against easy teams but its performance decreases as the strength of the opponent team increases. Since a neural network is a memoryless policy representation, it can learn simple policies very well, but it may not learn complex ones. To be able to win against strong teams, we need to evolve more complex policies. However, it may

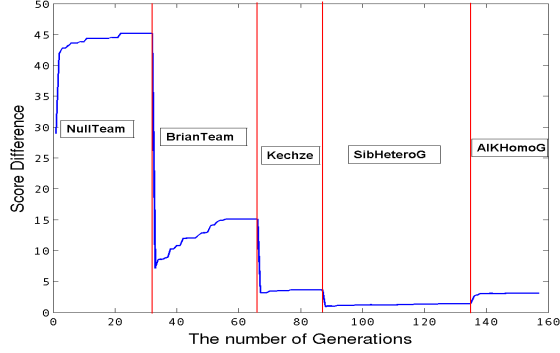


Figure 8: The Score Evolution

Table 2: The Comparison of Average Scores

Opponent Team	Neural Network Based Approach	FSC Based Approach [1]	RL Based Approach [12]	Random Policy
NullTeam	16.34	8.42	28.25	6.56
BrianTeam	10.50	7.04	17.80	1.20
Kechze	3.28	3.68	12.67	-0.96
SibHeteroG	0.94	1.31	-4.90	-0.74
AIKHomoG	1.14	2.48	N.A.	-2.11

not be possible for certain policies to be represented by a neural network.

There is a difference between the score at the end of evolution and the average score of 500 evaluation runs. This difference gets bigger as the number of generations increases. Since training starts with the easy teams, the difference is the greatest for them. In other words, the final best policy is highly adapted to the last team. Another reason is that the policies are trained to get higher scores for easy teams compared to stronger teams. The results reported for the Dec-POMDP model where robots can see the full state of the process and there is noise only in the movement of the ball are given in Figure 8 and Table 2

#### 4.2.1 Finite State Controller vs. Neural Network

When we compare the performance of two policy representation methods, there is a significant performance difference for strong teams such as *SibHeteroG* and *AIKHomoG*. For weak teams both approaches are almost equally successful. When we look at their success against each other, we see that they learn similar policies. The results of 500 evaluation games is given in Table 3. Most of the games finish with a draw and very few goals are scored. When we watch the games, we see that for the most of the games, the players get stuck around the ball. Since neither of the teams are able to move the ball, the games generally end with a draw.

### 4.3 Uncertainty and Partial Observability

In order to introduce uncertainty to the problem, we add artificial Gaussian noise to the observations and actions. Every robot chooses a discrete action from the action set. Those primitive actions produce an action vector which determines where the robot will move at the next time step. The simulator determines the heading of the desired motion and turns the robot according to the current heading. Then, the robot moves towards the heading.

Table 3: Game Scores: FSC vs. Neural Network.

	FSC Team	Neural Network Team
Average Score	0.021	-0.021
Win	43	35
Draw	422	422
Loss	35	43
Max Score	2	2

Table 4: Average Scores when Uncertainty and Partial Observability is Added

Opponent Team	Uncertainty	Partial Observability	Noisy Free
NullTeam	24.20	19.17	16.34
BrianTeam	4.97	10.01	10.50
Kechze	5.72	3.50	3.28
SibHeteroG	-0.01	0.06	0.94
AIKHomoG	1.31	-1.23	1.14

Therefore, there are two components of the movement: turning and translation. We add a zero mean, 0.01 standard deviation Gaussian noise to the turning action and a zero mean, 0.05 standard deviation Gaussian noise to the translation action. The radius of a robot is 0.06. We distort %30 actions 0.05 meters and more.

Every robot can take the position information of other agents and the ball. We add a zero mean, 0.05 standard deviation Gaussian noise to the positions of the robots. In Table 4, we show the effect of uncertainty to the game scores. In spite of the uncertainty, the policy achieves to retain its performance.

To achieve partial observability, we restrict the view angle of agents. By default, every robot can access the position of the other robots and the ball. We assume that the players of the Dec-POMDP team can view the outside world with a 120 degree view angle. We calculate the view cone of the robot from the current heading. We omit any superposition and assume that robot can see any robot which is in the view angle. Also, some of the primitive actions require the knowledge of other robots and the ball. We provide position information for the primitive actions and restrict it only as observations. Since the neural network gets the position of every robot and the ball as observation, the handling of missing observations is important. If a robot is not in the view angle, we use the last position of the robot as an input to the neural network.

The result of partial observability can be seen in Table 4. Although we restrict the view angle of robots, their performance is retained except the strong teams *SibHeteroG* *AIKHomoG*. Although we have negative average scores against *AIKHomoG*, the team loses most of the game with only one goal difference.

## 5. CONCLUSIONS

We model multi-agent decision making in robot soccer as Dec-POMDP and solve it using scalable algorithms. We show that it is possible to solve complex decision making problems with the genetic algorithms [6]. In addition to the finite state controller policy representation, we show the application of neural network policy representation for continuous Dec-POMDP model. Although, policy representations are different, the algorithm is able to learn similar policies. We show that the learned policies easily outperform the standard *TeamBots* teams. We also show that the performance of the algorithm is not severely affected from the noise and partial observability.

However, one of the limitations of the algorithm is the running

time. When there is a very little difference between the fitness of policies, to achieve convergence and improvement, we need better fitness estimations. To achieve better fitness estimations, we need to increase the number of simulations. However, the algorithm should have a feasible running time.

As a future work, we plan to improve the genetic algorithm so that we achieve better fitness estimations with less running time. Although we prove the usefulness of the algorithm in simulation, we aim to implement it for real robot soccer teams. There are many challenging real-world decision making problems. We aim to explore the potential of the algorithm in other domains such as search and rescue.

## 6. REFERENCES

- [1] O. Aşık and H. L. Akın. Solving Multi-Agent Decision Problems modeled as Dec-POMDP: A Robot Soccer Case Study. In *RoboCup 2012 Symposium, June 24, 2012, Mexico City, Mexico*.
- [2] T. Balch. Teambots mobile robot simulator, 2000.
- [3] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The Complexity of Decentralized Control of Markov Decision Processes. *Math. Oper. Res.*, 27:819–840, November 2002.
- [4] D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded Policy Iteration for Decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1287–1292, Edinburgh, Scotland, 2005.
- [5] B. Eker and H. L. Akın. Using evolution strategies to solve DEC-POMDP problems. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 14(1):35–47, 2010.
- [6] B. Eker and H. Akın. Solving decentralized pomdp problems using genetic algorithms. *Autonomous Agents and Multi-Agent Systems*, 27:161–196, 2013.
- [7] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Game theoretic control for robot teams. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1163 – 1169, april 2005.
- [8] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Pearson Education, NY, Second edition, 1998.
- [9] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.
- [10] A. Kumar and S. Zilberstein. Constraint-based dynamic programming for decentralized pomdps with structured interactions. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pages 561–568, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.
- [11] K. Meffert, J. Meseguer, E. D. Marti, A. Meskauskas, J. Vos, and N. Rotstan. Jgap: Java genetic algorithms package, 2011.
- [12] c. Meriçli, T. Meriçli, and H. L. Akın. A Reward Function Generation Method Using Genetic Algorithms: A Robot Soccer Case Study. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1513–1514, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.
- [13] T. Meriçli and H. L. Akın. Soccer without intelligence. In *ROBIO*, pages 2079–2084. IEEE, 2008.
- [14] A. D. Pietro, L. While, and L. Barone. Learning In RoboCup Keepaway Using Evolutionary Algorithms. In *GECCO 2002*, pages 1065–1072, 2002.
- [15] S. Seuken and S. Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 344–351, Vancouver, British Columbia, 2007.
- [16] S. Seuken and S. Zilberstein. Improved memory-bounded dynamic programming for decentralized pomdps. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 2007.
- [17] S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, Oct. 2008.
- [18] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu. Keepaway Soccer: From Machine Learning Testbed to Benchmark. In A. Bredendfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *RoboCup*, volume 4020 of *Lecture Notes in Computer Science*, pages 93–105. Springer, 2005.
- [19] P. Stone, R. S. Sutton, and S. Singh. Reinforcement Learning for 3 vs. 2 Keepaway. In *Lecture Notes in Computer Science*, volume 2019, pages 249–258, 2001.
- [20] P. Stone, R. S. Sutton, and S. P. Singh. Reinforcement Learning for 3 vs. 2 Keepaway. In *RoboCup 2000: Robot Soccer World Cup IV*, pages 249–258, London, UK, UK, 2001. Springer-Verlag.
- [21] S. Whiteson, N. Kohl, R. Mäkieläinen, and P. Stone. Evolving Soccer Keepaway Players Through Task Decomposition. *Machine Learning*, 59:5–30, 2005. 10.1007/s10994-005-0460-9.
- [22] F. Wu and X. Chen. Solving Large-Scale and Sparse-Reward DEC-POMDPs with Correlation-MDPs. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup*, volume 5001 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2007.



# A Coordinated MDP Approach to Multi-Agent Planning for Resource Allocation, with Applications to Healthcare

Hadi Hosseini  
David R. Cheriton School of  
Computer Science  
University of Waterloo  
h5hosseini@uwaterloo.ca

Jesse Hoey  
David R. Cheriton School of  
Computer Science  
University of Waterloo  
jhoey@uwaterloo.ca

Robin Cohen  
David R. Cheriton School of  
Computer Science  
University of Waterloo  
rcohen@uwaterloo.ca

## ABSTRACT

This paper considers a novel approach to scalable multi-agent resource allocation in dynamic settings. We propose an approximate solution in which each resource consumer is represented by an independent MDP-based agent that models expected utility using an average model of its expected access to resources given only limited information about all other agents. A global auction-based mechanism is proposed for allocations based on expected regret. We assume truthful bidding and a cooperative coordination mechanism, as we are considering healthcare scenarios. We illustrate the performance of our coordinated MDP approach against a Monte-Carlo based planning algorithm intended for large-scale applications, as well as other approaches suitable for allocating medical resources. The evaluations show that the global utility value across all consumer agents is closer to optimal when using our algorithms under certain time constraints, with low computational cost. As such, we offer a promising approach for addressing complex resource allocation problems that arise in healthcare settings.

## Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Multiagent Systems

## General Terms

Algorithm, Experimentation

## Keywords

Multiagent Planning, Multiagent MDP, Healthcare Applications

## 1. INTRODUCTION

This paper develops an approach for allocating resources in multi-agent systems for domains where there are multiple agents and multiple tasks, and the success of the agents carrying out tasks is dependent stochastically on their ability to obtain a sequence of resources over time. We are particularly interested in situations where agents must independently optimize over their individual states, actions, and utilities, but must also solve a complex coordination problem with other agents in the usage of limited resources.

In particular, we are concerned with allocating resources in settings that involve a set of  $N$  consumers, each of whom requires some subset of a total of  $M$  resources. The consumers each have a measure of *health*<sup>1</sup> that they are trying to optimize, and this quality is influenced stochastically by the resources they acquire and by time. Further, each consumer has a resource *pathway* that represents the partial ordering in which they need the resources. Consumers' states evolve independently over time, and are dependent only through their need for shared resources. Rewards are independent, and the global reward is the sum of individual consumer rewards.

We formulate this problem as a factored multiagent Markov Decision Process (MMDP) with explicit features for each consumer's state and resource utilization, and an explicit model of how each consumer's state progresses stochastically over time dependent on obtained resources. The actions are the possible allocations of resources in each time step. For realistic numbers of consumers and resources, however, such an MMDP has a state and action space that precludes computation of an optimal policy. This paper addresses this problem and makes three contributions:

1. We develop an approximate distributed approach, where the full MMDP is broken into  $N$  MDPs, one for each consumer. We call these consumer MDPs *agents*. Agents model the resources they expect to obtain using a probability distribution derived from average statistics of the other agents, and compute expected regret based on this distribution and on the known dynamics of their health state.
2. We propose an iterative auction-based mechanism for real-time resource allocation based on the agents' individual expected regret values. The iterative nature of this process ensures a reasonable allocation at minimal computational cost.
3. We demonstrate the advantages of our approach in a cooperative healthcare domain with patients seeking doctors and equipment in order to improve their health states. We present averages of simulations using randomly generated agents from a reasonable prior distribution. We compare our coordinated MDP approach against an alternate planning algorithm intended for large-scale applications, a state-of-the-art Monte Carlo sampling based method for solving the full MMDP model known as UCT. We also compare to two simple but realistic heuristic approaches for allocating medical resources.

Our approach is particularly well suited to large collaborative domains that require rapid responses to resource allocation demands

<sup>1</sup>We use the term *health* here in a general sense to denote a single quantity over which an agent's utility function (and hence, its reward) is defined. This can be for e.g. *quality* of a solution, *value* of an outcome, or patient state of *health*.

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with AAMAS, May 2013, St. Paul, Minnesota, USA.

in time-critical domains, and we use a healthcare scenario throughout the paper to clarify our solution. We start by introducing the MMDP model and our distributed approach, followed by descriptions of the baseline methods we compare to. We then develop a set of realistic models for use in simulation, and show results across a range of problem sizes.

## 2. MDPS AND COORDINATION

Our model is a factored MDP represented as a tuple of elements  $\langle N, M, \tau, \mathbf{R}, \mathbf{H}, P_T, \Phi, A \rangle$  where  $N$  is the number of consumers,  $M$  the number of resources, and  $\tau$  is the planning horizon.  $\mathbf{R} = \{\mathbf{R}_1, \dots, \mathbf{R}_N\}$  is a finite set of resource variables, each one representing the state of a single consumer's resource utilizations, where  $\mathbf{R}_i = \{R_{i1}, R_{i2}, \dots, R_{iM}\}$  is a set of variables representing consumer  $i$ 's utilization of resource  $j$ . Each  $R_{ij} \in \mathcal{R}$  where  $\mathcal{R}$  is the set of possible resource utilizations (how much resource is being used). We model each resource as distinct (so multiple copies of a resource are modeled separately).  $\mathbf{H} = \{H_1, \dots, H_N\}$  is a set of  $N$  variables measuring each consumer's health, each of which is  $H_i \in \mathcal{H}$  giving the different levels of health. We use  $s_i = \{\mathbf{R}_i, H_i\}$  to denote the complete set of **state variables** for consumer  $i$ , and  $S = (s_1, \dots, s_N)$  to denote the complete state for all consumers. Agent  $i$  receives a reward of  $\Phi_i(s_i, s'_i)$  for transition from  $s_i$  to  $s'_i$ , thus the multiagent system's **reward function** is  $\Phi(S, S') = \sum_i \Phi_i(s_i, s'_i)$ . The **transition model** is defined as  $P_T(S'|S, A) = \prod_i P_i(s'_i|s_i, a_i)$ , which denotes the probability of reaching joint state  $S'$  when in joint state  $S$ , and  $A$  is a set of permissible **actions**, one for each resource and each consumer representing all feasible allocations of resources (so the same resource cannot be allocated to two agents simultaneously). Resources are deterministic given the actions, and only one resource can be allocated to each consumer at a time. We assume a finite horizon undiscounted setting<sup>2</sup>.

The full MDP as described is an instance of a multiagent MDP (MMDP), and will be very challenging to solve optimally for reasonable numbers of consumers and resources. The total number of states is  $|S| = |\mathcal{H}|^N |\mathcal{R}|^{MN}$ , and the number of actions is  $\frac{N!}{(N-M)!}$ . We will show how to compute approximate (sample-based) solutions later in this paper, but first we show our approach to distributing this large MDP into  $N$  smaller MDPs, and introduce our coordination mechanism for computing approximate allocations.

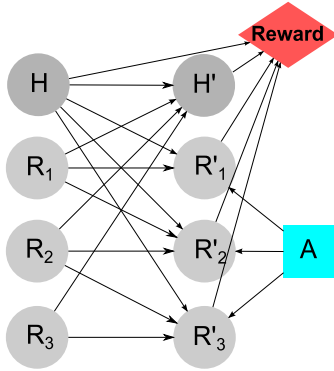


Figure 1: A patient's MDP with 3 resources shown as a two time slice influence diagram

We treat each consumer's MDP as independent (an *agent*), an

<sup>2</sup>This is realistic in healthcare scenarios as health states do not warrant discounting.

example of which is shown in Figure 1. We assume that the agent's state spaces, resource utilizations, health states, transition and reward functions are independent. The agents are only dependent through their shared usage of resources: only feasible allocations are permitted as described above (agents can't simultaneously share resources). Rewards are additive and each agent's actions now become *requests* for resources as described below. We make two further assumptions. First, the reward function for each agent is dependent on the agent's health,  $H$ , and is set to zero by a boolean factor at the end of resource acquisition (finishing the medical pathway by receiving all required resources). Second, the agent health ( $H$ ) is conditionally independent of the agent action given the current resources and the previous health, and the agent actions only influence the resource allocation, since the agent can only influence health indirectly by bidding for resources. Thus, for each agent  $i$ ,  $P_i(\mathbf{r}', h' | \mathbf{r}, h, a)$  factors as

$$P_i(\mathbf{r}', h' | \mathbf{r}, h, a) = P_i(\mathbf{r}' | \mathbf{r}, h, a) P_i(h' | \mathbf{r}, h) \quad (1)$$

where we define  $\Lambda_R \equiv P_i(\mathbf{r}' | \mathbf{r}, h, a)$  is the probability of getting the next set of resources given the current health, resources, and action, and  $\Omega_H \equiv P_i(h' | \mathbf{r}, h)$  is a dynamic model for the agent's health rate. We will refer to  $\Lambda_R$  as the *resource obtention* model and to  $\Omega_H$  as the *health progression* model.

*Health progression* is a property of a particular agent's condition or task and can be estimated from global statistics about the nature of the conditions (e.g. diseases).  $\Omega_H$  must be elicited from prior knowledge about diseases and treatments, and so forms part of a *disease model* that we henceforth assume is pre-defined (manually, or by learning based on historical statistics). On the other hand, the *resource obtention* model,  $\Lambda_R$ , will be dependent on the current state of the multiagent system, and is a property of how we are setting up our resource allocation mechanism and the expected regret computations of each agent. For example, the probability of a single agent obtaining a resource will depend on (i) the number of other agents currently bidding for that resource and (ii) the agent's model of health.

If using a single MDP for all agents as described at the start of this section, then resources would be deterministic given a joint allocation action. If modeled as a decentralized POMDP, the resources for each consumer would be conditioned on the unobservable states and actions of all the other consumers. In our model, we assume that the probability of obtaining a certain resource can be approximated reasonably well, either as a prior model based on the known distribution of diseases and the known requirements for treatments of each disease, or as a learned distribution based on simulated or real experiments.

In general, we can make no assumptions about further conditional independencies in the resource allocation factor. That is, the probability of obtaining a resource  $R'$  at time  $t$  may depend stochastically on the set of resources at time  $t - 1$ . However, in many domains, there may be further independencies that can be encoded in the model. For example, in Figure 1, resource  $R'_i$  is conditionally independent of all resources  $R_j$  where  $j \notin \{i, i - 1\}$  (for  $i > 1$ ) and for  $j \notin \{i\}$  (for  $i = 1$ ), so the resources are *ordered* according to the (linear) medical pathway of this particular patient. We assume that the health progression factor can be specified for each agent independently of the other agents.

A policy for each individual MDP is a function  $\pi_i(s_i) \mapsto A_i$  that gives an action for an agent to take in each state  $s_i$ . The policy can be obtained by computing a value function  $V_i^*(s_i)$  for each state  $s_i \in S_i$ , that is maximal for each state (i.e. satisfies the Bellman equation [2]). For simplicity of notation, we remove agent indices

and only show the indices for resources. Thus an individual agent's value function is represented as:

$$V^*(s) = \max_a \gamma \sum_{s' \in S} [\Phi(s, s') + P(s'|s, a)V^*(s')] \quad (2)$$

The policy is then given by the actions at each state that are the arguments of the maximization in Equation 2.

Agents compute their expected *regret* for not obtaining a given resource as follows. The expected value,  $Q_i(h, \mathbf{r}, a_i)$  for being in health state  $h$  with resources  $\mathbf{r}$  at time  $t$ , bidding for (denoted  $a_i$ ) and receiving resource  $r_i$  at time  $t + 1$  is:

$$Q_i \equiv \sum_{\mathbf{r}'_{-i}} \sum_{h'} P(h'|h, \mathbf{r}) V(r'_i, \mathbf{r}'_{-i}, h') \delta(\mathbf{r}_{-i}, \mathbf{r}'_{-i})$$

where  $\mathbf{r}_{-i}$  is the set of all resources except  $r_i$  and  $\delta(x, y) = 1 \leftrightarrow x = y$  and 0 otherwise. The equivalent value for not receiving the resource,  $\bar{Q}_i(h, \mathbf{r}, a_i)$ , is

$$\bar{Q}_i \equiv \sum_{\mathbf{r}'_{-i}} \sum_{h'} P(h'|h, \mathbf{r}) \bar{V}(\bar{r}'_i, \mathbf{r}'_{-i}, h') \delta(\mathbf{r}_{-i}, \mathbf{r}'_{-i})$$

Thus, the expected regret for not receiving resource  $r_i$  when in  $h$  with resources  $\mathbf{r}$  and taking action  $a_i$  is:

$$R_i(h, \mathbf{r}, a_i) = Q_i - \bar{Q}_i \quad (3)$$

We also refer to this as the expected *benefit* of receiving  $r_i$ . It is important for agents in this setting to consider regret (or benefit) instead of value, as two agents may value a resource the same, but one might depend on it much more (e.g. have no other option). Value-based bids will fail to communicate this important information to the allocation mechanism.

Note that  $Q$  is an optimistic estimate, since the expected value assumes the optimal policy can be followed after a single time step (which is untrue). This myopic approximation enables us to compute on-line allocations of resources in the complete multiagent problem, as described in the next section. In the following, we will use the notion of utilitarian social welfare by aggregating the total rewards amongst all agents as an evaluation measure.

## 2.1 Coordination Mechanism

A coordination mechanism must aim to respect the health needs of the patients to maximize the overall utility. Each agent estimates its expected individual regret given its estimate of future resources and health (as given by  $\Lambda_R$  and  $\Omega_H$ ). The regret values of different agents are compared globally, and an allocation is sought that minimizes the global regret. While the final allocation decisions are made greedily in the action-selection phase, the reported expected values of regret (for bidding) consider future rewards.

To implement this allocation, we use an iterative auction-like procedure, in which each consumer bids on the resource with highest regret. The highest bidder gets the resource, and all other agents bid on their next highest regret resource. Agents can also *resign*, receive no resources for one time step, and try again in a future time step.

## 2.2 Example

Consider a simplified scenario with 4 agents and 4 resources. We are assuming that agents require all four resources and the expected benefits for receiving resources (or regrets for not receiving resources) based on their internal utility function have been calculated as illustrated in Table 1. The worst-case scenario would be

when all the agents have attributed higher benefits to the same resources, so that their desire to acquire resources is in the same order or preference.

Agents	$r_1$	$r_2$	$r_3$	$r_4$
$a_1$	*7	8	9	<b>10</b>
$a_2$	<b>1</b>	3	*6	7
$a_3$	3	*4	5	6
$a_4$	5	6	<b>7</b>	*8

(a) Worst-case

Agents	$r_1$	$r_2$	$r_3$	$r_4$
$a_1$	3	8	*9	<b>10</b>
$a_2$	1	<b>3</b>	6	*7
$a_3$	*6	4	5	3
$a_4$	5	*6	<b>7</b>	8

(b) Average-case

Table 1: Example scenarios: 4 agents and 4 resources. \*X shows the optimal allocation, while **X** shows our method.

Agents first try to acquire the resource with highest benefit. In this scenario, all agents have associated the highest benefit to  $r_4$ , however, only one ( $a_1$ ) would be successful in getting it. All agents who have lost the previous auction, will now bid for the resource with the second-highest benefit, and so on. In this case, agents  $a_2$ ,  $a_2$ ,  $a_3$  all have attributed  $r_3$  as their second highest. Our auction-based method gives a benefit of 22 (shown in **bold** in Table 1a). The optimal allocation has the benefit of 25 (one shown with \* in Table 1a).

Table 1b shows an average-case scenario. Again we are assuming all agents require all the resources but with more diverse preferences over the set of resources. Our method gets a benefit of 26 compared to the optimal benefit of 28.

## 3. BASELINE SOLUTION METHODS

### 3.1 Sample-Based

We will compare our algorithm to the result of a sample-based solution on the full MMDP as described at the start of this section. UCT is a rollout-based Monte Carlo planning algorithm [11] where the MDP is simulated to a certain horizon many times, and the average rewards gathered are used to select the best action to take next. To balance between exploration and exploitation, UCT chooses an action by modeling an independent multi-armed bandit problem considering the number of times the current node and its chosen child node has been visited according to the UCB1 policy [1]. In general, UCT can be considered as an any-time algorithm and will converge to the optimal solution given sufficient time and memory [11]. UCT has become the gold standard for Monte-Carlo based planning in Markov decision processes [10].

To rollout at each state, we use a uniform random action selection from the set of permissible actions at each state. The permissible actions are the ones that do not cause any conflict over resource acquisition. Subsequently, the best action is then chosen based on the UCB1 policy. The amount of time UCT uses for rollouts is the *timeout*, and is a parameter that we must set carefully in our experiments, as it directly impacts the value of the sample-based solution. Although in some resource allocation settings lengthy decision periods would not have any impact on the efficiency of allocations, arguably, the time for making allocation decisions can be important in domains requiring urgent decisions such as emergency departments and environments exposed to significant change. Delayed decisions for critical patients with acute conditions in emergency departments can have huge impact on effectiveness of treatments [6]. Moreover, the allocation solution may become useless by the time an optimal decision is computed as a result of fluctuations in demand, and hence, requires recomputing the allocation decision. We will compare to UCT using a number of different realistic *timeout* settings.

### 3.2 Heuristic methods

We use three heuristic methods. In the first, only the agent’s level of criticality is considered (we call this “sickest first”). In the second, we use the reported regret values and only run one round of the auction-based allocation (so only one agent gets a resource at each time step: the agent with the biggest regret for not getting it). In the third, patients are treated in the order they arrive (first-come, first-served or FCFS - a traditional healthcare method).

## 4. EXPERIMENTS AND RESULTS

We demonstrate our approach in simulations with realistic probabilistic models of different conditions (e.g. diseases) and health and resource dynamics distributions. The simulations use a random sampling of agent MDPs, drawn from a realistic prior distribution over these models. It is important to note that we are not simply defining a single patient MDP, but rather our results are averages over randomly drawn MDPs: each simulated patient is different in each simulation, but drawn from the same underlying distribution.

We make three main assumptions. First, we assume that task durations are identical (e.g. it always takes one unit of time to consume each resource). The second assumption is that each agent is only able to bid on a single resource at each bidding round (but each bidding round includes a sequence of bids to determine the action for each MDP). The third assumption is that all patients arrive at the same time.

### 4.1 Agent Setup

We assume that the health variable  $H \in \{\text{healthy}, \text{sick}, \text{critical}\}$ , and each resource variable  $R_i \in \{\text{have}, \text{had}, \text{need}\}$ . Patients all start (enter the hospital) with  $H = \text{sick}$  and, depending on the resources they acquire, their health state improves to healthy or degrades to the critical condition. We further define a function to encode the states of the health variables as  $\nu(h) = \{0, 1, 2\}$  for  $h = \{\text{healthy}, \text{sick}, \text{critical}\}$ . We assume that there are  $D$  possible conditions (diseases), each with a *criticality level*, a real number  $c_d \in [1, 2]$  with  $c_d = 2$  being the most critical disease (makes the patient become sicker faster).

We first assume a multinomial distribution over the  $D$  conditions drawn from a set  $\mathcal{D}$ , such that each patient has condition  $d \in \mathcal{D}$  with probability  $\phi_d(d)$ . In the following, we assume conditions to be evenly distributed:  $\phi_d(d) = 1/|\mathcal{D}|$ , although in practice this distribution would reflect the current condition distribution in the population, community or hospital. Each condition has a *condition profile* that specifies a set of resources in a specific order that is derived from the clinical practice guidelines or the *medical pathway*, a distribution over *health state progression* models,  $\Omega_H$ , and a distribution over *resource obtention* models,  $\Lambda_R$ .

The medical pathway can be specified either within the  $\Omega_H$  (by making any set of  $\mathbf{r}$  not on the pathway lead to non-progression of the health state), or within  $\Lambda_R$  (by making it impossible to get resource allocations outside the pathway). We choose the latter in these experiments, but in practice the pathway may need to be specified by a combination of both, particularly if there is non-determinism in the pathways (i.e. different pathways can be chosen with different predicted outcomes). We assume that pathways for all agents are a linear chain through the required resources for each condition.

For our experiments, we have built priors over  $\Omega_H$  and  $\Lambda_R$  based on our prior knowledge of the health domain. We have made these priors reasonably realistic (capture some of the main properties of this domain), and sufficiently non-specific to allow for a wide range

of randomly drawn transition functions in the patient MDPs. In practice, these priors would be elicited from experts or learned from data.

**Health state progression model:** For each simulated agent,  $\Omega_H$  is drawn from a Dirichlet prior distribution over the three values of  $H'$  that puts more mass on the probability of healthier states (compared to the current health state) if the required resources are obtained, but more mass on the probability of sicker states if the disease is more critical. More precisely, define  $\omega_H \sim \text{Dir}(\alpha_H(d, \mathbf{r}))$  where  $\alpha_H$  is a triple of values over  $H = \{\text{healthy}, \text{sick}, \text{critical}\}$  and  $|\omega_H| = 1$ . If all the required resources are  $r = \text{had}$  in  $\mathbf{r}$ , then  $\alpha_H(d, \mathbf{r}) = (12, 4c_d, 2c_d)$ . If all required resources are either  $r = \text{had}$ , or  $r = \text{have}$ , then  $\alpha_H(d, \mathbf{r}) = (12, 4c_d, 4c_d)$ . Finally, if all the resources are needed, then  $\alpha_H(d, \mathbf{r}) = (4, 4c_d, 10c_d)$ . For all the other values of  $\mathbf{r}$ , i.e. the ones with partial resources needed, we define  $\alpha_H(d, \mathbf{r}) = (4, 10c_d, 10c_d)$ . Now for sampling purposes, we use these Dirichlet priors as parameters of multinomial distributions to sample the progression of health state. We have assumed similar progression of health over health states for all possible transitions based on  $\omega_H : (\omega_{H,1}, \omega_{H,2}, \omega_{H,3})$ . Thus,

$$\Omega_H \equiv P(h'|h, \mathbf{r}) = \begin{cases} (\omega_{H,1}, \omega_{H,2}, \omega_{H,3}) & \text{if } h = \text{sick} \\ (\omega_{H,1}, \omega_{H,3}, \omega_{H,2}) & \text{if } h = \text{healthy} \\ (\omega_{H,2}, \omega_{H,1}, \omega_{H,3}) & \text{if } h = \text{critical} \end{cases}$$

where  $\omega_{H,i}$  is the  $i^{\text{th}}$  element of  $\omega_H$ .

**Resource obtention model:** For each simulated agent,  $\Lambda_R$  is drawn from a Dirichlet prior distribution over the three values of  $R'$  that puts more mass on the probability of getting a resource if it is the next in the medical pathway, and if the patient is more sick (so their regret and bids will be larger, making it more likely they will get the resource). However, the probability mass shifts towards not getting a resource as  $N$  gets larger (so the more agents in the system, the less likely it is to get a resource). Recall from above that this model is meant to summarize the joint actions of  $N$  other agents, as would have been modeled in a full dec-POMDP solution. An adequate summary is important for good performance, and while we do not claim that the following prior is optimal, we believe it to be a good representation for these simulations. Ideally this function would be computed from the complete model directly, or learned from data. We define  $\Lambda_R \sim \text{Dir}(\alpha_r(N, h, \mathbf{r}))$  where  $\alpha_r$  is a triple of values over  $R = \{\text{have}, \text{had}, \text{need}\}$ . We define  $\nu'(h) = (1, 5, 10)$  for  $h = (\text{healthy}, \text{sick}, \text{critical})$ . If all resources in  $\mathbf{r}$  are either *had* or *have*, then  $\alpha_r = (10\nu'(h), \nu'(h), N)$ . If the previous resource in the medical pathway is *need*, then  $\alpha_r = (\nu'(h), 5\nu'(h), 10N)$ . Finally, if all resources are needed, then  $\alpha_r = (\nu'(h), \nu'(h), N)$ .

**Reward function:**  $\Phi(h, h')$  is fixed for all the agents, and rewards agents for becoming healthy, but penalizes them for staying sick or going to the critical state. More precisely: for  $h' = (\text{healthy}, \text{sick}, \text{critical})$ ,  $\Phi(h = \text{healthy}, h') = (10, -5, -10)$ ,  $\Phi(h = \text{sick}, h') = (15, 0, -5)$ , and  $\Phi(h = \text{critical}, h') = (5, 0, -5)$ . Further, once a patient is *healthy* and has received all resources, they are discharged and receive no further reward.

### 4.2 Results

We ran each of the benchmarks on a machine with 3.4GHz Quad-Core AMD and 4GB RAM available. We compare our auction-based coordinated MDP approach with (AucMDP-RegIter) and without (AucMDP-Reg) iteration using the expected regret bidding mechanism. We also compare to a version where agents only bid their expected values, not regrets (AucMDP-Iter), FCFS, sickest-first, and sample-based (UCT). Each simulated patient is randomly assigned a condition profile and then an MDP model with parameters

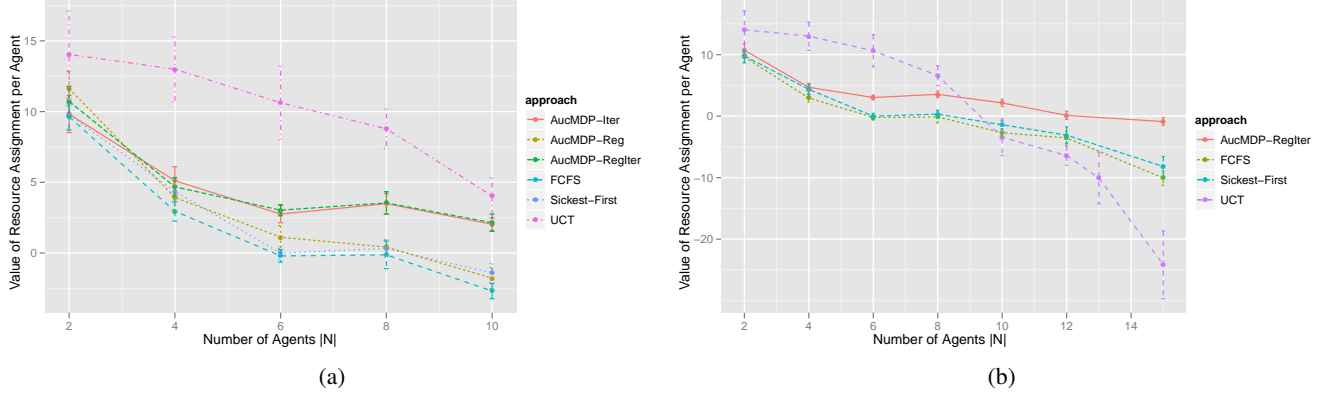


Figure 2: Evaluation of various approaches based on expected regret (AucMDP-Reg), expected value with iteration (AucMDP-Iter), expected regret with iteration (AucMDP-RegIter), and UCT with  $R = 4, D = 4$ . (a): Timeout is 300 seconds,  $\tau = 10N$  (b): Timeout is 120 seconds,  $\tau = 10N$

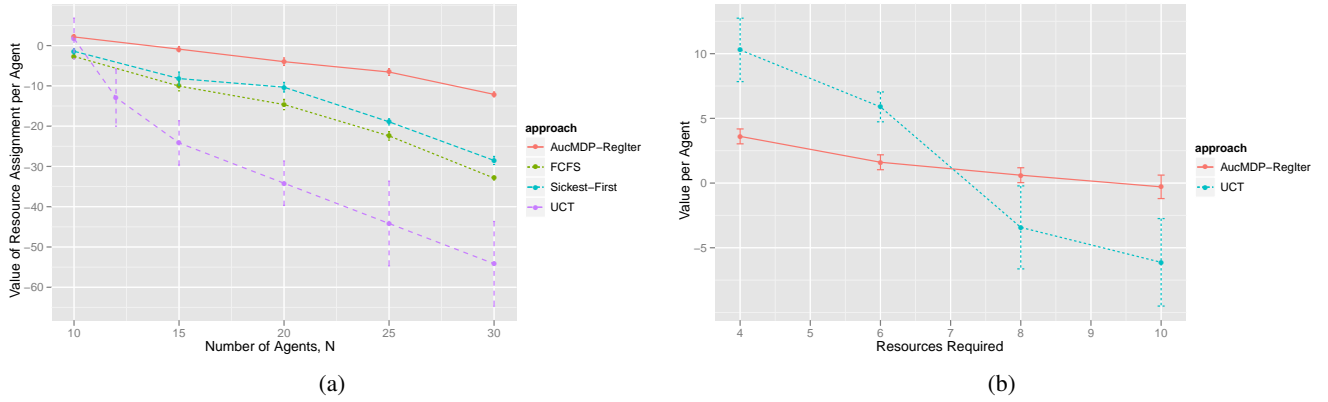


Figure 3: (a) Scaling to 30 agents, UCT with 10mins timeout and  $\tau = 20, R = 4, D = 4$  (b) Increasing required resources (actions), UCT with 60 seconds timeout and  $N = 6$

randomly drawn from the Dirichlet distributions defined above is assigned. 100 trials are done for each randomly drawn set of conditions and MDPs, and this is repeated 10 times. For the UCT results, we ran 10 trials, also repeated 10 times.

We present means and standard deviations over these simulations. We first present results with 4 total resources types and each agent requiring 4 resources based on randomly assigned condition profiles (Figure 2a). The y-axis is the average reward per patient gathered over an entire trial. We use a horizon that depends on the number of agents ( $\tau = 10N$ ), and UCT is given a 300 second timeout. The total computation time of the complete allocations for the AucMDP approach is less than 10 seconds for problems with 10 agents, and this computation time increases linearly with the number of agents and resources (as opposed to exponential growth in the MMDP case). We can see that the two AucMDP iterative approaches perform similarly, and outperform the heuristic approaches for  $N > 6$ . UCT is given sufficient time to outperform all other approaches.

Figure 2b shows the performance of our approach in a more realistic scenario with timeout set to a maximum of 120 seconds for rollouts. Similarly, each agent requires 4 resources. When the number of agents increases to more than 8 agents, UCT underperforms

compared to AucMDP, providing a policy as inferior as FCFS or sickest-first. This is mostly due to the fact that the number of possible actions grows exponentially by adding more agents, and thus, UCT requires significantly more rollouts in the action exploration phase. Figure 3a shows a further scaling to  $N = 30$ , again showing that our AucMDP approach outperforms the other methods for the larger problems. The number of joint actions also grows exponentially when the number of resources required by each agent is increased, since there are more individual options, but our AucMDP handles this well as a result of linear growth in the number of actions (Figure 3b).

As more resources are added into the system, the performance of approaches such as FCFS and sickest-first get closer to our approach because more diverse sets of resources are defined by condition profiles. Figure 4a denotes that introducing more resources yields more diversity in resource requirements: the allocation problem becomes “easier” to solve (fewer conflicts of interest), i.e., the smaller number of resources results in harder allocation. Figure 4b shows results of further scaling our AucMDP approach to 50 agents each requiring 10 resources with 10 condition profiles.

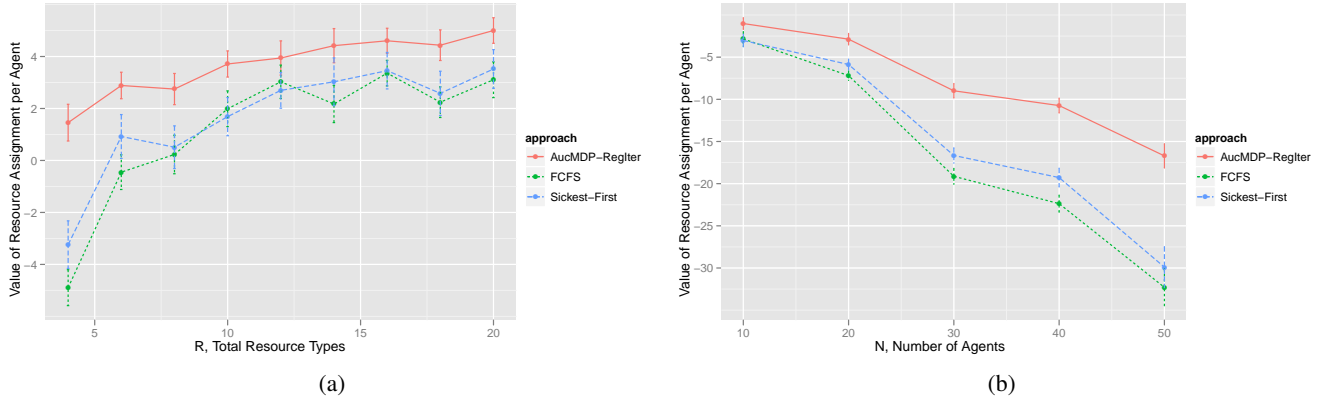


Figure 4: (a) Varying total resource types  $R = 20$ ,  $D = 5$ ,  $N = 10$ , more diversity in resource requirements results in fewer resource conflicts, (b) Scaling our auction-based coordination approach to  $N = 50$ ,  $R = 10$ ,  $D = 10$ : Comparison with traditionally practiced heuristic methods in healthcare.

## 5. RELATED WORK AND CONCLUSION

Our approach to coordinating MDPs contrasts with those of multi-agent MDPs [5] and dec-MDPs [9] in finding exact solutions, which face complexity problems for large-scale problems such as ours [3]. Instead, we offer an approximation method that collapses the state space of each agent down to only features that are available locally, and uses averaged effects of other agents for coordination. This is similar in spirit to [4] where effects of actions are estimated by agents (but without the central coordination, as in our work).

Our approach to resource allocation assumes additive utility independence, as in [13], and has state and action spaces decomposed into sets of features, with each feature relevant to only one subtask, but for cooperative settings, to maximize global utility. The use of auctions to coordinate local preferences through MDPs is also proposed in [8] where individual MDPs are submitted to a central decision maker to eventually solve the winner determination problem through a mixed integer linear program (MILP). However, this model only provides one-shot allocations and is not applicable to environments with dynamic agents or resources. Multiple allocation phases are addressed in [20], but the solution incurs greater communication overload with full agent preferences being modeled. Both approaches require a full preference model of all agents and their MDPs to be submitted to the auctioneer, which increases the computation effort on the side of the auctioneer for solving an MMDP and requires complicated (and often large) communication overload while raising privacy concerns. The work of [12] also addresses cooperative scenarios using auctions for allocating tasks to agents with fixed types and no individual preference models. However, we employ a multi-round mechanism to assign multiple resources to dynamic agents, with expected regret dictating winner determination.

The problem of medical resource allocation is perhaps best addressed to date by [17, 18] which also integrates a health-based utility function to address fairness based on the severity of health states. This model does not, however, consider temporal dependency when determining allocations and our approach of considering future events provides a broader consideration of possible uncertainty. Markov decision processes have been used to model elective (non-emergency) patient scheduling in [15].

In all, our auction-based MDP approach addresses dynamic allocation of resources using multiagent stochastic planning, employ-

ing an auction mechanism to converge fast with low communication cost. Our experiments demonstrate effectiveness in achieving global utility, using regret, for large-scale medical applications.

Future work includes exploring auction-coordinated POMDPs [4] to estimate resource demands, and learning resource models from data. We are also interested in studying combinatorial bidding mechanisms [7, 19], and bidding languages [14] in order to optimize allocations based on richer preferences. Online mechanisms and dynamic auctions [16] may also be of value to consider, to continue to explore changing environments.

## 6. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments.

## 7. REFERENCES

- [1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.
- [2] R.E. Bellman. *Dynamic programming*. Courier Dover Publications, 2003.
- [3] D.S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.
- [4] Aurélie Beynier and Abdel-Ilh Mouaddib. An iterative algorithm for solving constrained decentralized Markov decision processes. In *Proceedings of AAAI*, 2006.
- [5] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *IJCAI*, pages 478–485, 1999.
- [6] D.B. Chalfin, S. Trzeciak, A. Likourezos, B.M. Baumann, R.P. Dellinger, et al. Impact of delayed transfer of critically ill patients from the emergency department to the intensive care unit\*. *Critical care medicine*, 35(6):1477–1483, 2007.
- [7] P. Cramton, Y. Shoham, and R. Steinberg. *Introduction to combinatorial auctions*. MIT Press, 2006.
- [8] D.A. Dolgov and E.H. Durfee. Resource allocation among agents with MDP-induced preferences. *Journal of Artificial Intelligence Research*, 27(1):505–549, 2006.
- [9] C.V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity

- analysis. *Journal of Artificial Intelligence Research*, 22(1):143–174, 2004.
- [10] Thomas Keller and Patrick Eyerich. PROST: Probabilistic planning based on UCT. In *Proc. ICAPS*, 2012.
  - [11] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.
  - [12] S. Koenig, C. Tovey, X. Zheng, and I. Sungur. Sequential bundle-bid single-sale auction algorithms for decentralized control. In *Proceedings of the international joint conference on artificial intelligence*, pages 1359–1365, 2007.
  - [13] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled Markov decision processes. In *Proceedings AAAI*, pages 165–172, 1998.
  - [14] N. Nisan. Bidding and allocation in combinatorial auctions. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 1–12. ACM, 2000.
  - [15] L.G.N. Nunes, S.V. de Carvalho, and R.C.M. Rodrigues. Markov decision process applied to the control of hospital elective admissions. *Artificial intelligence in medicine*, 47(2):159–171, 2009.
  - [16] D.C. Parkes. Online mechanisms. *Algorithmic Game Theory*, ed. N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, pages 411–439, 2007.
  - [17] T.O. Paulussen, N.R. Jennings, K.S. Decker, and A. Heinzl. Distributed patient scheduling in hospitals. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1224–1232. Citeseer, 2003.
  - [18] T.O. Paulussen, A. Zoller, F. Rothlauf, A. Heinzl, L. Braubach, A. Pokahr, and W. Lamersdorf. Agent-based patient scheduling in hospitals. *Multiagent Engineering*, pages 255–275, 2006.
  - [19] S.J. Rassenti, V.L. Smith, and R.L. Bulfin. A combinatorial auction mechanism for airport time slot allocation. *The Bell Journal of Economics*, pages 402–417, 1982.
  - [20] J. Wu and E.H. Durfee. Sequential resource allocation in multiagent systems with uncertainties. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 114. ACM, 2007.



# Bayesian Reinforcement Learning for Multiagent Systems with State Uncertainty

Christopher Amato  
CSAIL  
MIT  
Cambridge, MA 02139  
camato@csail.mit.edu

Frans A. Oliehoek  
Department of Knowledge Engineering  
Maastricht University  
Maastricht, The Netherlands  
frans.oliehoek@maastrichtuniversity.nl

## ABSTRACT

Bayesian methods for reinforcement learning are promising because they allow model uncertainty to be considered explicitly and offer a principled way of dealing with the exploration/exploitation tradeoff. However, for multiagent systems there have been few such approaches, and none of them apply to problems with state uncertainty. In this paper we fill this gap by proposing two frameworks for Bayesian RL for multiagent systems with state uncertainty. This includes a multiagent POMDP model where a team of agents operates in a centralized fashion, but has uncertainty about the model of the environment. We also consider a best response model in which each agent also has uncertainty over the policies of the other agents. In each case, we seek to learn the appropriate models while acting in an online fashion. We transform the resulting problem into a planning problem and prove bounds on the solution quality in different situations. We demonstrate our methods using sample-based planning in several domains with varying levels of uncertainty about the model and the other agents' policies. Experimental results show that overall, the approach is able to significantly decrease uncertainty and increase value when compared to initial models and policies.

## 1. INTRODUCTION

In recent years Bayesian reinforcement learning (RL) techniques have received increased attention. Bayesian methods are promising in that, in principle, they give an optimal exploration/exploitation trade-off with respect to the prior belief. In many real-world situations, the true model may not be known, but a prior can be expressed over a class of possible models. The uncertainty over models can be explicitly considered to choose actions that will maximize expected value over models, reducing uncertainty as needed to improve performance. In practice, experience and domain knowledge can be used to construct an informed prior which can be improved online while acting in the environment.

Unfortunately, in multiagent systems, only a few Bayesian RL methods have been considered. For example, the Bayesian RL framework has been used in stochastic games [7] and factored Markov decision processes (MDPs) [28]. While either model is intractable to solve optimally, both approaches generate approximate solutions (based on the value of perfect information) which perform well in practice. Both approaches

also assume the state of the problem is fully observable (or can be decomposed into fully observable components). This is a common assumption to make, but many real-world problems have partial observability due to noisy or inadequate sensors as well as a lack of communication with the other agents. To the best of our knowledge, no approaches have been proposed that can model and solve problems with partial observability. In fact, while planning in partially observable domains has had success [4, 5, 6, 13, 14, 16, 17, 19], very few multiagent RL approaches of any kind consider partially observable domains (notable exceptions, e.g., [1, 8, 20, 34]).

In this work, we propose two approaches for Bayesian learning in multiagent systems with state uncertainty: a centralized perspective using multiagent partially observable MDPs (MPOMDPs) and a decentralized perspective using best response models. In the centralized perspective, all agents share the same partially observable view of the world and can coordinate on their actions, but have uncertainty about the underlying environment. To model this problem, we extend the Bayes Adaptive POMDP (BA-POMDP) model [23, 24], which represents beliefs over possible model parameters using Dirichlet distributions. A BA-POMDP can be characterized and solved as a (possibly infinite state) POMDP. Because an MPOMDP can be mapped to a POMDP, our centralized approach can be converted into and solved as a BA-POMDP. This approach learns policies and the underlying model while acting, but makes strong assumptions about centralization of viewpoints and decisions.

As an alternative, we consider a decentralized perspective. We explore two different models that assume 1) all other agents' policies are known and fixed, 2) all other agent policies are fixed, but unknown. In both scenarios, we assume there is uncertainty about the underlying environment model. To represent and solve these models we show how the BA-POMDP approach can be extended using appropriate distributions over other agent policies. In both the centralized and decentralized perspectives, we propose a Bayesian approach to online learning which represents the initial model and the initial policies for the other agents using priors and updates probability distributions over these models as the agent acts in the real world. In many real-world scenarios, these approaches should be able to quickly learn a high-quality policy while acting online.

In Section 2, we describe the POMDP, MPOMDP and Dec-POMDP [6] models as well as summarize the BA-POMDP approach. We then introduce the BA-MPOMDP model in Section 3 and discuss the theoretical results that transfer from the BA-POMDP case, allowing solution quality to be

---

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013)*, held in conjunction with *AAMAS*, May 2013, St. Paul, Minnesota, USA.



bounded when solving a finite approximation. In Section 4, we describe the best-response models and extend theoretical results to include uncertainty over the other agents’ policies. In Section 5, we present proof-of-concept experimental results showing that model uncertainty and solution quality can be improved over a small number of learning episodes when compared to priors over models and policies. We discuss related work in Section 6 and conclude in Section 7.

## 2. BACKGROUND

This section provides a concise description of the relevant frameworks for multiagent planning under uncertainty as well as the previous work on Bayesian RL for POMDPs.

### 2.1 POMDPs, MPOMDPs, & Dec-POMDPs

Dec-POMDPs form a framework for multiagent planning under uncertainty. We will reserve this name for the setting where there is no explicit communication (e.g., no sharing of observations) between agents. This means that each agent will act based only on its individual observations. Formally, a Dec-POMDP is a tuple  $\langle I, S, \{A_i\}, T, R, \{Z_i\}, O, h \rangle$  with:

- $I$ , a finite set of agents;
- $S$ , a finite set of states with designated initial state distribution  $b_0$ ;
- $A_i$ , a finite set of actions for each agent,  $i$ ;
- $T$ , a set of transition probabilities:  $T^{sas'} = \Pr(s'|s, \vec{a})$ , the probability of transitioning from state  $s$  to  $s'$  when the set of actions  $\vec{a}$  are taken by the agents;
- $R$ , a reward function:  $R(s, \vec{a})$ , the immediate reward for being in state  $s$  and taking the set of actions  $\vec{a}$ ;
- $Z_i$ , a finite set of observations for each agent,  $i$ ;
- $O$ , a set of observation probabilities:  $O^{\vec{a}s'z} = \Pr(\vec{z}|\vec{a}, s')$ , the probability of seeing the set of observations  $\vec{z}$  given the set of actions  $\vec{a}$  was taken which results in state  $s'$ ;
- $h$ , the horizon.

When agents are permitted to have different reward functions, this model becomes the partially observable stochastic game (POSG)[14]. Alternatively, it is possible to consider the multiagent setting where the agents are allowed to share their individual observations. In this case, we will restrict ourselves to the setting where such communication is free of noise, costs and delays and call the resulting model a multiagent POMDP (MPOMDP). Thus, an MPOMDP is a Dec-POMDP with the additional assumption of explicit communication.

A POMDP [15] can be seen as the special case of a Dec-POMDP with just one agent. Also, an MPOMDP can be reduced to a special type of POMDP in which there is a single centralized controller that takes joint actions and receives joint observations [22].

Most research concerning these models has considered the task of *planning*: given a full specification of the model, determine an optimal (joint) policy (e.g., [6, 14]). However, in many real-world applications, the model is not (perfectly) known in advance, which means that the agents have to learn about their environment during execution. This is the task considered in (multiagent) *reinforcement learning (RL)* [27].

### 2.2 Bayesian RL for POMDPs

A fundamental problem in RL is that it is difficult to decide whether to try new actions in order to learn about the environment, or to exploit the current knowledge about

the rewards and effects of different actions. In recent years, Bayesian RL methods have become popular because they potentially can provide a principled solution to this exploration/exploitation trade-off [9, 11, 12, 21, 30].

In particular, we consider the framework of Bayes-Adaptive POMDPs [23, 24]. This framework utilizes Dirichlet distributions to model uncertainty over transitions,  $T^{sas'}$ , and observations,  $O^{as'z}$  (typically assuming the reward function is chosen by the designer and thus known). In particular, if the agent could observe both states and observations, it could maintain vectors  $\phi$  and  $\psi$  of counts for transitions and observations respectively. That is,  $\phi_{ss'}$  is the transition count representing the number of times state  $s'$  resulted from taking action  $a$  in state  $s$  and  $\psi_{s'z}$  is the observation count representing the number of times observation  $z$  was seen after taking action  $a$  and transitioning to state  $s'$ .

While the agent cannot observe the states and has uncertainty about the actual count vectors, this uncertainty can be represented using the regular POMDP formalism. That is, the count vectors are included as part of the hidden state of a special POMDP, called BA-POMDP. Formally, a BA-POMDP is a tuple  $\langle S_{BP}, A, T_{BP}, R_{BP}, Z, O_{BP}, h \rangle$  with

- $S_{BP}$ , the set of states  $S_{BP} = S \times \mathcal{T} \times \mathcal{O}$ ;
- $A$ , a finite set of actions;
- $T_{BP}$ , a set of state transition probabilities;
- $R_{BP}$ , a reward function;
- $Z$ , a finite set of observations;
- $O_{BP}$ , a set of observation probabilities;
- $h$ , the horizon.

We discuss these components in more detail below.

First, we point out that actions and observations remain the same as in case there was no uncertainty about the transition and observation function (i.e., the same as in the regular POMDP). However, as mentioned, the state of the BA-POMDP now includes the Dirichlet parameters:  $s_{BP} = \langle s, \phi, \psi \rangle$  and the set of states  $S_{BP} = S \times \mathcal{T} \times \mathcal{O}$  where  $\mathcal{T} = \{\phi \in \mathbb{N}^{|S||A||S|}\}$  is the space of all possible transition parameters where each state action pair is visited at least once. Similarly  $\mathcal{O} = \{\psi \in \mathbb{N}^{|S||A||Z|}\}$  is the space of all possible observation parameters.<sup>1</sup>

Given a pair of count vectors  $\phi, \psi$ , we can define the expected transition and observation probabilities as:

$$T_{\phi}^{sas'} = \mathbf{E}[T^{sas'} | \phi] = \frac{\phi_{ss'}}{N_{\phi}^{sa}}, \quad O_{\psi}^{as'z} = \mathbf{E}[O^{as'z} | \psi] = \frac{\psi_{s'z}}{N_{\psi}^{as'}},$$

where  $N_{\phi}^{sa} = \sum_{s''} \phi_{ss''}$ , and  $N_{\psi}^{as'} = \sum_{z'} \psi_{s'z'}$ .

Remember that these count vectors are not observed by the agent, since that would require observations of the state. The agent can only maintain belief over these count vectors. Still, when interacting with the environment, *the ratio of the true—but unknown—count vectors will converge to coincide with the true transition and observation probabilities in expectation*. It is important to realize, however, that this convergence of count vector ratios does not directly imply learnability by the agent: even though the ratio of the count vectors specified by the true hidden state will converge, *the agent’s belief over count vectors might not*.

The expected transition and observation probabilities can be used to define the transition and observation model of

<sup>1</sup>Note that at least one of the counts per Dirichlet parameter vector needs to be non-zero.

the BA-POMDP. In particular, the transition probabilities  $P((s', \phi', \psi') | (s, \phi, \psi), a)$  can be defined using a vector  $\delta_{ss'}^a$  which is 1 at the index of  $a, s$  and  $s'$  and 0 otherwise. Similarly, for observations, we define  $\delta_{s'z}^a$  to be a vector that has value 1 at the index  $a, s'$  and  $z$  and 0 otherwise. The resulting transition and observation models are

$$T_{BP}((s, \phi, \psi), a, (s', \phi', \psi')) = \begin{cases} T_{\phi}^{sas'} O_{\psi}^{as'z} & \text{if } \phi' = \phi + \delta_{ss'}^a \text{ and } \psi' = \psi + \delta_{s'z}^a \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

$$O_{BP}((s, \phi, \psi), a, (s', \phi', \psi'), z) = \begin{cases} 1 & \text{if } \psi' = \psi + \delta_{s'z}^a \\ 0 & \text{otherwise} \end{cases} \quad (2.2)$$

Note that the observation model is now deterministic as the observation expectations are in the transition function: in  $T_{BP}$ ,  $z$  is determined by  $\psi' - \psi = \delta_{s'z}^a$ .

The reward model remains the same (since it is assumed to be known),  $R_{BP}((s, \phi, \psi), a) = R(s, a)$ . An initial state distribution  $b_0$  as well as initial count vectors  $\phi_0$  and  $\psi_0$  are also assumed.

Notice that the BA-POMDP model described above has an infinite number of states if we allow ourselves to consider all possible count vectors for transitions and observations. This infinite state representation makes performing belief updates and solving the BA-POMDP impossible without sampling. Fortunately, Ross et al. prove that the solution quality can be bounded when considering a finite number of count vectors [24]:

**THEOREM 1.** *Given any BA-POMDP,  $\epsilon > 0$  and horizon  $h$ , it is possible to construct a finite POMDP by removing states with count vectors  $\phi, \psi$  that have  $N_{\phi}^{s,a} > N_S^{\epsilon}$  or  $N_{\psi}^{a,s'} > N_Z^{\epsilon}$  for suitable thresholds  $N_S^{\epsilon}, N_Z^{\epsilon}$ .*

However, even with finite count vectors, the BA-POMDP formulation remains very large, necessitating sample-based planning approaches to provide solutions. For example, the approach of Ross et al. used an online planning approach that determined the best action to take at a given belief by performing dynamic programming using a simulator for a small horizon, updating the belief (approximately) after taking that action and receiving an observation and continuing this process until the end of the of the problem is reached. Different methods for updating the belief were used such as Monte Carlo sampling and heuristics for limiting the number of belief states considered.

### 3. BA-MPOMDPS

In this section, we extend the BA-POMDP to the multi-agent setting. As mentioned in Section 2 it is well-known that, under the assumption of instantaneous communication without noise or costs, a Dec-POMDP can be reduced to an MPOMDP, which can then be treated as a single agent model. In the same way, for a multiagent setting in which there are uncertainties about the model, we propose to treat the problem as a *BA-MPOMDP*. A BA-MPOMDP can be seen as a BA-POMDP where the actions are joint actions and the observations are joint observations. Due to this correspondence, the theoretical results related to BA-POMDPs also apply to the BA-MPOMDP model. The BA-MPOMDP model is in principle applicable in any multiagent RL setting where there is such instantaneous communication.

### 3.1 The Model

Formally, a BA-MPOMDP is a tuple  $\langle I, S_{BM}, \{A_i\}, T_{BM}, R_{BM}, \{Z_i\}, O_{BM}, h \rangle$  with:

- $I$ , a finite set of agents;
- $S_{BM}$ , states  $S \times \mathcal{T} \times \mathcal{O}$  with initial state distribution  $b_0$  and initial counts  $\phi_0$  and  $\psi_0$ ;
- $A_i$ , a finite set of actions for each agent,  $i$ ;
- $T_{BM}$ , a set of state transition probabilities as defined below;
- $R_{BM}$ , a reward function as defined below;
- $Z_i$ , a finite set of observations for each agent,  $i$ ;
- $O_{BM}$ , a set of observation probabilities as defined below;
- $h$ , the horizon.

The framework is very similar to the POMDP case, but actions,  $a$ , now become joint actions,  $\vec{a}$ , and observations,  $z$ 's, become joint observations,  $\vec{z}$ . This means that a BA-MPOMDP is specified using count vectors  $\phi_{ss'}^{\vec{a}}$  and  $\psi_{s'z}^{\vec{a}}$ , from their respective spaces:  $\mathcal{T} = \{\phi \in \mathbb{N}^{|S||A||S|}\}$  is the space of all possible transition counts and similarly  $\mathcal{O}$  is the space of all possible observation parameters  $\mathcal{O} = \{\psi \in \mathbb{N}^{|S||A||Z|}\}$ .

Each pair of vectors has associated expected transition and observation probabilities  $T_{\phi}^{sas'} = \phi_{ss'}^{\vec{a}} / N_{\phi}^{s,\vec{a}}$ ,  $O_{\psi}^{as'z} = \psi_{s'z}^{\vec{a}} / N_{\psi}^{a,s'}$ . These in turn are used to specify the transition and observation model:

$$T_{BM}((s, \phi, \psi), \vec{a}, (s', \phi', \psi')) = \begin{cases} T_{\phi}^{sas'} O_{\psi}^{as'z} & \text{if } \phi' = \phi + \delta_{ss'}^{\vec{a}} \text{ and } \psi' = \psi + \delta_{s'z}^{\vec{a}} \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

$$O_{BM}((s, \phi, \psi), \vec{a}, (s', \phi', \psi'), z) = \begin{cases} 1 & \text{if } \phi' = \phi + \delta_{ss'}^{\vec{a}} \text{ and } \psi' = \psi + \delta_{s'z}^{\vec{a}} \\ 0 & \text{otherwise} \end{cases} \quad (3.2)$$

The reward model is given as  $R_{BM}((s, \phi, \psi), \vec{a}) = R(s, \vec{a})$

### 3.2 Solution Methods

BA-MPOMDP formalism also suffers from an infinite state space, since there can be infinitely many count vectors. However, also in the multiagent case, it is possible to create a finite approximate model

**THEOREM 2.** *Given any BA-MPOMDP,  $\epsilon > 0$  and horizon  $h$ , it is possible to construct a finite POMDP by removing states with count vectors  $\phi, \psi$  that have  $N_{\phi}^{s,a} > N_S^{\epsilon}$  or  $N_{\psi}^{a,s'} > N_Z^{\epsilon}$  for suitable thresholds  $N_S^{\epsilon}, N_Z^{\epsilon}$  that depend linearly on the number of states and joint observations, respectively.*

**PROOF.** Since an MPOMDP is a special case of POMDP, the BA-MPOMDP is a special case of BA-POMDP, thus this follows directly from Theorem 1.  $\square$

While this result is straightforward, the interesting part is that while  $N_Z^{\epsilon}$  does depend on the number of joint observations, the count thresholds do not have any dependence on the number of joint actions. Therefore, for problems with few observations per agent, constructing this approximation might be feasible, even if there are many actions.

Of course, in general even a finite approximation of a BA-MPOMDP is intractable to solve optimally. Fortunately, online sample-based planning approaches in principle apply to this setting too. That is, the team of agents could perform online planning by considering a small finite horizon. After taking the joint action that resulted from the online planning phase, the environment makes a transition, the agents get observations, these observations are synchronized via communication and every agent computes the new ‘joint belief’. Then this process repeats, etc.

The actual planning could take place in a number of ways: one agent could be designated the planner, which would require this agent to broadcast the computed joint action. Alternatively, each agent can in parallel perform an identical planning process (by, in the case of randomized planning, syncing the random number generators). Then each agent will compute the same joint action and execute its component. An interesting direction of future work is whether the planning itself can be done more effectively by distributing the task over the agents.

However, in the above, there is an additional bottleneck compared to the BA-POMDP: the number of joint actions and joint observations is exponential in the number of agents.

## 4. BA-BRM

In many real-world scenarios, instantaneous, noise and cost free communication may not be possible or practical. Similarly, in competitive domains, this type of communication often does not make sense. As a result, agents must learn during execution based solely on their own local information. In this section, we describe different way of applying Bayesian RL techniques in multiagent systems by giving a subjective description of the problem. That is, we describe the problem from a single agent’s perspective by defining its *best-response model* (BRM). We propose a Bayesian approach to online learning which represents the initial model and the initial policies for the other agents using priors and updates probability distributions over these models as the agent acts in the real world.

### 4.1 Best-Response Models

When making a subjective model from the perspective of a given agent, there are a number of assumptions one can make about uncertainty: First, we can assume that the agent is uncertain about only the transition and observation functions, but certain about the policies of the other agents. Second, we can assume that it also is uncertain about the other agents’ policies, but that these are fixed. Finally, we can also assume that the other agents in turn are adaptive. We discuss these different assumptions in the subsections below, focusing on the first two. It is also possible that the agent is certain about  $T, O$  but uncertain about the policy of other agent. For such cases, the model we introduce for the second setting applies. Alternatively, such a setting might also be modeled using an I-POMDP (see Section 6).

### 4.2 Transition & Observation Uncertainty

Under the first assumption, agent  $i$  is uncertain about the transition and observation functions, but knows the (deterministic) policy  $\pi_j$  for all other agents.<sup>2</sup> Since those  $\pi_j$

<sup>2</sup>These policies can be represented as look-up tables or be computational procedures themselves. We do make the re-

map observation histories  $\omega_j^t = (z_j^1, \dots, z_j^t)$  to actions  $a_j^t$  at time  $t$ , agent  $i$  can model this situation using an augmented POMDP [16], which we will call the *best-response model*, in which states are tuples  $\langle s, \vec{\omega}_{-i}^t \rangle$  of nominal states  $s$  and observation histories of other agents  $\vec{\omega}_{-i}^t = \langle \vec{\omega}_1^t \dots \vec{\omega}_{i-1}^t \vec{\omega}_{i+1}^t \dots \vec{\omega}_n^t \rangle$ . Given that this is a POMDP, we can incorporate uncertainty about transition and observation models by transforming it to its Bayes adaptive variant.

Formally, we define a *BA-BRM* as a tuple  $\langle S_{BB}, A_i, T_{BB}, R_{BB}, Z_i, O_{BB}, h \rangle$  where

- $A_i, Z_i$  are the sets of actions and observations of agent  $i$ ;
- $S_{BB}$  is the set of states  $\langle s, \vec{\omega}_{-i}^t, \phi, \psi \rangle$  where  $\phi$  is the vector of counts  $\phi_{s\vec{\omega}s'\vec{\omega}'}$  and  $\psi$  is the vector of counts  $\psi_{s'\vec{\omega}'z}$ ;
- $T_{BB}$  is the transition function (see below);
- $R_{BB}$  is the reward function defined as  $R_{BB}(s, \vec{\omega}_{-i}^t, \phi, \psi, a_i) = \sum_{a_{-i}} \pi_{-i}(a_{-i} | \vec{\omega}_{-i}^t) R(s, a_i, a_{-i})$ ;
- $O_{BB}$  is the observation function. As earlier  $O_{BB}((s, \vec{\omega}_{-i}^t, \phi, \psi), a_i, (s', \vec{\omega}_{-i}^{t+1}, \phi', \psi'))$  is 1 iff the count vectors add correctly;
- $h$  is the horizon.

The transition function is defined as

$$T_{BB}((s, \vec{\omega}_{-i}^t, \phi, \psi), a_i, (s', \vec{\omega}_{-i}^{t+1}, \phi', \psi')) = \begin{cases} T_{\phi}^{s\vec{\omega}as'\vec{\omega}'} O_{\psi}^{s'\vec{\omega}'az} & , \phi' = \phi + \delta_{s\vec{\omega}s'\vec{\omega}'}^a, \psi' = \psi + \delta_{s'\vec{\omega}'z}^a \\ 0 & \text{otherwise.} \end{cases}$$

where we dropped sub- and superscripts that are clear from context. In this equation, the expected transition and observation functions are defined as

$$T_{\phi}^{s\vec{\omega}as'\vec{\omega}'} = \frac{\phi_{s\vec{\omega}s'\vec{\omega}'}}{\sum_{s''\vec{\omega}''} \phi_{s\vec{\omega}as''\vec{\omega}''}} \quad (4.1)$$

$$O_{\psi}^{s'\vec{\omega}'az} = \frac{\psi_{s'\vec{\omega}'z}}{\sum_{z'} \psi_{s'\vec{\omega}'z'}} \quad (4.2)$$

The former count ratio converges to  $P(s', z_{-i} | s, a_i, a_{-i})$  (where  $a_{-i}$  is the action profile specified by  $\pi_{-i}$  for  $\vec{\omega}_{-i}$ ) which is the true probability of  $s', \vec{\omega}_{-i}^{t+1}$  given  $s, \vec{\omega}_{-i}^t, a_i$  and  $\pi_{-i}$  for the true, but unknown, count vector  $\phi^*$ . The latter ratio, for  $\psi^*$ , converges to  $P(z_i | a_i, \vec{a}_{-i}, s')$ , the true probability of receiving observation  $z_i$  given  $s', \vec{\omega}_{-i}^{t+1}, a_i$  and  $\pi_{-i}$ .

We point out that for this formulation, all the BA-POMDP theory holds even with the inclusion of other agent histories as part of the state information. Nevertheless, this model assumes the policies of other agents are *fixed, known* and *deterministic*. This latter assumption can be removed. When the other agents use a *stochastic policy*, those  $\pi_j$  map *action-observation histories*  $h_j^t = (a_j^0 z_j^1 \dots a_j^{t-1} z_j^t)$  to actions  $a_j^t$ . For this case, we can trivially adapt the BA-BRM by replacing the  $\vec{\omega}_{-i}^t$  by  $\vec{h}_{-i}^t$ .

### 4.3 Policy Uncertainty

The substitution of  $\vec{\omega}_{-i}^t$  by  $\vec{h}_{-i}^t$  for stochastic policies brings an interesting insight: two subsequent states  $(s, \vec{h}_{-i}^t)$  and  $(s', \vec{h}_{-i}^{t+1})$  specify what actions the other agents took in the previous step (since those are specified in the action-observation histories). As such, counting these transitions, in general

striction, however, that they do not depend on the policy followed by agent  $i$ .

may also allow us to learn about the policies of others if we have uncertainty about them.

That is, the expected transition can be calculated as

$$T_{\phi}^{shas'h'} = \frac{\phi_{shs'h'}^a}{\sum_{s'',h''} \phi_{shs''h''}^a}$$

with  $\vec{h} = \vec{h}^t$  and  $\vec{h}' = \vec{h}^{t+1}$ , and therefore the true count vector ratio will converge to the probability

$$\begin{aligned} P(s^{t+1}, \vec{h}^{t+1} | s^t, \vec{h}^t, a_i^t) &= P(s^{t+1}, (\vec{h}_{-i}^t, \vec{a}_{-i}^t, \vec{z}_{-i}^{t+1}) | s^t, \vec{h}_{-i}^t, a_i^t) \\ &= \pi_{-i}(\vec{a}_{-i}^t | \vec{h}_{-i}^t) P(s^{t+1} | s^t, \vec{a}^t) \sum_{z_i^{t+1}} P(\vec{z}^{t+1} | s^{t+1}, \vec{a}^t) \end{aligned} \quad (4.3)$$

Note that we only need to consider  $h'$  if it includes  $h$ . This value includes the probabilities induced by the policies of the other agents, allowing uncertainty with respect to the other agents' policies to also be represented.

An interesting aspect of this formulation is that it can be used to bound the loss of computing a best response to one particular policy while in fact the agent uses a different one. To show this, we assume that there is a single other agent and that for two policies  $\pi_j^x, \pi_j^y$  of agent  $j$  we have that

$$\forall a_j h_j \quad |\pi_j^x(a_j | h_j) - \pi_j^y(a_j | h_j)| \leq \epsilon. \quad (4.4)$$

Assume that  $\pi_j^x$  is the true policy of agent  $j$ . In that case the  $\phi$  count vectors converge to some  $\phi_x^*$  that satisfies

$$\forall shas'h' \quad \frac{\phi_{shs'h'}^{a,x}}{\mathcal{N}_{sh}^{a,x}} = \pi_j^x(a_j | h_j^t) P(s^{t+1}, z_j^{t+1} | s^t, a_i^t, a_j^t)$$

(where  $\mathcal{N}_{sh}^a$  denotes the normalization constant) while, when  $\pi_j^y$  is the true policy, these count ratios converge to

$$\pi_j^y(a_j | h_j^t) P(s^{t+1}, z_j^{t+1} | s^t, a_i^t, a_j^t) = \frac{\phi_{shs'h'}^{a,y}}{\mathcal{N}_{sh}^{a,y}}$$

Additionally, we have that, independently of  $\pi_j$  the policy of the other agent, the  $\psi$  count ratios of the true hidden state converge to

$$P(z_i^{t+1} | a_i^t, a_j^t, s^{t+1}, z_j^{t+1}) = \frac{\psi_{s'h',z}^{a}}{\mathcal{N}_{s'h'}^{a}}$$

Note that here  $\vec{h}'$  specifies both  $a_j^t$  and  $z_j^{t+1}$ .

It follows that, upon convergence of these ratios, we have

$$\begin{aligned} &\left| \frac{\phi_{shs'h'}^{a,x}}{\mathcal{N}_{sh}^{a,x}} \frac{\psi_{s'h',z}^{a}}{\mathcal{N}_{s'h'}^{a}} - \frac{\phi_{shs'h'}^{a,y}}{\mathcal{N}_{sh}^{a,y}} \frac{\psi_{s'h',z}^{a}}{\mathcal{N}_{s'h'}^{a}} \right| \\ &= |(\pi_j^x(a_j | h_j) - \pi_j^y(a_j | h_j)) P(s^{t+1}, z_j^{t+1} | s^t, a_i^t, a_j^t)| \\ &\quad P(z_i^{t+1} | a_i^t, a_j^t, s^{t+1}, z_j^{t+1}) \\ &\leq \epsilon P(s^{t+1}, z_i^{t+1}, z_j^{t+1} | s^t, a_i^t, a_j^t) \end{aligned}$$

Moreover,

$$\begin{aligned} \sum_s \sum_z \left| \frac{\phi_{shs'h'}^{a,x}}{\mathcal{N}_{sh}^{a,x}} \frac{\psi_{s'h',z}^{a}}{\mathcal{N}_{s'h'}^{a}} - \frac{\phi_{shs'h'}^{a,y}}{\mathcal{N}_{sh}^{a,y}} \frac{\psi_{s'h',z}^{a}}{\mathcal{N}_{s'h'}^{a}} \right| &\leq \\ \sum_s \sum_z \epsilon P(s^{t+1}, z_i^{t+1}, z_j^{t+1} | s^t, a_i^t, a_j^t) &= \epsilon \end{aligned} \quad (4.5)$$

That is, given that the difference between two policies is bounded, the difference between count vector ratios, and thus expected transition probabilities that they will induce

is bounded as well. This can subsequently be used to bound the loss in value when optimizing against a wrong policy.

**THEOREM 3.** *Given  $\phi_x^*, \phi_y^*$  and  $\psi^*$ , the converged count vectors corresponding to two policies  $\pi_j^x, \pi_j^y$  of agent  $j$  that satisfy (4.4), then, for all stages-to-go  $t$ , then for any  $t$ -steps-to-go policy for agent  $i$ , the associated values are bounded:*

$$\max_{s \in S} |\alpha_t(s, \phi_x^*, \psi^*) - \alpha_t(s, \phi_y^*, \psi^*)| \leq \frac{\epsilon(\gamma - \gamma^t) \|R\|_{\infty}}{(1 - \gamma)^2} \quad (4.6)$$

**PROOF.** Here,  $\|R\|_{\infty}$  is the reward with greatest magnitude and  $\gamma$  is the discount factor. The proof is given in the appendix.  $\square$

The implication of this theorem is that if we compute a best-response against some policy  $\pi_j^x$  which differs from  $\pi_j^y$ , the true policy used by agent  $j$ , by at most  $\epsilon$ , then that loss in value is bounded by (4.6). While this relates to bounds for model equivalence [33], no bounds on the loss in value for different policies of the other agent have been proposed. Also, we expect that these bound could have big implications for work on influence based abstraction [31], and, in particular, using approximate influences.

Finally, we point out that, when other agents are adaptive, the assumption of a unknown, but fixed policy is violated. In fact there are inherent limits to what can be learned by Bayesian learners that perform a best-response [32]. Nevertheless, methods such as Q-learning have been shown to be effective in such domains [25, 29]. We expect that it might be possible to deal with this issue by, for instance, performing discounting of counts while learning during execution. This is a fertile area for future research.

## 4.4 Solving BA-BRMs

BA-BRMs have an intractable (infinite) number of parameters, but again, the theory from [23] applies such that we can ensure that a solution that is boundedly optimal can be generated using a finite model.

**THEOREM 4.** *Given any BA-MPOMDP,  $\epsilon > 0$  and horizon  $h$ , it is possible to construct a finite POMDP by removing states with count vectors  $\phi, \psi$  that have  $N_{\phi}^{s,a} > N_S^{\epsilon}$  or  $N_{\psi}^{a,s'} > N_Z^{\epsilon}$  for suitable thresholds  $N_S^{\epsilon}, N_Z^{\epsilon}$  that depend linearly on the number of augmented states and individual observations, respectively.*

**PROOF.** Again, since a BRM is a special case of POMDP, the BA-BRM is a special case of BA-POMDP, thus this follows directly from Theorem 1.  $\square$

The result itself is straightforward. In this case, however,  $N_Z^{\epsilon}$  only depends on the size of the individual observation set. However, this comes at a cost, since  $N_{\psi}^{a,s'}$  now depends linearly on the number of *augmented* states, which is  $O(|S|(|A_j||Z_j|)^{h(n-1)})$ . In case of a single other agent, this means that complexity dependence on joint observations in the BA-MPOMDP is replaced by an exponential dependence on the horizon.

Even in this case, the model will often be large and difficult to learn. Sample-based planning can also be used in this scenario by transforming the BA-BRM into a BA-POMDP and solving it. The number of states may become large, but the number of number of actions in the BA-BRM remains the same as in the original Dec-POMDP model (unlike the

BA-MPOMDP). Communication can also be incorporated to coordinate the learning and improve its efficiency.

Prior distributions over environment and agent models can be represented as initial count vectors. As is clear from (4.3), the  $\phi$  ratios correspond to (should converge to) the true probability  $\pi_{-i}(\vec{a}_{-i}^t | \vec{h}_{-i}^{t-1}) P(s' | s, \vec{a}) \sum_{z_i} P(z_i | s', \vec{a})$ . If these probabilities can be estimated, the count vectors can be set to ratios representing this quantity. Then, the confidence in this estimation can be reflected in a scaling factor of the various counts. In this way, different aspects of the agent and environment models can have different parameters and confidence based on knowledge of the problem. In the absence of domain knowledge a uniform prior with small counts can be utilized.

## 5. EXPERIMENTAL EVALUATION

We performed a preliminary empirical evaluation of the BA-MPOMDP and BA-BRM models.

### 5.1 Sample-Based Planning

To test performance of the different models, we implemented a simulation of agents that interact with an environment over a number of episodes ( $N_{episodes}$ ). Importantly, at the end of each episode, the belief over states is reset to the initial belief, but the belief over count vectors is maintained. That way, the agents learn across all the episodes.

The agents use an on-line sample-based planner to act: in each stage of each episode, the agent(s) perform sample-based planning in order to select a (joint) action. This action is subsequently executed, a transition and observation is sampled, the agents update their (joint) beliefs, and a new round of online planning is initiated, etc. The beliefs are represented using a particle filter (with  $N_{particles}$  particles).

As the sample-based planner, we use Monte Carlo planning: the expected value for each (joint) action is evaluated using a number ( $N_{samples}$ ) of Monte Carlo rollouts (i.e., using a random (joint) action selection) up to a particular lookahead planning horizon (which can be shorter than  $h$ ). This planner also uses particle-based belief representations.<sup>3</sup>

### 5.2 Experimental Setup

We evaluate our BA-MPOMDP and BA-BRM approaches by performing online learning using the common decentralized tiger benchmark [16] for horizon 3. In this cooperative problem, two agents have the choice of listening or opening one of two doors. If both agents listen, they each hear a noisy signal of the tiger's location (0.85 probability of hearing the correct location). If either agent opens a door, there is a shared penalty for opening the door with the tiger and a shared reward for opening the other door (which has treasure behind it). If both agents choose to open the same door, they receive a reduced penalty or a greater reward. Whenever a door is opened the tiger transitions uniformly to be behind one of the doors.

For illustration, we show only error in the observation model, but unlike Ross et al. we do not assume the transition function is known. Instead, we assume we have high confidence in the transition parameters and reflect this in the transition count vectors, ( $\phi$  were initialized as 1000 times the

true probability as discussed above). The observation priors that were used are listed in Table 1.

For the BA-BRMs, the optimal pair of (deterministic) policies was found (using [2]) and one of these policies was used as the fixed policy of the other agent. This policy represents the other agent listening until it has heard the tiger on the same side twice and then opening the appropriate door. The count vectors for the transitions were set similarly to those above (with 1000 times the true transition probabilities) and the observation count vectors were set as in Table 1.

To determine performance, we consider model error and value. Model error is calculated as a sum of the L1 distances between the correct and estimated probabilities weighted by the probability associated with the count vectors (as described in [23]). The error and value produced are averaged over a number of simulations ( $N_{simulations}$ ). Note that at the start of each simulation also the count vectors are reset, so there is no learning across simulations — these are only to determine average values. These experiments were run on a 2.5 GHz Intel i7 using a maximum of 2GB of memory.

### 5.3 Results

The errors in the observation functions for all methods are shown in Figure 1. For the BA-MPOMDP formulation, for which we have performed  $N_{simulations} = 50$  simulations with  $N_{episodes} = 50$ ,  $N_{particles} = 200$ ,  $N_{samples} = 500$ , and a lookahead horizon of two. We see that the error decreases sharply and then decreases more slowly. This error will likely continue to improve if more episodes are completed. The nonmonotonic improvement is due to randomness in the sample values and the fact that only 50 simulations were used to average the data. Value for this problem from the initial belief and count vectors is approximately -6 (the value of listening at each step) and the optimal value for this problem with known parameters is 13.015 (listening twice and then collectively opening the appropriate door if the same observation was heard twice). After 50 episodes, the value was estimated and was found to be -4.5.

For the BA-BRM experiments, we used the following parameters:  $N_{simulations} = 50$  simulations with  $N_{episodes} = 50$ ,  $N_{particles} = 100$ ,  $N_{samples} = 200$  and a lookahead horizon of two. It is worth noting that the optimal value of this version of the problem with known parameters is 5.19. Notice that this is less than in the MPOMDP case because the agents can no longer coordinate to always open the same door at the same time.

The model error for BA-BRM with a known other agent policy is also shown in Figure 1. Similar to the BA-MPOMDP case, we see a decrease in model error over as the number of episodes increases. Sampling to determine the value produces a value of approximately -3.78 with the initial count vectors and -2.03 after 50 episodes.

For BA-BRM with unknown other agent policy, the settings were:  $N_{simulations} = 10$  simulations with  $N_{episodes} = 50$ ,  $N_{particles} = 50$ ,  $N_{samples} = 50$  and a lookahead horizon of one. This reduction in samples is due to the increased time required for updating and evaluation by considering unknown actions in the other agent histories. The value produced in this model after learning is unchanged from the initial value. It is likely that increased sampling would improve both uncertainty and the resulting value, suggesting the need for more computationally efficient methods.

<sup>3</sup>When the number of reachable states was small, we used a closed-form description of beliefs to speed up planning.

joint action	joint observation	count
both listen	both correct	5
	1 correct	2
	both incorrect	1
other	all	2

action	observation	count
listen	correct	3
	incorrect	2
open	all	2

Table 1: Observation prior counts for the BA-MPOMDP (left) and BA-BRM (right).

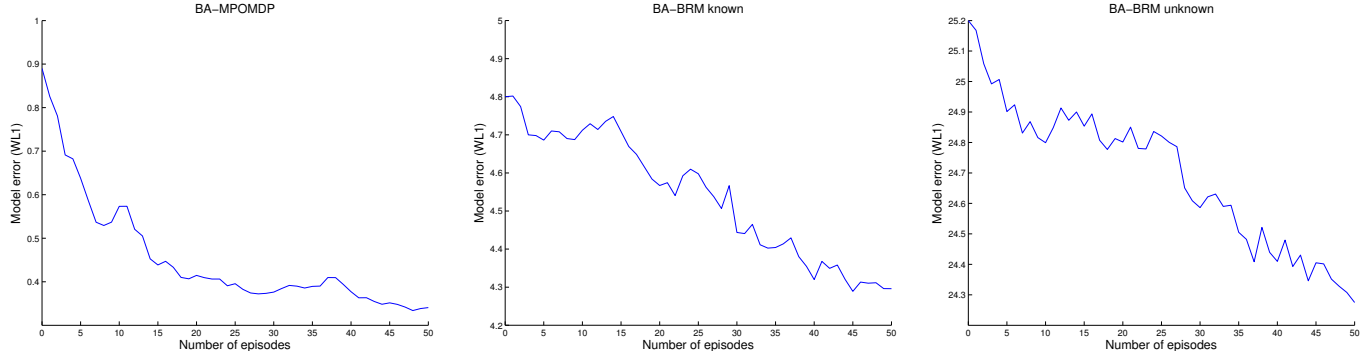


Figure 1: Model error (weighted L1) in the 2-agent tiger problem for the BA-MPOMDP and BA-BRM with known and unknown other agent policies

## 6. RELATED WORK

The BA-BRM is closely related to the framework of I-POMDPs [13, 33] in that it is also a model for describing an interactive setting from the perspective of a protagonist agent. There are a few differences, however. First, the I-POMDP treats another agent as an entity that (potentially) also reasons about the protagonist agent, but at a lower level. In contrast, the BA-BRM just considers the other agent as executing a particular policy. As such, there are no difficulties with infinite recursions of beliefs. Second, the I-POMDP does not consider uncertainty about the actual transition and observation model. However, we point out that since an I-POMDP is a special type of POMDP, it is possible to consider Bayes adaptive extensions that consider such uncertainty [18].

Other work that is out of the scope of this paper has developed other learning techniques for Dec-POMDPs. These approaches include model-free reinforcement learning methods using gradient-based methods to improve the policies [10, 20] and learning using local signals and modeling the remaining agents as noise [8]. Another approach has utilized communication and sample-based planning to generate best-response policies [3].

## 7. CONCLUSIONS

In this paper, we presented the first set of approaches for Bayesian reinforcement learning for multiagent systems with state uncertainty. We consider a centralized perspective where the team of agents is modeled as a multiagent POMDP, allowing Bayesian RL techniques from the POMDP literature to be applied. Because the centralized perspective assumes centralization or full communication between the agents, we also consider a decentralized perspective. In this perspective, we explore cases in which an agent knows the fixed policies of the others, but has uncertainty over the environment model and when an agent has uncertainty over the

policies of the fixed agents and environment models. Each of these cases reflects realistic assumptions about real-world scenarios. We present proofs bounding the solution quality under these different assumptions. Our experimental results show a proof of concept for Bayesian RL in multiagent systems with state uncertainty, demonstrating how an agent can improve model estimates and performance while acting online.

These approaches can serve as the basis for many future work directions in multiagent learning. This could include the use of (delayed or noisy) communication to allow the best response model to update parameters based on information from the other agents. Similarly, additional domain or policy assumptions could be imposed to improve scalability. For instance, with transition and observation independence [4], models of others can be represented as mappings from local states to actions and using finite-state controllers [5], parameters can be limited by the size of the controllers. We also expect more efficient solutions for these models could be generated by more sophisticated sample-based planning methods such as [26], allowing greater scalability to larger domains and a larger number of agents. Lastly, even though our experiments consisted of a cooperative domain, our approach extends to competitive models and we plan to test its effectiveness in those problems as well.

## Acknowledgements

Research supported in part by AFOSR MURI project #FA9550-091-0538 and in part by NWO CATCH project #640.005.003.

## APPENDIX

An  $\alpha$ -vector for a BA-POMDP, can be expressed as the immediate reward for the specified action  $a$  plus value of next stage vectors for some mapping  $\alpha' : \mathcal{Z} \rightarrow \Gamma_{t-1}$  [24]. Therefore, for any policy

$\pi_t$ , for all states  $s$ , we have that

$$\begin{aligned}
& |\alpha_t^{\pi_t}(s, \phi_x, \psi_x) - \alpha_t^{\pi_t}(s, \phi_y, \psi_y)| \\
&= \gamma \left| \sum_{s'} \sum_z \left[ \frac{\phi_x^{sas'}}{\mathcal{N}_x^{sa}} \frac{\psi_x^{as'z}}{\mathcal{N}_x^{as'}} (\alpha'(z)(s', \phi_x^\delta, \psi_x^\delta) - \alpha'(z)(s', \phi_y^\delta, \psi_y^\delta)) \right. \right. \\
&\quad \left. \left. - \left( \frac{\phi_y^{sas'}}{\mathcal{N}_y^{sa}} \frac{\psi_y^{as'z}}{\mathcal{N}_y^{as'}} - \frac{\phi_x^{sas'}}{\mathcal{N}_x^{sa}} \frac{\psi_x^{as'z}}{\mathcal{N}_x^{as'}} \right) \alpha'(z)(s', \phi_y^\delta, \psi_y^\delta) \right] \right| \\
&\leq \gamma \sum_{s'} \sum_z \frac{\phi_x^{sas'}}{\mathcal{N}_x^{sa}} \frac{\psi_x^{as'z}}{\mathcal{N}_x^{as'}} \left| \alpha'(z)(s', \phi_x^\delta, \psi_x^\delta) - \alpha'(z)(s', \phi_y^\delta, \psi_y^\delta) \right| \\
&\quad + \gamma \sum_{s'} \sum_z \left| \frac{\phi_y^{sas'}}{\mathcal{N}_y^{sa}} \frac{\psi_y^{as'z}}{\mathcal{N}_y^{as'}} - \frac{\phi_x^{sas'}}{\mathcal{N}_x^{sa}} \frac{\psi_x^{as'z}}{\mathcal{N}_x^{as'}} \right| \left| \alpha'(z)(s', \phi_y^\delta, \psi_y^\delta) \right| \\
&\leq \gamma \max_{s', z} \left| \alpha'(z)(s', \phi_x^\delta, \psi_x^\delta) - \alpha'(z)(s', \phi_y^\delta, \psi_y^\delta) \right| \\
&\quad + \frac{\gamma \|R\|_\infty}{(1-\gamma)} \sum_{s'} \sum_z \left| \frac{\phi_y^{sas'}}{\mathcal{N}_y^{sa}} \frac{\psi_y^{as'z}}{\mathcal{N}_y^{as'}} - \frac{\phi_x^{sas'}}{\mathcal{N}_x^{sa}} \frac{\psi_x^{as'z}}{\mathcal{N}_x^{as'}} \right|
\end{aligned}$$

Now, we can substitute in (4.5) and get

$$\begin{aligned}
& |\alpha_t^{\pi_t}(s, \phi_x, \psi_x) - \alpha_t^{\pi_t}(s, \phi_y, \psi_y)| \\
&\leq \gamma \max_{s', z} \left| \alpha'(z)(s', \phi_x^\delta, \psi_x^\delta) - \alpha'(z)(s', \phi_y^\delta, \psi_y^\delta) \right| + \frac{\epsilon \gamma \|R\|_\infty}{(1-\gamma)}
\end{aligned}$$

We note that this holds for an arbitrary  $\pi_t$  and thus for arbitrary  $\langle a, a' \rangle$ . Now, define a recurrence by via the max:

$$\begin{aligned}
& \max_{\alpha_t, s} |\alpha_t(s, \phi_x, \psi_x) - \alpha_t(s, \phi_y, \psi_y)| \\
&\leq \gamma \max_{s, a, z} \max_{\alpha_{t-1}, s'} \left| \alpha_{t-1}(s', \phi_x^\delta, \psi_x^\delta) - \alpha_{t-1}(s', \phi_y^\delta, \psi_y^\delta) \right| + \frac{\epsilon \gamma \|R\|_\infty}{(1-\gamma)} \\
&\leq \sum_{\tau=1}^{t-1} \gamma^\tau \frac{\epsilon \|R\|_\infty}{(1-\gamma)} = \frac{\epsilon \|R\|_\infty}{(1-\gamma)} \sum_{\tau=1}^{t-1} \gamma^\tau = \frac{\epsilon \|R\|_\infty}{(1-\gamma)} \frac{\gamma - \gamma^t}{(1-\gamma)}.
\end{aligned}$$

## A. REFERENCES

- [1] S. Abdallah and V. Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *AAMAS*, pages 172–179, 2007.
- [2] C. Amato, J. S. Dibangoye, and S. Zilberstein. Incremental policy generation for finite-horizon DEC-POMDPs. In *ICAPS*, pages 2–9, 2009.
- [3] B. Banerjee, J. Lyle, L. Kraemer, and R. Yellamraju. Sample bounded distributed reinforcement learning for decentralized POMDPs. In *AAAI*, 2012.
- [4] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition-independent decentralized Markov decision processes. *Journal of AI Research*, 22:423–455, 2004.
- [5] D. S. Bernstein, C. Amato, E. A. Hansen, and S. Zilberstein. Policy iteration for decentralized control of Markov decision processes. *Journal of AI Research*, 34:89–132, 2009.
- [6] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.
- [7] G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: A Bayesian approach. In *AAMAS*, pages 709–716, 2003.
- [8] Y.-H. Chang, T. Ho, and L. P. Kaelbling. All learning is local: Multi-agent learning in global reward games. In *NIPS* 16, 2004.
- [9] M. Duff. *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts Amherst, 2002.
- [10] A. Dutech, O. Buffet, and F. Chappillet. Multi-agent systems by incremental gradient reinforcement learning. In *IJCAI*, pages 833–838, 2001.
- [11] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with gaussian processes. In *ICML*, pages 201–208, 2005.
- [12] M. Ghavamzadeh and Y. Engel. Bayesian actor-critic algorithms. In *ICML*, pages 297–304, 2007.
- [13] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:24–49, 2005.
- [14] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, pages 709–715, 2004.
- [15] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:1–45, 1998.
- [16] R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, pages 705–711, 2003.
- [17] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs. In *AAAI*, 2005.
- [18] B. Ng, K. Boakye, C. Meyers, and A. Wang. Bayes-adaptive interactive POMDPs. In *AAAI*, 2012.
- [19] F. A. Oliehoek. Decentralized POMDPs. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State of the Art*, volume 12, pages 471–503. Springer, 2012.
- [20] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *UAI*, pages 489–496, 2000.
- [21] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete bayesian reinforcement learning. In *ICML*, pages 697–704, 2006.
- [22] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.
- [23] S. Ross, B. Chaib-draa, and J. Pineau. Bayes-adaptive POMDPs. In *NIPS* 19, 2007.
- [24] S. Ross, J. Pineau, B. Chaib-draa, and P. Kreitmman. A Bayesian approach for learning and planning in partially observable Markov decision processes. *Journal of Machine Learning Research*, 12:1729–1770, 2011.
- [25] T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning in the iterated prisoner’s dilemma. *Biosystems*, 37:147–166, 1995.
- [26] D. Silver and J. Veness. Monte-carlo planning in large POMDPs. In *NIPS* 23, pages 2164–2172, 2010.
- [27] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [28] W. T. L. Teacy, G. Chalkiadakis, A. Farinelli, A. Rogers, N. R. Jennings, S. McClean, and G. Parr. Decentralized Bayesian reinforcement learning for online agent collaboration. In *AAMAS*, pages 417–424, 2012.
- [29] G. Tesauro and J. O. Kephart. Pricing in agent economies using multi-agent Q-learning. *Autonomous Agents and Multi-Agent Systems*, 5(3):289–304, 2002.
- [30] N. Vlassis, M. Ghavamzadeh, S. Mannor, and P. Poupart. Bayesian reinforcement learning. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State of the Art*, volume 12. Springer, 2012.
- [31] S. J. Witwicki and E. H. Durfee. Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *ICAPS*, 2010.
- [32] H. P. Young. *Strategic Learning and Its Limits*. Oxford University Press, 2004.
- [33] Y. Zeng, P. Doshi, Y. Pan, H. Mao, M. Chandrasekaran, and J. Luo. Utilizing partial policies for identifying equivalence of behavioral models. In *AAAI*, pages 1083–1088, 2011.
- [34] C. Zhang, V. Lesser, and S. Abdallah. Self-organization for coordinating decentralized reinforcement learning. In *AAMAS*, pages 739–746, 2010.