Fifth Workshop on Multi-agent Sequential Decision Making in Uncertain Domains (MSDM)
Toronto, Canada

May 11, 2010

Table of Contents

Coordination vs. information in multi-agent decision processes

Maike Kaufman and Stephen Roberts
Department of Engineering Science
University of Oxford
Oxford, OX1 3PJ, UK
{maike, sjrob}@robots.ox.ac.uk

ABSTRACT

Agent coordination and communication are important issues in designing decentralised agent systems, which are often modelled as flavours of Markov Decision Processes (MDPs). Because communication incurs an overhead, various scenarios for sparse agent communication have been developed. In these treatments, coordination is usually considered more important than making use of local information. We argue that this is not always the best thing to do and provide alternative approximate algorithms based on local inference. We show that such algorithms can outperform the guaranteed coordinated approach in a benchmark scenario.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms

Keywords

Multiagent communication, Multiagent coordination, local decision-making

1. INTRODUCTION

Various flavours of fully and partially observable Markov Decision Processes (MDPs) have gained increasing popularity for modelling and designing cooperative decentralised multi-agent systems [11, 18, 23]. In such systems there is a trade-off to be made between the extent of decentralisation and the tractability and overall performance of the optimal solution. Communication plays a key role in this as it increases the amount of information available to agents but creates an overhead and potentially incurs a cost. At the two ends of the spectrum lie fully centralised multi-agent (PO)MDPs and completely decentralised (PO)MDPs. Decentralized (PO)MDPs have been proven to be NEXP-complete [5, 18] and a considerable amount of work has gone into finding approximate solutions, e.g. [1, 2, 4, 7, 8, 10, 12, 15, 16, 14, 21, 22, 25]

Most realistic scenarios arguably lie somewhere in between full and no communication and some work has focused on scenarios with more flexible amounts of communication. Here, the system usually alternates between full communication among all agents and episodes of zero communication, either to facilitate policy computation for decentralised scenarios [9, 13], to reduce communication overhead by avoiding redundant communications [19, 20] and/or to determine a (near-)optimal communication policy in scenarios where communication comes at a cost [3, 26]. Some of these approaches require additional assumptions, such as transition independence. The focus in most of this work lies on deciding when to communicate, either by pre-computing communication policies or by developing algorithms for on-line reasoning about communication. Such treatment is valuable for scenarios in which inter-agent communication is costly but reliably available. In many real-world systems on the other hand, information exchange between agents will not be possible at all times. This might, for example, be due to faulty communication channels, security concerns or limited transmission ranges. As a consequence agents will not be able to plan ahead about when to communicate but will have to adapt their decision-making algorithms according to whichever opportunities are available. We would therefore like to view the problem of sparse communication as one of good decision-making under differing beliefs about the world. Agents might have access to local observations, which provide some information about the global state of the system. However, these local observations will in general lead to different beliefs about the world and making local decision-choices based on them could potentially lead to uncoordinated collective behaviour. Hence, there is again a trade-off to be made: should agents make the most of their local information, or should overall coordination be valued more highly?

Existing work seems to suggest that coordination should in general be favoured over more informed local beliefs, see for example [8, 19, 20, 24], although the use of local observations has shown some improvement of performance to an existing algorithm for solving DEC-POMDPs [7]. We would like to argue more fundamentally here that focusing on guaranteed coordination will often lead to lower performance and that a strong case can be made for using what local information is available in the decision-making process. For simplicity we will concentrate on jointly observable systems with uniform transition probabilities and free communication, in which agents must sometimes make decisions without being able to communicate observations. Such simple 1-step scenarios could be solved using a dec-POMDP or dec-MDP, but in more complicated settings (e.g. when varying subsets of

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

agents communicate or when communication between agents is faulty) application of a dec-(PO)MDP is not straightforward and possibly not even intractable. Restricting the argument to a subset of very simple scenarios arguably limits its direct appliccability to more complex settings, especially those with non-uniform transition functions. However, it allows us to study the effects of different uses of information in the decision-making algorithms in a more isolated way. Taking other influencing factors such as the approximation of infinite-horizon policy computation into account at this stage, would come at the cost of a less rigorous treatment of the problem. The argument for decision-making based on local information is in principle extendable to more general systems and we believe that understanding the factors which influence the trade-off between coordination and local information gain for simple cases ultimately enable the treatment of more complicated scenarios. In that sense this paper is intended as a first proof of concept.
In the following we will describe exact decision-making based on local beliefs and discuss three simple approximations by which it can be made tractable. Application of the resulting local decision-making algorithms to a benchmark scenario show that they can outperform an approach based on guaranteed coordination for a variety of reward matrices.

2. MULTI-AGENT DECISION PROCESS

Cooperative multi-agent systems are often modelled by a Multi-agent MDP (**MMDP**) [6], Multi-agent POMDP, [18], decentralized MDP (**dec-MDP**) [5] or decentralized POMDP (**dec-POMDP**) [5]. A summary of all these approaches can be found in [18].
Let let $\{N, S, A, O, p_T, p_O, \Theta, R, B\}$ be a tuple where:

- N is a set of n agents indexed by i
- $S = \{S^1, S^2, \ldots\}$ is a set of global states
- $A_i = \{a_i^1, a_i^2, \ldots\}$ is a set of local actions available to agent i
- $A = \{A^1, A^2, \ldots\}$ is a set of joint actions with $A = A_1 \times A_2 \times \ldots \times A_n$
- $O_i = \{\omega_i^1, \omega_i^2, \ldots\}$ is a set of local observations available to agent i
- $O = \{O^1, O^2, \ldots\}$ is a set of joint observations with $O = O_1 \times O_2 \times \ldots O_n$
- $p_T : S \times S \times A \rightarrow [0, 1]$ is the joint transition function where $p_T(S^q|A^k, S^p)$ is the probability of arriving in a state S^q when taking action A^k in state S^p
- $p_O : S \times O \rightarrow [0, 1]$ is a mapping from states to joint observations, where $p_O(O^k|S^l)$ is the probability of observing O^k in state S^l
- $\Theta : O \rightarrow S$ is a mapping from joint observations to global states
- $R : S \times A \times S \rightarrow \mathbb{R}$ is a reward function, where $R(S^p, A^k, S^q)$ is the reward obtained for taking action A^k in a state S^p and transitioning to S^q
- $B = (b_1, \ldots, b_n)$ is the vector of local belief states

A local policy π_i is commonly defined as a mapping from local observation histories to individual actions, $\pi_i : \bar{w}_i \rightarrow A_i$. For the purpose of this work, let a local policy more generally be a mapping from local belief states to local actions, $\pi_i : b_i \rightarrow A_i$, and let a joint policy π be a mapping from global (belief-)states to joint actions, $\pi : S \rightarrow A$ and $\pi : B \rightarrow A$ respectively. Depending on the information exchange between agents, this model can be assigned to one of the following limit cases:

Multi-Agent MDP If agents have guaranteed and free communication among each other and Θ is a surjective mapping, the system is collectively observable. The problem simplifies to finding a joint policy π from global states to joint actions.

Multi-Agent POMDP If agents have guaranteed and free communication but Θ is not a surjective mapping, the system is collectively partially observable. Here the optimal policy is defined as a mapping from belief states to actions.

DEC-MDP If agents do not exchange their observations and Θ is a surjective mapping, the process is jointly observable but locally only partially observable. The aim is to find the optimal joint policy consisting of local policies $\pi = (\pi_1, \ldots, \pi_n)$.

DEC-POMDP If agents do not exchange their observations and Θ is not a surjective mapping, the process is both jointly and locally partially observable. As with the DEC-MDP the problem lies in finding the optimal joint policy comprising local policies.

In all cases the measure for optimality is the discounted sum of expected future rewards. For systems with uniform transition probabilities in which successive states are equally likely and independent of the actions taken, finding the optimal policy simplifies to maximising the immediate reward:

$$V_\pi(S) = R(S, \pi(S)) \tag{1}$$

3. EXACT DECISION-MAKING

Assume that agents are operating in a system in which they rely on regular communication, e.g. a MMDP, and that at a certain point in time they are unable to fully synchronise their respective observations. This need not mean that no communication at all takes place, only that not all agents can communicate with all others. In such a situation their usual means of decision-making (the centralised policy) will not be of use, as they do not hold sufficient information about the global state. As a result they must resort to an alternative way of choosing a (local) action. Here, two general possibilities exist: agents can make local decisions in a way that conserves overall coordination or by using some or all of the information which is only locally available to them.

3.1 Guaranteed coordinated

Agents will be guaranteed to act coordinatedly if they ignore their local observations and use the commonly known prior distribution over states to calculate the optimal joint policy by maximising the expected reward:

$$V_\pi = \sum_S p(S) R(S, \pi(S)) \tag{2}$$

However, this guaranteed coordination comes at the cost of discarding potentially valuable information, thus making a decision which is overall less informed.

3.2 Local

Consider instead calculating a local solution π_i to $V_{\pi_i}(b_i)$, the local expected reward given agent i's belief over the global state:

$$V_{\pi_i}(b_i) = \sum_S \sum_{B_{-i}} p(B_{-i}|b_i)p(S|B) \sum_{\pi_{-i}} R(S, \pi(B)) \quad (3)$$

where $B = (b_1, \ldots, b_n)$ is the vector comprising local beliefs and $\pi(B) = (\pi_1(b_1), \ldots, \pi_n(b_n))$ is the joint policy vector and we have implicityly assumed that without prior knowledge all other agents' policies are equally likely. With this the total reward under policy π_i as expected by agent i is given by

$$V_{\pi_i} = \sum_S \sum_{b_i} p(S)p(b_i|S)V_{\pi_i}(b_i) \quad (4)$$

Calculating the value function in equation 3 requires marginalising over all possible belief states and policies of other agents and will in general be intractable. However, if it were possible to solve this equation exactly, the resulting local policies should never perform worse than an approach which guarantees overall coordination by discarding local observations. This is because the coordinated policies are a subset of all policies considered here and should emerge as the optimal policies in cases where coordination is absolutely crucial. As a result the overall reward V_{π_i} expected by any agent i will always be greater or equal to the expected reward under a guaranteed coordinated approach as given by equation 2. The degree to which this still holds and hence to which a guaranteed coordinated approach is to be favoured over local decision-making therefore depends on the quality of any approximate solution to equation 3 and the extent to which overall coordination is rewarded.

4. APPROXIMATE DECISION-MAKING

The optimal local one-step policy of an individual agent is simply the best response to the possible local actions the others could be choosing at that point in time. The full marginalisation over others' local beliefs and possible policies therefore amounts to a marginalisation over all others' actions. Calculating this requires knowing the probability distribution over the current state and remaining agents' action choices, given agent i's local belief b_i, $p(S, A_{-i}|b_i)$. Together with equation 3 the value of a local action given the current local belief over global state then becomes

$$V_i(a_i, b_i) = \sum_S \sum_{A_{-i}} p(S, A_{-i}|b_i)R(S, a_i, A_{-i}) \quad (5)$$

This re-formulation in terms of $p(S, A_{-i}|b_i)$ significantly reduces the computational complexity compared to iterating over all local beliefs and policies. However, its exact form will in general not be known without performing the costly iteration over others actions and policies. To solve equation 5 we therefore need to find a suitable approximation to $p(S, A_{-i}|b_i)$.

Agent i's joint belief over the current state and other agents' choice of actions can be expanded as

$$p(S, A_{-i}|b_i) = p(A_{-i}|S)p(S|b_i) = p(A_{-i}|S)b_i(S) \quad (6)$$

Finding the local belief state $b_i(S)$ is a matter of straightforward Bayesian inference based on the knowledge of the system's dynamics. One convenient way of solving this calculation is by casting the scenario as a graphical model and using standard solution algorithms to obtain the marginal distribution $b_i(S)$. For the systems considered in this work, where observations only depend on the current state we can use the sum-product algorithm [17], which makes the calculation of local beliefs particularly easy.

Obtaining an expression for the local belief over all other agents' actions is less simple: Assuming $p(A_{-i}|S)$ were known agent i could calculate it's local expectation of future rewards according to equation 6 and choose the local action which maximises this value. All remaining agents will be executing the same calculation simultaneously. This means that agent i's distribution over the remaining agents' actions is influenced by the simultaneous decision-making of the other agents, which in turn depends on agent i's action choice. Finding a solution to these interdependent distributions is not straightforward. In particular, an iterative solution based on reasoning over others' choices will lead to an infinite regress of one agent trying to choose its best local policy based on what it believes another agent's policy to be even though that action is being decided on at the same time. Below we describe three heuristic approaches by which the belief over others actions could be approximated in a quick, simple way.

4.1 Optimistic approximation

From the point of agent i an optimistic approximation to $p(A_{-i}|S)$ is to assume that all other agents choose the local action given by the joint centralised policy for a global state, that is

$$p(A_{-i}|S) = \begin{cases} 1 & \text{if } A_{-i} = \pi(S)_{-i} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

This is similar to the approximation used in [7].

4.2 Uniform approximation

Alternatively, agents could assume no prior knowledge about the actions others might choose at any point in time by putting a uniform distribution over all possible local actions:

$$p(a_j = a_j^k|S) = \frac{1}{|A_j|} \quad (8)$$

and

$$p(A_{-i}|S) = \prod_{j \neq i} p(a_j^k|S) \mid a_j^k \in A \quad (9)$$

4.3 Pessimistic approximation

Finally, a pessimistic agent could assume that the local decision-making will lead to sub-optimal behaviour and that the other agents can be expected to choose the worst possible action in a given state.

$$p(A_{-i}|S) = \begin{cases} 1 & \text{if } A_{-i} = (\arg\min_A V_{centralised}(S))_{-i} \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

Each of these approximations can be used to implement local decision-making by calculating the expected value of a local action according to equation 5. Ideally we would like to compare the overall expected reward (see equation 4) under each of the approximate local algorithms and compare

Actions	Rewards
both choose tiger	−50
both choose reward	100
both choose nil	0
both wait	−2
one tiger, one nil	−100
one tiger, one reward	−50
one tiger, one waits	−101
one nil, one waits	−1
one nil, one reward	50
one reward, one waits	49

(a) Reward setting 1: some reward for uncoordinated actions

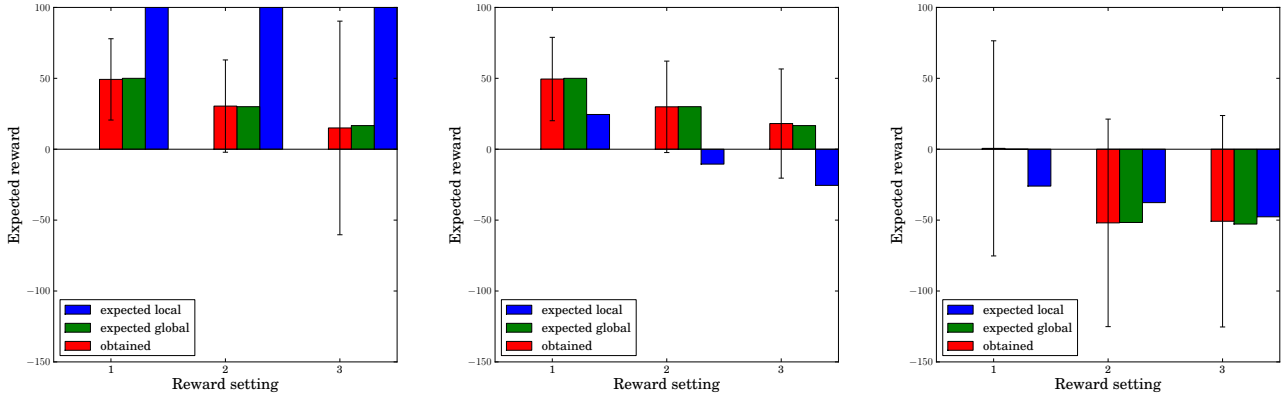
Actions	Rewards
both choose tiger	−20
both choose reward	100
both choose nil	0
both wait	−2
one tiger, one nil	−100
one tiger, one reward	−100
one tiger, one waits	−101
one nil, one waits	−1
one nil, one reward	20
one reward, one waits	19

(b) Reward setting 2: small reward for uncoordinated actions

Actions	Rewards
both choose tiger	−20
both choose reward	100
both choose nil	0
both wait	−2
one tiger, one nil	−100
one tiger, one reward	−100
one tiger, one waits	−101
one nil, one waits	−1
one nil, one reward	0
one reward, one waits	−1

(c) Reward setting 3: no reward for uncoordinated actions

Table 1: Reward matrices for the Tiger Scenario with varying degrees by which uncoordinated actions are rewarded. Joint actions for which the rewards were varied are shaded.



(a) Optimistic algorithm (b) Uniform algorithm (c) Pessimistic algorithm

Figure 1: Average obtained reward (red diamonds) compared to expected reward (green squares) for different approximate decision-making algorithms. Data points were obtained by averaging over 500 time-steps. The uniform algorithm consistently under-estimates the expected reward, while the pessimistic algorithm both under- and over-estimates, depending on the setting of the reward matrix. The optimistic algorithm tends to over-estimate the reward but has the smallest deviation and in particular approximates it well for the setting which is most favourable to uncoordinated actions.

it to the overall reward expected under a guaranteed coordinated approach, as given by equation 2. This is not possible because the expectation values calculated from the approximate beliefs will in turn only be approximate. For example the optimistic algorithm might be expected to make over-confident approximations to the overall reward, while the pessimistic approximation might underestimate it. In general it will therefore not be possible to tell from the respective expected rewards which algorithm will perform best on average for a given decision process. We can, however, obtain a first measure for the quality of an approximate algorithm by comparing its expected performance to the actual performance for a benchmark scenario.

5. EXAMPLE SIMULATION

We have applied the different decision-making algorithms to a modified version of the Tiger Problem, which was first introduced by Kaelbling et. al. [11] in the context of single-agent POMDPs and has since been used in modified forms as a benchmark problem for dec-POMDP solution techniques [2, 12, 13, 16, 19, 20, 22, 25]. For a comprehensive description of the initial multi-agent formulation of the problem see [12]. To adapt the scenario to be an example of a dec-MDP with uniform transition probabilities as discussed above, we have modified this scenario in the following way: Two agents are faced with three doors, behind which sit a tiger, a reward or nothing. At each time step both agents can choose to open one of the doors or to do nothing and wait. These actions are carried out deterministically and after both agents have chosen their actions, an identical reward is received (according to the commonly known reward matrix) and the configuration behind the doors is randomly re-set to a new state. Prior to choosing their actions the agents are both informed about the contents behind one of the doors, but never both about the same door. If agents can exchange their observations prior to making their decisions, the problem becomes fully observable and the optimal choice of action is straightforward. If, on the other hand, they cannot exchange their observations, they will both hold differing, incomplete information about the global state which will lead to differing beliefs over where the tiger and the reward are located.

5.1 Results

We have implemented the Tiger Scenario as described

above for different reward matrices and have compared the performance of the various approximate algorithms to a guaranteed coordinated approach in which agents discard their local observations and use their common joint belief over the global state of the system to determine the best joint action. Each scenario run consisted of 500 time-steps. In all cases the highest reward (lowest penalty) was given to coordinated joint actions. The degree by which agents received partial awards for uncoordinated actions varied for the different settings. For a detailed listing of the reward matrices used see table 1.

Figure 4 shows the expected and average obtained rewards for the different reward settings and approximate algorithms described above. As expected the average reward gained during the simulation differs from the expected reward as predicted by an individual agent. While this difference is quite substantial in some cases, it is consistently smallest for the optimistic algorithm.

Figure 2 shows the performance of the approximate algorithms compared to the performance of the guaranteed coordinated approach. The pessimistic approach consistently performs worse than any of the other algorithms, while the optimistic and the uniform approach achieve similar performance. Interestingly, the difference between the expected and actual rewards under the different approximate algorithms (figure 4) does not provide a clear indicator for the performance of an algorithm. Compared to the guaranteed coordinated algorithm the performance of the optimistic/uniform algorithms depends on the setting of the reward matrix. They clearly outperform it for setting 1, while achieving less average reward for setting 3. In the intermediate region all three algorithms obtain similar rewards. It is important to remember here that even for setting 1 the highest reward is awarded to coordinated actions and that setting 3 is the absolute limit case in which no reward is gained by acting uncoordinatedly. We would argue that the latter is a somewhat artificial scenario and that many interesting applications are likely to have less extreme reward matrices. The results in figure 2 suggest that for such intermediate ranges even a simple approximate algorithm for decision-making based on local inference might outperform an approach which guarantees agent coordination.

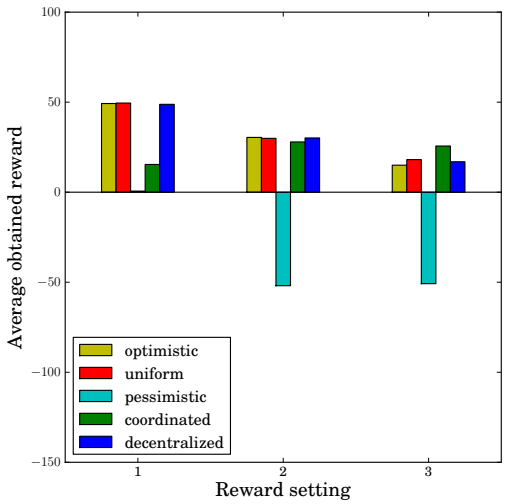


Figure 2: Average obtained reward under approximate local decision-making compared to guaranteed coordinated algorithm for different reward matrices. Data points were obtained by averaging over 500 time-steps

6. CONCLUSIONS

We have argued that coordination should not automatically be favoured over making use of local information in multi-agent decision processes with sparse communication and have described three simple approximate approaches that allow local decision-making based on individual beliefs. We have compared the performance of these approximate local algorithms to that of a guaranteed coordinated approach on a modified version of the Tiger Problem. Some of the approximate algorithms showed comparable or better performance than the coordinated algorithm for some settings of the reward matrix. Our results can thus be understood as first evidence that strictly favouring agent coordination over the use of local information can lead to lower collective performance than using an algorithm for seemingly uncoordinated local decision making. More work is needed to fully understand the influence of the reward matrix, system dynamics and belief approximations on the performance of the respective decision-making algorithms. Future work will also include the extension of the treatement to truly sequential decision processes where the transition function is no longer uniform and independent of the actions taken.

7. REFERENCES

[1] C. Amato, D. S. Bernstein, and S. Zilberstein. Optimal fixed-size controllers for decentralized pomdps. In *In Proceedings of the Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM) at AAMAS*, 2006.

[2] C. Amato, A. Carlin, and S. Zilberstein. Bounded dynamic programming for decentralized pomdps. In *In AAMAS 2007 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 2007.

[3] R. Becker, V. Lesser, and S. Zilberstein. Analyzing myopic approaches for multi-agent communication. In *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 550–557, Sept. 2005.

[4] D. S. Bernstein. Bounded policy iteration for decentralized pomdps. In *In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1287–1292, 2005.

[5] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27(4):819–840, 2002.

[6] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 478–485, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[7] A. Chechetka and K. Sycara. Subjective approximate solutions for decentralized pomdps. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, New York, NY, USA, 2007. ACM.

[8] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 136–143, Washington, DC, USA, 2004. IEEE Computer Society.

[9] C. V. Goldman and S. Zilberstein. Communication-based decomposition mechanisms for decentralized mdps. *Artificial Intelligence Research*, 32:169–202, 2008.

[10] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI'04: Proceedings of the 19th national conference on Artifical intelligence*, pages 709–715. AAAI Press / The MIT Press, 2004.

[11] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[12] R. Nair, R. Nair, M. Tambe, M. Tambe, S. Marsella, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *In IJCAI*, pages 705–711, 2003.

[13] R. Nair, M. Roth, and M. Yohoo. Communication for improving policy computation in distributed pomdps. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1098–1105, Washington, DC, USA, 2004. IEEE Computer Society.

[14] F. A. Oliehoek, M. T. J. Spaan, S. Whiteson, and N. Vlassis. Exploiting locality of interaction in factored dec-pomdps. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 517–524, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[15] F. A. Oliehoek and N. Vlassis. Q-value functions for decentralized pomdps. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[16] F. A. Oliehoek and N. Vlassis. Q-value heuristics for approximate solutions of dec-POMDPs. In *Proc. of the AAAI spring symposium on Game Theoretic and Decision Theoretic Agents*, pages 31–37, 2007.

[17] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[18] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:2002, 2002.

[19] M. Roth, R. Simmons, and M. Veloso. Decentralized communication strategies for coordinated multi-agent policies. In *Multi-Robot Systems: From Swarms to Intelligent Automata, volume IV*. Kluwer Avademic Publishers, 2005.

[20] M. Roth, R. Simmons, and M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 786–793, New York, NY, USA, 2005. ACM.

[21] M. Roth, R. Simmons, and M. Veloso. Exploiting factored representations for decentralized execution in multiagent teams. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–7, New York, NY, USA, 2007. ACM.

[22] S. Seuken. Memory-bounded dynamic programming for dec-pomdps. In *In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI*, pages 2009–2015, 2007.

[23] 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, 2008.

[24] M. T. J. Spaan, F. A. Oliehoek, and N. Vlassis. Multiagent planning under uncertainty with stochastic communication delays. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 338–345, 2008.

[25] D. Szer and F. Charpillet. Point-based dynamic programming for dec-pomdps. In *AAAI'06: proceedings of the 21st national conference on Artificial intelligence*, pages 1233–1238. AAAI Press, 2006.

[26] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: model and experiments. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 616–623, New York, NY, USA, 2001. ACM.

Influence-based Policy Abstraction for Weakly-coupled Dec-POMDPs

Stefan J. Witwicki and Edmund H. Durfee
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109
{witwicki,durfee}@umich.edu

ABSTRACT

Decentralized POMDPs are powerful theoretical models for coordinating agents' decisions in uncertain environments, but the generally-intractable complexity of optimal joint policy construction presents a significant obstacle in applying Dec-POMDPs to problems where many agents face many policy choices. Here, we argue that when most agent choices are independent of other agents' choices, much of this complexity can be avoided: instead of coordinating full policies, agents need only coordinate policy abstractions that explicitly convey the essential interaction influences. To this end, we develop a novel framework for influence-based policy abstraction for weakly-coupled transition-dependent Dec-POMDP problems that subsumes several existing approaches. In addition to formally characterizing the space of transition-dependent influences, we provide a method for computing optimal and approximately-optimal joint policies. We present an initial empirical analysis, over problems with commonly-studied flavors of transition-dependent influences, that demonstrates the potential computational benefits of influence-based abstraction over state-of-the-art optimal policy search methods.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Theory, Algorithms, Performance

Keywords

Coordination, Structured Interaction, Agent Influence

1. INTRODUCTION

Agent team coordination in partially-observable, uncertain environments is a problem of increasing interest to the research community. The decentralized partially-observable Markov decision process (Dec-POMDP) provides an elegant theoretical model for representing a rich space of agent behaviors, observability restrictions, interaction capabilities, and team objectives. Unfortunately, its applicability and effectiveness in solving problems of significant size has been substantially limited by its generally-intractable complexity [6]. This is largely due to the policy space explosion that comes with each agent having to consider the possible observations and actions of its peers, on top of its own observations and actions.

To combat this complexity, researchers have sought tractable Dec-POMDP subclasses wherein agents are limited in their interactions. For instance, there has been significant effort in developing efficient, scalable solution methods for Transition-Independent DEC-MDPs [2] and Network-Distributed POMDPs [10, 13], where agents can only influence one another through their reward functions. Because the agents' transitions and observations remain independent, the complexity of this subclass is immune to the growth of the general Dec-POMDP class. Some work has been done in exploring subclasses where agents can influence each others' transitions [2, 4, 7, 8, 11, 12, 16]. However, these either introduce additional restrictions on individual agent behavior [4, 7, 16], yield no guarantees of optimality [12], or have only been shown effective for teams of two or three agents executing for a handful of time steps [2, 8, 11].

This paper presents an alternative approach to planning for teams of agents with transition-dependent influences. To address the issue of policy space complexity head-on, we contribute a formal framework for policy abstraction that subsumes several existing methods [2, 15, 16]. The primary intuition of our work is that by planning joint behavior using abstractions of policies rather than the policies themselves, weakly-coupled agents can form a compact *influence space* over which to reason more efficiently. In contrast to Allen and Zilberstein's work on quantifying the amount of influence agents have on the problem (for predicting efficiency and effectiveness of solution methods) [1], we instead focus on characterizing efficient representations of interagent influence (for design of efficient, exploitative solution methods).

We begin by framing the problem as a class of Transition-Decoupled POMDPs (TD-POMDPs) with an expressive, yet natural, representation of agents with rich behaviors whose interactions are limited. Moreover, TD-POMDPs lead us to a systematic analysis of the influences agents can exert on one another, culminating in a succinct model that accommodates both exact and approximate representations of interagent influence. To take advantage of these beneficial traits, we contribute a general-purpose influence-space search algorithm that, based on initial empirical evidence, demonstrably advances the state-of-the-art in exploiting weakly-coupled structure and scaling transition-dependent problems to larger teams of agents without sacrificing optimality.

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

2. COORDINATION OF WEAKLY-COUPLED AGENTS

We focus on the problem of planning for agents who are nearly independent of one another, but whose limited, structured dependencies require coordination to maximize their collective rewards. Domains for which such systems have been proposed include the coordination of military field units [14], disaster response systems [12], and Mars rover exploration [8]. Here, we introduce a class of Dec-POMDPs called Transition-Decoupled POMDPs (TD-POMDPs) that, while still remaining quite general, provides a natural representation of the weakly-coupled structure present in these kinds of domains.

2.1 Autonomous Planetary Exploration

As a concrete example, consider the team of agents pictured in Figure 1A whose purpose is to explore the surface of a distant planet. There are rovers that move on the ground collecting and analyzing soil samples, and orbiting satellites that (through the use of cameras and specialized hardware) perform various imaging, topography, and atmospheric analysis activities. In representing agents' activities, we borrow from the TAEMS language specification [5], assigning to each abstract task a *window* of feasible execution times and a set of possible *outcomes*, each with an associated *duration* and *quality* value. For example, the satellite agent in Figure 1A has a path-planning task that may take 2 hours and succeed with probability 0.8 or may fail (achieving quality 0) with probability 0.2 (such as when its images are too blurry to plan a rover path). Surface conditions limit the rover's visit to site A to occur between the hours of 2 and 8. Additional constraints and dependencies exist among each individual agent's tasks (denoted by lines and arrows).

Although each agent has a different view of the environment and different capabilities (as indicated by their local model bubbles), it is through their limited, structured interactions that they are able to successfully explore the planet. For instance, the outcome of the satellite's path-planning task influences the probabilistic outcome of the rover's site-visiting task. Navigating on its own, the rover's trip will take 6 hours, but with the help of the satellite agent, its trip will take only 3 hours (with 0.9 probability). In order to maximize productivity (quantified as the sum of outcome qualities achieved over the course of execution), agents should carefully plan (in advance) policies that coordinate their execution of interdependent activities. Though simplistic, this example gives a flavor of the sorts of planning problems that fit into our TD-POMDP framework.

2.2 Transition-Decoupled POMDP Model

The problem from Figure 1 can be modeled using the finite-horizon Dec-POMDP, which we now briefly review. Formally, this decision-theoretic model is described by the tuple $\langle S, A, P, R, \Omega, O\rangle$, where S is a finite set of world states (which model all features relevant to all agents' decisions), with distinguished initial state s^0. $A = \times_{1\leq i\leq n} A_i$ is the joint action space, each component of which refers to the set of actions of an agent in the system. The transition function $P(s'|s,a)$ specifies the probability distribution over next states given that joint action $a = \langle a_1, a_2, \ldots, a_n\rangle \in A$ is taken in state $s \in S$. The reward function $R(s,a,s')$ expresses the immediate value of taking joint action $a \in A$ in state $s \in S$ and arriving in state $s' \in S$; the aim is to maximize the expected cumulative reward from time steps 1 to T (the horizon). The observation function $O(o|a,s')$ maps joint actions and resulting states to probabilities of joint observations, drawn from finite set $\Omega = \times_{1\leq i\leq n}\Omega_i$. We denote the observation history for agent i as $\vec{o}_i^{\,t} = \langle o_i^1, \ldots, o_i^t\rangle \in \Omega_i^t$, the set of observations i experienced from time step 1 to $t \leq T$. A solution to the Dec-POMDP comes in the form of a joint policy $\bar{\pi} = \langle \pi_1, \ldots, \pi_n\rangle$, where each component π_i (agent i's local policy) maps agent i's observation history $\vec{o}_i^{\,t}$ to an action a_i, thereby providing a decision rule for any sequence of observations that each agent might encounter.

Though the general class of Dec-POMDPs accommodates arbitrary interactions between agents through the transition and reward functions, our example problem contains structure that translates to the following useful properties. First, the world state is *factored* into state features $s = \langle a, b, c, d, \ldots\rangle$, each of which represent a different aspect of the environment. In particular, different features are relevant to different agents. Whereas a rover agent may be concerned with the composition of the soil sample it has just collected, this is not relevant to the satellite agent. As with other related models (e.g. those discussed in Section 2.3), we assume a particular grouping of world state features into local features that make up an agent's *local state* s_i. We introduce a further decomposition of local state (that is unique to the TD-POMDP class) whereby agent i's local state s_i is comprised of three disjoint feature sets: $s_i = \langle \bar{u}_i, \bar{l}_i, \bar{n}_i\rangle$, whose contents are as follows.

- *uncontrollable features* $\bar{u}_i = \langle u_{i1}, u_{i2}, ...\rangle$ are those features that are not controllable [6] by any agent, but may be observable by multiple agents. Examples include *time-of-day* or *temperature*.
- *locally-controlled features* $\bar{l}_i = \langle l_{i1}, l_{i2}, ...\rangle$ are those features whose values may be altered through the actions of agent i, but are not (directly) altered through the actions of any other agent; a rover's position, for instance.
- *nonlocal(ly-controlled) features* $\bar{n}_i = \langle n_{i1}, ...\rangle$ are those features that are each controlled by some other agent but whose values directly impact i's local transitions (Eq. 1).

With this factoring, division of world state features into agents' local states is not strict. *Uncontrollable features* may be part of more than one agent's state. And each *nonlocal feature* in agent i's local state appears as a *locally-controlled feature* in the local state of exactly one other agent. In the example (Figure 1), the rover models whether or not the satellite agent has planned a path for it, so *path-A-planned* would be a nonlocal feature in the rover's local state.

The reward function R is decomposed into into local reward functions, each dependent on local state and local action: $R(s,a,s') = F(R_1(s_1,a_1,s_1'), ..., R_n(s_n,a_n,s_n'))$. The joint reward composition function $F()$ has the property that increases in component values do not correspond to decreases in joint value: $r_i > r_i' \rightarrow F(r_1, ..., r_{i-1}, r_i, r_{i+1}, ..., r_n) \geq F(r_1, ..., r_{i-1}, r_i', r_{i+1}, ..., r_n)\ \forall r_1, ..., r_{i-1}, r_{i+1}, ..., r_n$. In the example problem, local rewards are the qualities attained from the tasks that the agents execute, which combine by summation to yield the joint reward by which joint policies are evaluated.

The observation function is similarly factored $O(o|a,s') = \prod_{1\leq i\leq n} O_i(o_i|a_i,s_i')$, allowing agents direct (partial) observations of their local state features but not of features outside their local states. Note, however, that this does not imply

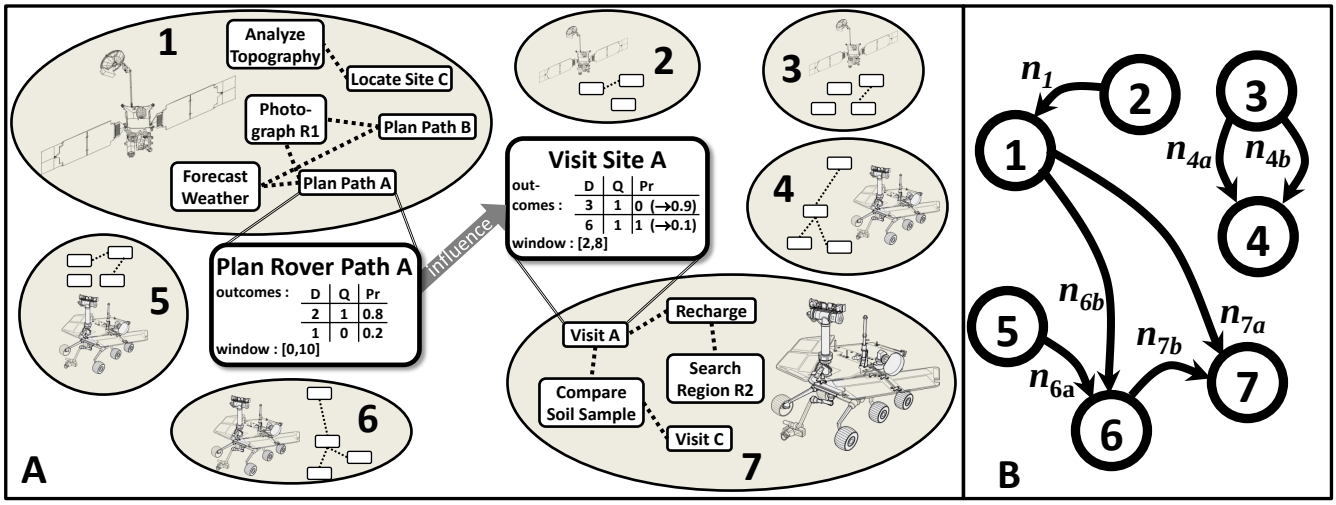


Figure 1: Autonomous Exploration Example.

observation independence (whereas other models [3, 10] do) because of the shared uncontrollable and nonlocal state features. Likewise, this model is *transition-dependent*, as the values of nonlocal features controlled by agent j may influence the probabilistic outcomes of agent i's actions. Formally, agent i's local transition function, describing probability of next local state $s_i^{t+1} = \langle \bar{u}_i^{t+1}, \bar{l}_i^{t+1}, \bar{n}_i^{t+1} \rangle$ given that joint action a is taken in world state s^t, is the product of three independent terms:

$$Pr\left(s_i^{t+1}|s^t, a\right) = Pr\left(\bar{u}_i^{t+1}|\bar{u}_i^t\right) \cdot Pr\left(\bar{l}_i^{t+1}|\bar{l}_i^t, \bar{n}_i^t, \bar{u}_i^t, a_i\right) \cdot Pr\left(\bar{n}_i^{t+1}|s^t - \bar{l}_i^t, a_{\neq i}\right) \quad (1)$$

The result of this factorization is a structured transition dependence whereby agents alter the effects of each others' actions sequentially but not concurrently. Agent i may set the value of one of agent j's nonlocal state features and agent j's subsequent transitions are influenced by the new value.

The structure we have identified is significant because it decouples the Dec-POMDP model into a set of weakly-coupled local POMDP models that are tied to one another by their transition influences. Without the existence of *nonlocal features*, an agent cannot influence another's observations, transitions, or rewards, and the agents' POMDPs become completely independent decision problems. With an increasing presence of nonlocal features, the agents subproblems become more and more strongly coupled. We can concisely describe the coupling and locality of interaction in a TD-POMDP problem with an *interaction digraph* (Figure 1B), which represents each instance of a nonlocal feature with an arc between agent nodes. As pictured, the interaction digraph for our example problem contains an arc from agent 1 (the satellite) to agent 7 (the rover) labeled n_{7a} that refers to the nonlocal feature *path-A-planned*.

2.3 Relationships to Other Classes

Although the TD-POMDP is less general than the Dec-POMDP (and the factored Dec-POMDP [11]), it is more general than prior transition-dependent Dec-POMDP subclasses [2, 4]. The OC-DEC-MDP [4] assumes fixed execution ordering over agent tasks and dependencies in the form of task precedence relationships. The Event-Driven DEC-MDP [2] is more closely related, but it assumes local full observability, and restricts transition dependencies to take the form of mutually exclusive events which could trivially be mapped to nonlocal features in the TD-POMDP model. The TD-POMDP is also more general than the DPCL model [12] in its representation of observation (since local observations can depend on other agents' actions), but less general in its representation of interaction (since agents cannot affect each others' local transitions concurrently). Generality aside, we contend that the factorization that we have defined provides a very natural representation of agent influences, making it straightforward to model problems with TD-POMDPs. Furthermore, as we shall see in Section 3, the structure made explicit by the TD-POMDP leads us to a broad characterization of transition-dependent influences and a systematic methodology for abstracting those influences that subsumes several other approaches [2, 14, 16].

2.4 Decoupled Solution Methodology

To take advantage of the TD-POMDP's weakly-coupled interaction structure, we build upon a general solution methodology that decouples the joint policy formulation. Central to this approach is the use of local models, whereby each agent can separately compute its individual policy. As derived by Nair *et al.* [9], any Dec-POMDP can be transformed into a single-agent POMDP for agent i once the policies of i's peers have been fixed. This *best-response* model is prohibitive to solve in the general case (given that the agent must reason about the possible observations of the other agents), but in various restricted contexts, iterative best-response algorithms have been devised which provide substantial computational leverage [3, 10]. As we describe later on, the TD-POMDP (which is composed of weakly-coupled local POMDPs) can be decoupled into fully-independent POMDPs that have been augmented with compact models of influence.

Given this decoupling scheme, planning the joint policy becomes a search through the space of combinations of optimal local policies (each found by solving a local best-response model). This approach is taken in much of the literature to solve transition-independent reward-dependent models (e.g

TI-DEC-MDPs [3], ND-POMDPs [10, 13]). And while some approaches [2, 12, 14, 15] have solved transition-dependent models in this way, the results have been either limited to just two agents [2], or to approximately-optimal solutions without formal guarantees [12, 14, 15]. In the remainder of this paper, we present and evaluate a formal framework that subsumes previous transition-dependent methods and produces provably optimal solutions, focusing on abstraction to make the search tractable and scalable.

3. INFLUENCE-BASED POLICY ABSTRACTION

The Dec-POMDP joint policy space (which is exponential in the number of observations and doubly exponential in the number of agents and the time horizon) grows intractably large very quickly. The primary intuition behind how our approach confronts this intractability is that, by abstracting weakly-coupled interaction influences from local policies, an influence space emerges that is more efficient to explore than the joint policy space. We begin by discussing policy abstraction in the context of a simple, concrete example with some very restrictive assumptions. Over the course of this section, we gradually build up a less restrictive language through which agents can convey their abstract influences, culminating in a formal characterization of the general space of interaction influences for the class of TD-POMDPs.

Figure 2 portrays an interaction wherein one rover (R5) must prepare a site before another rover (R6) can benefit from visiting the site. Assume that apart from this interaction, the two agents' problems are completely independent. Neither of them interact with any other agents, nor do they share any observations except for the occurrence of site C's preparation and the current time. In a TD-POMDP, this simple interaction corresponds to the assignment of a single boolean nonlocal feature *site-C-prepared* that is locally-controlled by R5, but that influences (and is nonlocal to) R6. Thus, in planning its own actions, R6 needs to be able to make predictions about *site-C-prepared*'s value (influenced by R5) over the course of execution.

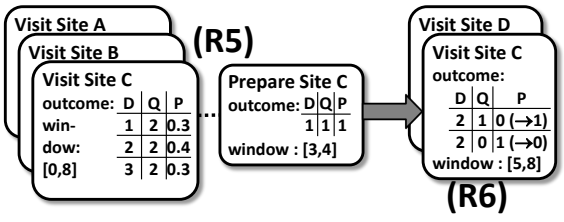


Figure 2: Example of highly-constrained influence.

Definition 1. For a TD-POMDP interaction x represented by agent j as nonlocal feature n_{jx}, which is controllable by agent i and affects the transitions of agent j, we define the **influence** of i's policy π_i on n_{jx}, denoted $\Gamma_{\pi_i}(n_{jx}) = Pr\left(n_{jx}|\ldots\right)$, to be a sufficient summary of π_i for agent j to model expected changes to n_{jx} and to plan optimal decisions given that agent i adopts π_i.

By representing the influence of R5's policy with a distribution $Pr(\textit{site-C-prepared}|...)$ as in Definition 1, R6 can construct a transition model for nonlocal feature *site-C-prepared*. The last multiplicand of Equation 1 suggests that this construction requires computing a transition probability for every value of the (nonlocal subsection of) world state $(s^t - \overline{l}_i^t)$, nonlocal action $(a_{\neq i})$, and next nonlocal feature value. However, in this particular problem, R6 does not need a complete distribution that is conditioned on all features. In fact, the only features that R6 can use to predict the value of *site-C-prepared* are *time* and *site-C-prepared* itself. Although *site-C-prepared* is dependent on other features from R5's local state, R6 cannot observe any evidence of these features except through its (perhaps partial) observations of *site-C-prepared* and *time*. Thus, all other features can be marginalized out of the distribution $Pr(\textit{site-C-prepared}|...)$.

In this particular example, the only influence information that is relevant to R6 is the probability with which *site-C-prepared* will become *true* conditioned on $time = 4$. At the start of execution, *site-C-prepared* will take on value $false$ and remain $false$ until R5 completes its "Prepare Site C" task (constrained to finish only at time 4, if at all, given the task window in Figure 2). After the site is prepared, the feature will remain *true* thereafter until the end of execution. With these constraints, there is no uncertainty about <u>when</u> *site-C-prepared* will become *true*, but only <u>if</u> it will become *true*. Hence, the influence of R5's policy can be summarized with just a single probability value, $Pr(\textit{site-C-prepared} = true|time = 4)$, from which R6 can infer all transition probabilities of *site-C-prepared*.

Reasoning about concise influence distributions instead of full policies can be advantageous in the search for optimal joint policies. The **influence space** is the domain of possible assignments of the influence distribution, each of which is achieved by some feasible policy. This corresponds to the feasible values of $Pr(\textit{site-C-prepared} = true|time = 4)$ in our simple example. As shown in Figure 2, R5 has several sites it can visit, each with uncertain durations. In general, different policies that it adopts may achieve different interaction probabilities. However, due to the constraints in Figure 2, many of R5's policies will map to the same influence value. For instance, any two policies that differ only in the decisions made after time 3 will yield the same value for $Pr(\textit{site-C-prepared} = true|time = 4)$. For this example, the influence space is strictly smaller than the policy space. Thus, by considering only the feasible influence values, agents avoid joint reasoning about the multitude of local policies with equivalent influences.

3.1 A Categorization of Influences

The influence in the example from Figure 2 has a very simple structure due to the highly-constrained transitions of the nonlocal feature. By removing constraints, we can more generally categorize the influence between R5 and R6. Let the window of execution of "Prepare Site C" be unconstrained: $[0, 8]$. With this change, there is the possibility of R5 preparing site C at any time during execution. The consequence is that a single probability is no longer sufficient to characterize R5's influence. Instead of representing a single probability value, R6 needs to represent a probability for each time *site-C-prepared* could be set to true. In other words, this influence is dependent on a feature of the agents' state: *time*.

Definition 2. An influence $\Gamma_{\pi_i}(n_{jx})$ is **state-dependent** w.r.t. feature f if its summarizing distribution must be conditioned on the value of f: $Pr\left(n_{jx}|f, ...\right)$.

As we have seen in prior work [16], the set of probabilities $Pr(\textit{site-C-prepared}|\textit{time})$ is an abstraction of R5's policy that accommodates temporal uncertainty of the interaction.

Generalizing further, the probability of an interaction may differ based on both present and past values of state features. For instance, in the example from Figure 1A, if it is cloudy in the morning, this might prohibit the satellite from taking pictures, and consequently lower the probability that it plans a path for the rover in the afternoon. So by monitoring the history of the weather, the rover could anticipate the lower likelihood of help from the satellite, and might change some decisions accordingly. Becker employs this sort of abstraction in his Event-driven DEC-MDP solution algorithm, where he relates probabilities of events to dependency histories [2].

Definition 3. Influence $\Gamma_{\pi_i}(n_{jx})$ is **history-dependent** w.r.t. feature f if its summarizing distribution must be conditioned on the history of f: $Pr\left(n_{jx}^{t+1}|\vec{f}^{\,t},...\right)$.

Moreover, there may also be dependence between influences. For instance, agent 4 has two arcs coming in from agent 3 (in Figure 1B), indicating that agent 3 is exerting two influences, such as if agent 3 could plan two different paths for agent 4. In the case that agent 3's time spent planning one path leaves too little time to plan the other path, the nonlocal features n_{4a} and n_{4b} are highly correlated, requiring that their joint distribution be represented.

Definition 4. Influence $\Gamma_{\pi_i}(n_x)$ and influence $\Gamma_{\pi_i}(n_y)$ are **influence-dependent** (on each other) if their summarizing distributions are correlated, requiring $Pr\left(n_x, n_y|...\right)$.

3.2 A Comprehensive Influence Model

With the preceding terminology, we have systematically though informally introduced an increasingly comprehensive characterization of transition influences. A given TD-POMDP influence might be state-dependent and history-dependent on multiple features, or even dependent on the history of another influence. Furthermore, there may be chains of influence-dependent influences. In Figure 1B, for example, agent 7 models two nonlocal features, one (n_{7a}) influenced by agent 1 and the other (n_{7b}) influenced by agent 6. The additional arc between agents 1 and 6 forms an undirected cycle that implies a possible dependence between n_{7a} and n_{7b} by way of n_{6b}. The only way to ensure a complete influence model is to incorporate all three influences into a joint distribution.

In general, for any team of TD-POMDP agents, their influences altogether constitute a Dynamic Bayesian Network (DBN) whose variables consist of the nonlocal features as well as their respective dependent state features and dependent history features with links corresponding to the dependence relationships. This *influence DBN* encodes the probability distributions of all of the outside influences affecting each agent. Once all of an agent's incoming influences (exerted by its peers) have been decided, the agent can incorporate this probability information into a local POMDP model[1] with which to compute optimal decisions. The agent constructs the local POMDP by combining the TD-POMDP local transition function (terms 1 and 2 of Equation 1) with the probabilities of nonlocally-controlled feature transitions $Pr\left(\bar{n}_j^{t+1}|...\right)$ encoded (as conditional probabilities) by the influence DBN.[2] Rewards and observations for this local POMDP are dictated by the TD-POMDP local reward function R_i and local observation function O_i, respectively.

As agents' interactions become more complicated, more variables are needed to encode their effects. However, due to TD-POMDP structure, the DBN need contain only those critical variables that link the agents' POMDPs together.

Proposition 1. *For any given TD-POMDP, the influence $\Gamma_{\pi_i}(n_{jx})$ of agent i's fixed policy π_i on agent j's nonlocal feature n_{jx} need only be conditioned on histories (denoted $\vec{m}_j$) of mutually-modeled features $\bar{m}_j = \bigcup_{k\neq j}(s_j \cap s_k)$.*

Proof Sketch. The proof of this proposition emerges from the derivation of a *belief-state* (as in [9]) representation for TD-POMDP agent j's best-response POMDP:

$$b_j^t = \left\langle Pr\left(s_j^t, \vec{m}_j^{\,t-1}|\vec{a}_j^{\,t-1}, \vec{o}_j^{\,t}\right), \forall s_j^t, \vec{m}_j^{\,t-1}\right\rangle \tag{2}$$

We can derive an equation for the components (each indexed by one value of $\left\langle s_j^{t+1}, \vec{m}_j^{\,t}\right\rangle$) of j's belief-state b_j^{t+1} at time $t+1$ by applications of Bayes' rule, conditional probability, and the factored TD-POMDP local observation function $O_i()$:

$$\begin{aligned} b_j^{t+1}\left(s_j^{t+1}, \vec{m}_j^{\,t}\right) &= Pr\left(s_j^{t+1}, \vec{m}_j^{\,t}|\vec{a}_j^{\,t}, \vec{o}_j^{\,t+1}\right) \\ &= \frac{O_j\left(o_j^{t+1}|a_j^t, s_j^{t+1}\right)\sum_{s_j^t} Pr\left(s_j^{t+1}|s_j^t, \vec{m}_j^{\,t-1}, \vec{a}_j^{\,t}, \vec{o}_j^{\,t}\right) b_j^t\left(s_j^t, \vec{m}_j^{\,t-1}\right)}{Pr\left(o_j^{t+1}|\vec{a}_j^{\,t-1}, \vec{o}_j^{\,t}, a_j^t\right)} \text{ : a normalizing constant} \end{aligned} \tag{3}$$

Next, from conditional independence relationships implied by the factored transitions (Equation 1) of the TD-POMDP:

$$= \frac{O_j(...)\sum_{s_j^t - \bar{m}_j^t} Pr\left(\vec{l}_j^{\,t+1}|s_j^t, a_j^t\right) Pr\left(\vec{u}_j^{\,t+1}|s_j^t\right) Pr\left(\vec{n}_j^{\,t+1}|\vec{m}_j^{\,t}\right) b_j^t(...)}{Pr\left(o_j^{t+1}|\vec{a}_j^{\,t-1}, \vec{o}_j^{\,t}, a_j^t\right)} \text{ : a normalizing constant} \tag{4}$$

The DBN in Figure 3 provides a graphical depiction of the dependencies between various groupings of j's mutually-modeled ($\bar{m}_j$), exclusively-modeled ($s_{\subseteq j}$), and unmodeled ($s_{\neq j}$) world state features. (In contrast to the *influence DBN*, this DBN includes all features of the world state.) The

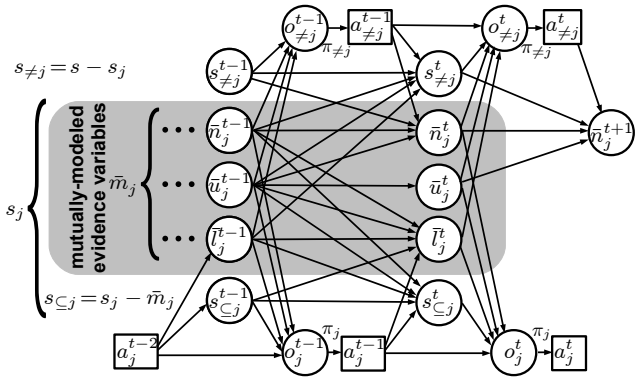


Figure 3: Independence of $\bar{n}_j^{t+1}$ given evidence $\vec{m}_j^t$.

existence of an arrow from node a to node b indicates that b's

[1]This POMDP is an extension and generalization of the local model that we have described in detail in past work [15].

[2]In the case of history-dependent influences, the POMDP's belief-state space is augmented to capture the necessary history as detailed in the proof sketch of Proposition 1 (a *small* compromise for avoiding modeling the full multiagent belief state as in [9]).

value directly depends on a's value; the absence of said arrow indicates conditional independence. The features $\vec{m}_j^t$ appear in a shaded region to illustrate the d-separation of agent j's own action and observation history from future nonlocal feature values given belief-state evidence $\vec{m}_j^t$, which justifies the reduction $Pr\left(\bar{n}_j^{t+1}|s_j^t, \vec{m}_j^{t-1}, \vec{a}_j^t, \vec{o}_j^t\right) = Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$. All paths originating from features which j can observe (given the TD-POMDP factored local observations) and ending at $\bar{n}_j^{t+1}$ must pass through the shaded region. Consequently, $\vec{m}_j^t$ is a sufficient summary of j's observations for making predictions about $\bar{n}_j^{t+1}$.

Equation 4 has three important consequences. First, agent j can compute its next belief state using only its peers' policies, the TD-POMDP model, its previous belief state, and its latest action-observation pair (without having to remember the entire history of observations). Second, the denominator of Equation 4 (which is simply a summation of the numerators across all belief-state components) allows the agent to compute the probability of its next observation (given its current action) using only its peers' policies, the TD-POMDP model, and its previous belief state. These two consequences by themselves prove sufficiency of the belief state representation for optimal decision-making. Third, the only term in the numerator of Equation 4 that depends upon peers' fixed policies is $Pr\left(\bar{n}_j^{t+1}|\vec{m}_j^t\right)$, and hence this distribution is a sufficient summary of all peers' policies. □

COROLLARY 1. *The influence DBN grows with the number of shared state features irrespective of the number of local state features and irrespective of the number of agents.*

The implication of Proposition 1 is that the local POMDP can be compactly augmented with histories of only those state features that are shared among agents. Moreover, the complexity with which an agent models a peer is controlled by its tightness of coupling and *not* by the complexity of the peer's behavior. Efficiency and compactness of local models is significant because they will be solved repeatedly over the course of a distributed policy-space search.

Another way to interpret this result is to relate it to the relative complexity of the influence space, which is the number of possible influence DBNs. Each DBN is effectively a HMM whose state is made up of shared features (and histories of shared features) of the TD-POMDP world state. Given Proposition 1's restrictions on feature inclusion, the space of DBNs should scale more gracefully than the joint policy space with the number of world features and number of agents (under the assumption that agents remain weakly-coupled), a claim that is supported by our empirical results.

4. SEARCHING THE INFLUENCE SPACE

Given the compact representations of influence that we have developed, agents can generate the optimal joint policy by searching through the space of influences and computing optimal local policies with respect to each. Drawing inspiration from Nair *et al*'s GOA method [10] for searching through the policy space, here we describe a general algorithm for searching the (TD-POMDP) influence space.

Algorithm 1 outlines the skeleton of a depth-first search that enumerates all feasible values, one influence at a time, as it descends from root to leaf. At the root of the search tree, influences are considered that are independent of all of other influences. And at lower depths, feasible influence values are determined by incorporating any higher-up influence values on which they depend. This property is ensured given any total ordering of agents (denoted *ordering* in Algorithm 1) that maintains the partial order of the acyclic interaction digraph.[3] At each node of the depth-first search, procedure OIS() is called on agent i, who invokes the next agent's OIS() procedure execution and later returns its result to the previous agent. Thus, the algorithm is decentralized, but is initiated by a root agent whose influence does not depend on its peers.

Algorithm 1 Optimal Influence-Space Search

```
OIS(i, ordering, DBN, vals)
 1: POMDP_i ← BUILDBESTRESPONSEMODEL(DBN)
 2: if i = LASTAGENT(ordering) then
 3:     ⟨vals[i], π_i⟩ ← EVALUATE(POMDP_i)
 4:     return ⟨vals, DBN⟩
 5: end if
 6: j ← NEXTAGENT(i, ordering)
 7: I ← GENERATEFEASIBLEINFLUENCES(POMDP_i)
 8: bestVal ← −∞
 9: bestDBN ← nil
10: for each influence_i ∈ I do
11:     thisVals ← COPY(vals)
12:     ⟨thisVals[i], π_i⟩ ← EVALUATE(POMDP_i, influence_i)
13:     DBN_i ← COMBINE(DBN, influence_i)
14:     ⟨thisVals, DBN_child⟩ ← OIS(j, ordering, DBN_i, thisVals)
15:     jointVal ← COMPOSEJOINTREWARD(thisVals)
16:     if jointVal > bestVal then
17:         vals ← thisVal
18:         bestDBN ← DBN_child
19:     end if
20: end for
21: return ⟨vals, bestDBN⟩
```

The search begins with the call OIS($root, ordering, \emptyset, \infty$), prompting the first agent to build its (independent) local POMDP (line 1) and to generate all of the feasible combinations of its outgoing influence values (line 7), each in the form of a DBN (as described in Section 3.2). A naïve implementation of GENERATEFEASIBLEINFLUENCES() would simply enumerate all local policies, and for each, compute the requisite conditional probabilities that the policy implies and incorporate them into a DBN model. In Section 4.2, we suggest a more sophisticated generation scheme. The root creates a branch for each feasible influence DBN, passing down the influence along with the value of the best local policy that achieves the DBN's influences (computed using EVALUATE()). Each such call to OIS() prompts the next agent to construct a local POMDP in response to the root's influence (as described in Section 3.2), compute its feasible influences and values, and pass those on to the next agent.

At the root of the tree, the DBN starts out as empty and gradually grows as it travels down the tree, each iteration accumulating another agent's fixed influences. The agent at the leaf level of the tree does not influence others, so simply computes a best response to all of the fixed influences and

[3]In the event of a cyclic interaction digraph, we can still ensure this property, but with modifications to Algorithm 1. Note that although the digraph may contain cycles, the influence DBN itself cannot contain cycles (due to the non-concurrency of agent influences described following equation 1). We can therefore separate an agent's time-indexed influence variables into those {*dependent upon*, *independent of*} another influence, and reason about those sets at separate levels of the search tree. If we separate influence variables sufficiently, cyclic dependence can be avoided as we progress down the search tree.

passes up its policy value (lines 1-4). Local utility values get passed down and washed back up so that intermediate agents can evaluate them via COMBINE() (which composes expected local utilities into expected joint utilities). In this manner, the best outgoing influence values get chosen at each level of the tree and returned to the root. When the search completes, the result is an optimal influence-space point: a DBN that encodes the feasible influence settings that optimally coordinate the team of agents. As a post-processing step, the optimal joint policy is formed by computing all agents' best-response policies (via BUILDBESTRESPONSEMODEL() and EVALUATE()) in response to the optimal influence point returned by the search.

4.1 Approximation Techniques

An attractive trait of this framework is the natural accommodation of approximation methods that comes with representing influences as probability distributions. One straightforward technique is to discretize the DBN space, grouping probability values that are within ϵ of each other so as to guarantee that a distribution is found whose influences are close to that of the optimal influence. A second technique is to approximate the structure of the influence DBN. For a given influence, feature selection methods could be used to remove all but the most valuable dependencies of each influence, thereby sacrificing completeness of the abstraction for a reduction in search space.

4.2 Efficient Generation and Evaluation of Feasible State-Dependent Influences

One commonly-studied subclass of TD-POMDPs involves state-dependent, history-independent influences whereby (in particular) agents coordinate the timings of interdependent task executions [2, 4, 16]. To reason about these influences, agents can utilize a constrained policy formulation technique (that we have introduced more formally in past work [14]) based upon the dual-form Linear Program (LP) for solving Markov decision processes. Under the assumption of a non-recurrent state space, the dual form represents the probabilities of reaching states as *occupancy measure* variables, which are exactly what we need to represent the influences that an agent exerts. For instance, $Pr\,(\textit{site-C-prepared}|\textit{time} = 4)$ corresponds to the probability of R5 (in Figure 2) entering any state for which $\textit{time} = 4$ and $\textit{site-C-prepared} = \textit{true}$, which is the summation of LP occupancy measures associated with these states (or belief-states, for POMDPs). The agent can use this LP method to (1) calculate its outgoing influence given any policy, (2) determine whether a given influence is feasible, and if so (3) compute the optimal local policy that is constrained to exert that influence [14].

We devise a useful extension: an LP that *finds* relevant influence points. For a given influence parameter p, we would like the agent to find all feasible values for that parameter (achievable by any deterministic policy). This can be accomplished by solving a series of (MI)LPs, each of which looks for a (pure) policy that constrains the parameter value to lie within some interval: $p_{\min} < p < p_{\max}$ (starting with interval $[0, 1]$). If the LP returns a solution, the agent has simultaneously found a new influence ($p = p_0$) and computed a policy that exerts that influence, subsequently uncovering two new intervals $\{(p_{\min}, p_0), (p_0, p_{\max})\}$ to explore. If the LP returns "no solution" for a particular interval, there is no feasible influence within that range. By divide and conquer, the agent can find all influences or stop the search once a desired resolution has been reached (by discarding intervals smaller than ϵ). In general, this method allows agents to generate all of their feasible influences without exhaustively enumerating and evaluating all of their policies.

5. EMPIRICAL RESULTS

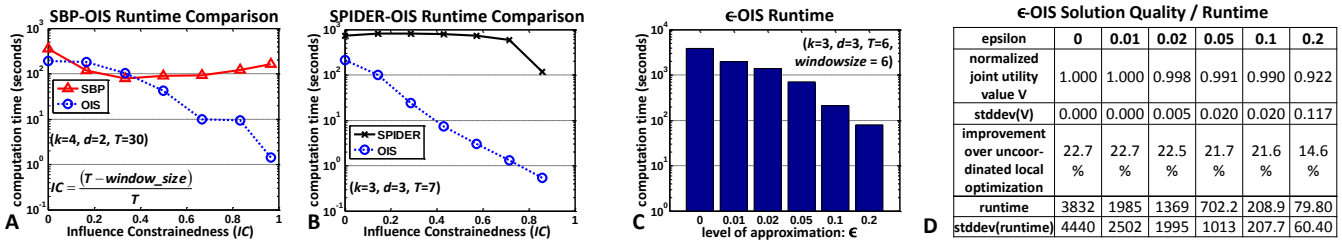
We present an initial empirical study analyzing the computational efficiency of our framework. Results marked "OIS" correspond to our implementation of Algorithm 1 that follows the LP-based influence generation approach (discussed previously). We compare influence-space search to two state-of-the-art optimal policy search methods: (1) a Separable Bilinear Programming ("SBP") algorithm [8] for problems of the same nature as ED-DEC-MDPs [2] and (2) an implementation of "SPIDER" [13] designed to find optimal policies for two-agent problems with transition dependencies [7]. Both implementations were graciously supplied by their respective authors to improve the fairness of comparison.

Plots 4A and 4B evaluate the claim that influence-space search can exploit weak coupling to find optimal solutions more efficiently than policy-space search. These two plots compare OIS with SBP and SPIDER, respectively, on sets of 25 randomly-generated 2-agent problems from the planetary exploration domain, each of which contains a single interaction whereby a task of a satellite agent influences the outcome of a task of a rover agent.[4] For each problem, the *influence constrainedness* was varied by systematically decreasing the window size (from T to 1) of the influencing task. While the computation time[5] (plotted on a logarithmic scale) taken by SBP and SPIDER to generate optimal solutions remains relatively flat, OIS becomes significantly faster as influences are increasingly constrained. This result, although preliminary, demonstrates that influence-based abstraction can take great advantage of weak agent coupling but might prove less valuable in tightly-coupled problems.

The third experiment (shown in Figure 4C-D) evaluates OIS on a set of 10 larger problems (where SBP and SPIDER were infeasible), each with 4 agents connected by a chain of influences. One of agent 1's tasks (chosen at random) influences one of agent 2's tasks, and one of agent 2's tasks influences one of agent 3's tasks, etc. We compare optimal OIS with "ϵ-OIS", which discretizes probabilities with a step size of ϵ in the probability space. The quality and runtime figures indicate that, for this space of problems, influence-space approximation can achieve substantial computational savings at the expense of very little solution quality. Additionally, this result is notable for demonstrating tractability of *optimal* joint policy formulation on a size of problems (4 agents, 6 time units) that has been beyond the reach of the prior approaches to solving transition-dependent problems (with relatively unrestricted local POMDP structure).

[4] As denoted in Figure 4, the agents each have k tasks, each with d randomly-selected durations (with duration probabilities generated uniformly at random) and randomly-selected outcome qualities executed for a horizon of T time units. Because the implementations of SBP and SPIDER were tailored to specific domains, we could not run them on the same problems. For instance, the SBP implementation assumes that agents are not able to wait between task executions. Both domains assume partial observability such that agents can directly observe all of their individually-controlled tasks, but not the outcomes of the tasks that influence them.

[5] All computation was performed on a single shared CPU.



ϵ-OIS Solution Quality / Runtime

epsilon	0	0.01	0.02	0.05	0.1	0.2
normalized joint utility value V	1.000	1.000	0.998	0.991	0.990	0.922
stddev(V)	0.000	0.000	0.005	0.020	0.020	0.117
improvement over uncoordinated local optimization	22.7 %	22.7 %	22.5 %	21.7 %	21.6 %	14.6 %
runtime	3832	1985	1369	702.2	208.9	79.80
stddev(runtime)	4440	2502	1995	1013	207.7	60.40

Figure 4: Empirical Evaluation of Influence-space Search.

6. CONCLUSIONS

This paper contributes a formal framework that characterizes a broad array of weakly-coupled agents' influences and abstracts them from the agents' policies. Although previous methods have abstracted specialized flavors of transition influence [2, 16] or used abstraction to guide heuristic search [14], the comprehensive model we have devised places these conceptually-related approaches into a unified perspective. As a foundation for our framework, we have introduced a TD-POMDP class whose factored transition structure engenders a decoupling of agents' subproblems and a compact model of nonlocal influence. Inspired by the successful scaling of the (transition-independent, reward-dependent) ND-POMDP model [10, 13] to teams of many agents, we have cast the TD-POMDP joint policy formulation problem as one of local best-response search.

Prior to this work, there have been few results shown in scaling transition-dependent problems to teams of several agents whilst maintaining optimality. Our compactness result suggests that, for weakly-coupled transition-dependent problems, agents can gain traction by reasoning in an abstract influence space instead of a joint policy space. We give evidence supporting this claim in our initial empirical results, where we have demonstrated superior efficiency of optimal joint policy generation through an influence-space search method on random instances of a class of commonly-studied weakly-coupled problems. But more importantly, our general influence-based framework offers the building blocks for more advanced algorithms, and a promising direction for researchers seeking to apply Dec-POMDPs to teams of many weakly-coupled transition-dependent agents. Future work includes a more comprehensive investigation into problem characteristics (e.g. digraph topology and influence type) that impact the performance of influence-space search, and further development and comparison of approximate flavors of OIS with other approximate approaches (such as TREMOR [12]).

7. ACKNOWLEDGEMENTS

This material is based upon work supported, in part, by AFOSR under Contract No. FA9550-07-1-0262. We thank Hala Mostafa and Janusz Marecki for supplying us with the SBP and SPIDER implementations, respectively, and the anonymous reviewers (from AAMAS, ICAPS, and MSDM) for their thoughtful comments and suggestions.

8. REFERENCES

[1] M. Allen and S. Zilberstein. Agent influence as a predictor of difficulty for decentralized problem-solving. In *AAAI*, pages 688–693, Vancouver, Canada, 2007.

[2] R. Becker, S. Zilberstein, and V. Lesser. Decentralized Markov decision processes with event-driven interactions. In *AAMAS*, pages 302–309, 2004.

[3] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Solving transition independent decentralized Markov Decision Processes. *JAIR*, 22:423–455, 2004.

[4] A. Beynier and A. Mouaddib. A polynomial algorithm for decentralized Markov decision processes with temporal constraints. In *AAMAS*, pages 963–969, 2005.

[5] K. Decker. TAEMS: A framework for environment centered analysis & design of coordination mechanisms. In *Foundations of Distributed Artificial Intelligence, Ch. 16*, pages 429–448, 1996.

[6] C. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *JAIR*, 22:143–174, 2004.

[7] J. Marecki and M. Tambe. Planning with continuous resources for agent teams. In *AAMAS*, pages 1089–1096, 2009.

[8] H. Mostafa and V. Lesser. Offline planning for communication by exploiting structured interactions in decentralized MDPs. In *Proceedings of IAT-09*, pages 193–200, Milan, Italy, 2009.

[9] R. Nair, M. Tambe, M. Yokoo, D. V. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, pages 705–711, 2003.

[10] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed cnstrnt. optim. and POMDPs. *AAAI*, pages 133–139, 2005.

[11] F. Oliehoek, M. Spaan, S. Whiteson, and N. Vlassis. Exploiting locality of interaction in factored dec-pomdps. In *AAMAS*, pages 517–524, 2008.

[12] 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*, 2009.

[13] P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, and M. Yokoo. Letting loose a spider on a network of POMDPs: generating quality guaranteed policies. In *AAMAS*, pages 817–824, 2007.

[14] S. Witwicki and E. Durfee. Commitment-driven distributed joint policy search. In *AAMAS*, 2007.

[15] S. Witwicki and E. Durfee. Commitment-based service coordination. *IJAOSE*, 3(1):59–87, 2009.

[16] S. Witwicki and E. Durfee. Flexible approximation of structured interactions in decentralized Markov decision processes. In *AAMAS*, pages 1251–1252, 2009.

Coordination-VCG Mechanism for Controlling Multiagent Planning

Ayman Ghoneim
The Australian National University and NICTA
Canberra ACT, Australia
ayman.ghoneim@anu.edu.au, ayman.ghoneim@nicta.com.au

ABSTRACT

Multiagent planning has numerous real life applications, but it has not been widely applied because controlling selfish agents who have conflicting interests is non-trivial, especially in informationally decentralized environments where agents have their own private information. Agents can strategically misrepresent their private information, aiming for a more biased global plan toward their own interests. In such settings, mechanism design is considered a useful tool for defining the rules that will govern the agents' interactions and their reporting strategies, while guaranteeing some desirable outcome for the whole system. Classical mechanism design normally assumes only a declaration phase, where the main focus is on extracting the agents' private information truthfully. However, multiagent planning settings have an additional execution phase, where the agents will execute their plans, and thus, mechanisms controlling such settings must take both phases into consideration. In this paper, we introduce a truthful Coordination-VCG mechanism which prevents agents from lying about their private information in multiagent planning settings. The mechanism establishes truthfulness by mapping the agents' payments back to their local planning problem in the form of reward functions to influence their behavior during execution, in order to implement the optimal determined global plan. We discuss budget balance in relation to the proposed mechanism, showing that it can achieve pareto-optimality in special case where pairwise interactions between agents within the planning setting are independent. Finally, we provide one possible implementation for the proposed mechanism based on Markov decision processes.

Categories and Subject Descriptors

J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics*; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Economics, Theory, Algorithms

Keywords

Multiagent Planning, Mechanism design, VCG mechanism

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

1. INTRODUCTION

Most multiagent planning studies focus on finding a computationally efficient solution by exploiting the problem structure - whether this solution is computed in a centralized or decentralized way. And usually assuming cooperative or altruistic agents operating under complete information, who can be viewed as acting on behalf of one individual aiming to maximize his overall utility while knowing everything about other agents. For such setups, several approaches have been proposed, including a single planner creating plans for several agents [14, 6], a local coordination procedure for each agent [7], and plan modification and merging [1]. However, in more realistic situations, rational and selfish agents may represent companies or independent entities competing for common opportunities. The agents will be maximizing their own utility, and usually will have *conflicting interests*. In such situations, the coordination mechanism will rely on *game-theoretic* concepts as a powerful tool for capturing the complex interactions among these agents, while the informational assumptions will still play considerable role.

In this study, we are not concerned with finding a computationally efficient solution for the multiagent planning problem. We assume that the problem can be solved efficiently in a centralized way by some unbiased central authority, which in turn will guide the agents through their courses of action. However, we are more concerned with multiagent planning settings where selfish agents are involved in informationally decentralized environments, meaning, the information required to solve the multiagent planning problem is distributed among selfish agents. Such settings require different treatment than those adopted with selfish planning agents operating under complete information. Each agent has its own *private information* which express its planning problem (e.g. its initial state, possible actions, actions' associated costs, goals and the goals' associated rewards), while knowing nothing about other agents. Knowing that the central authority - which knows nothing in advance about the agents - will determine a global plan based on the agents' reported information, the selfish agents may be engaged in a *strategic misrepresentation* of their private information, aiming for a global plan biased toward their own interests.

For such contexts, *mechanism design* is the fundamental tool used in designing the appropriate *rules* that regulate the agents' interactions to guarantee some overall desirable outcome. Classical mechanism design mainly focuses on the *declaration phase*, where the aim is to extract the agents' private information truthfully, by eliminating any incentives for them to lie. However, while designing mechanisms for controlling multiagent planning and other similar applications (e.g. task scheduling), after the declaration phase there is an *execution phase* where the agents will execute their agreed upon plans, and which provides an additional dimension for the agents' misrepresented information. Although mechanism de-

sign is widely investigated and applied to multiagent systems (e.g. auctions and resource allocation), there are very few studies [15, 17] that address the use of mechanism design in controlling multiagent planning. Previous efforts investigated the applicability of the well-known VCG mechanism for controlling multiagent planning settings, showing its failure in handling over-reporting actions (i.e. reporting actions that an agent doesn't possess), and suggested deposits and compensating schemas to avoid over-reporting and handle the execution phase.

In this study, we present a novel mechanism - which belongs to the VCG family of mechanisms - for controlling multiagent planning. The idea behind our mechanism is intuitive and stems from the consideration of common characteristics of the multiagent planning problem, where the mechanism will reformulate the agents' payments as reward functions - mapped to their planning state space - that will influence the agents' planning decisions. The optimal plans decided by agents using these reward functions will be optimal from the global view point. We illustrate the truthfulness of the proposed mechanism, and show that it achieves pareto-optimality in multiagent planning settings where pairwise interactions between agents are independent. Moreover, we illustrate a possible implementation for our mechanism for practical use-cases.

In the next section, we cover the relevant definitions and concepts of mechanism design applied to multiagent planning, followed by related work and suggested modification for previous mechanism [15]. Then, we outline our proposed mechanism, demonstrating its conceptual and technical details. We then verify the useful properties attained by our mechanism, and illustrate a possible implementation for it using Markov decision processes as the local (global) planner used by the agents (mechanism), while showing how the agents' reward functions are designed. Finally, we conclude our work, and propose future directions.

2. MECHANISM DESIGN AND MULTIAGENT PLANNING

Mechanism design is a general methodology for designing mechanisms that help an arbitrator (i.e. some central authority) in resolving the conflict in hand - here, the multiagent planning setting. A mechanism simply dictates the rules that will be proposed to the agents. These rules will define the agents' strategies, and generally accommodate: *1)* The communication system (i.e. who communicates with whom, the language of communication, and what to communicate); *2)* The payment procedures (i.e. rewards and penalties from the system or between agents on top of those related to the agents' internal structure); and *3)* The characteristics of the desirable outcome (e.g. maximizing revenue or social welfare, achieving equity and/or preserving a good environment) and how to reach there (e.g. coordination among agents). Assuming selfish agents [1], the mechanism's chosen outcome (global plan) is implemented (i.e. will be reached) under some game-theoretic solution concept. Different mechanisms then can be ranked according to the arbitrator's goal(s) (e.g. social welfare or revenue).

Mechanism design has passed some main stages in its development, with initial steps taken by Leonid Hurwicz [9] by introducing a communication system among agents for commodity exchange that laid down the crucial concepts and definitions of the process. Hurwicz [10] emphasized that *incentive compatibility* is an important characteristic that needs to be possessed by mechanisms, meaning, agents must not have any incentive to misrepresent their private information while communicating with the arbitrator. This can be achieved if the agent's utility when reporting truthfully is equal or higher than the utility when reporting untruthfully, thus there will be no point of misrepresenting its private information. Defining the *revelation principle* - by several discoverers including Roger Myerson [11] - was an important milestone in the mechanism design development, where the principle states that the performance of any mechanism can be replicated by a *direct revelation* mechanism, in which the agents' reporting strategies will be simply reporting their types. Thus, we will focus our attention here on developing a direct revelation mechanism. Formally, we can define a mechanism design setting for multiagent planning as follows:

Definition 1: *A* **mechanism design setting for multiagent planning** *consists of:*

- *A set of m agents, denoted by α, and a central authority that knows nothing in advance about the private information of the agents;*
- *Each agent i has its own private information regarding its planning problem. Information concerning the agent's initial state, actions and goal(s) imply the space of possible plans of the agent i, denoted by Π_i. While information concerning the actions' costs and goals' rewards imply the agent's value for any plan $\pi_i \in \Pi_i$, which can be expressed by the difference between the agent's reward from its achieved goal(s) and the cost of its actions, denoted by the valuation function $v_i(\pi_i)$. Both kinds of information represent the agent's type $\theta_i \in \Theta_i$, where Θ_i is the set of all the agent's possible types;*
- *An agent can lie in three ways [15] while reporting its type: 1) Lie about its value function, i.e. actions' costs and/or goals' rewards; 2) Under-report its available actions - by declaring an infinite cost for an action, this can be similarly viewed as the first type of lying, this implies unreported possible plans; and 3) Over-report its available actions, which implies reporting some imaginary plans. We will call the first two types of lying traditional lying;*
- *Based on the agents' reported types, the central authority will choose a global plan π_g, which can be viewed as m local plans for the participating agents (i.e. $\pi_g = (\pi_g^1, \ldots, \pi_g^m)$), where these local plans are desirable from the system's view point. The global plan π_g is chosen from the space of all "possible" global plans for the multiagent setting, denoted by $\Pi_G \subseteq \Pi_1 \times \ldots \times \Pi_m$, where $\pi_g \in \Pi_G$;*
- *Agent i valuation function will map the chosen π_g together with the agent's true type θ_i to a real number quantifying the agent's value under the selected global plan, denoted by $v_i(\pi_g, \theta_i)$, which is equivalent to $v_i(\pi_g^i)$;*
- *The selected global plan maximizes a social welfare (i.e. utilitarian) function f, defined by the summation of all the agents' values under the selected global plan π_g and the agents' reported types $\theta = (\theta_1, \ldots, \theta_m)$, i.e. the selected $\pi_g = argmax_{\pi_g \in \Pi_g} \sum_{i \in \alpha} v_i(\pi_g, \theta_i)$.*

A direct revelation mechanism usually uses payments as an external factor on top of the agents' valuations - which depend on their internal structure and capabilities - for the selected global plan π_g to align the agents' incentives with those of the system while reporting their private information in the declaration phase. However, this will not provide the full picture for a successful execution

[1]Mechanism design can also be considered for cooperative agents, where the main concern will be designing informationally efficient communication systems.

in a multiagent planning setting. The mechanism still needs to control the agents while executing their agreed upon plans in the execution phase. This implies that payments in multiagent planning are needed to control the agents in both phases. Using payments may raise some concerns about the applicability of such mechanisms for controlling multiagent planning in practise, for instance, in Robotics domains where monetary payments are not possible. In spite of the validity of these concerns regarding some application domains, there remain numerous domains where payments are meaningful, such as supply chains, logistics, road and air traffic control, among others. We will use a deterministic mechanism which determines a single global plan rather than a randomized one - ordinarily used for theoretical purposes, which defines a probability distribution over the possible global plans' space Π_G. A deterministic mechanism with payments that maximizes the social welfare is defined as follows:

Definition 2: A deterministic direct revelation mechanism with payments *has a social welfare function f that, given any reported vector of types $\theta = (\theta_1, \ldots, \theta_m) \in \Theta_1 \times \ldots \times \Theta_m$, produces a single global plan π_g that maximizes f, and a payment function $p_i : \Theta_1 \times \ldots \times \Theta_m \rightarrow \mathbb{R}$ for each agent i, where $p_i(\theta)$ denotes the payment made by agent i. Thus, the overall utility of agent i will be expressed by $u_i(\pi_g, \theta_i) = v_i(\pi_g, \theta_i) - p_i(\theta)$.*

Allocative efficiency and *budget balance* are two desirable properties that our proposed mechanism possesses. The *allocative efficiency* property is attained by choosing f as a function that maximizes the social welfare (definition 1). On the other hand, a mechanism is said to be budget balanced if no net payments transfer out of the system or into the system (i.e. for all agents' types $\theta = (\theta_1, \ldots, \theta_m)$, $\sum_{i \in \alpha} p_i(\theta) = 0$). These two properties together imply *pareto optimality*, which means that under the mechanism's chosen global plan π_g, no agent can be better off without making other agents worse. The mechanism normally assumes that the chosen outcome π_g will be reached in an equilibrium based on some game-theoretic solution concept such as the *Nash* equilibrium, the *Bayesian-Nash* equilibrium or the *Dominant Strategy* equilibrium. The incentive compatibility can be defined for the Nash equilibrium as follows:

Definition 3: *Given a deterministic mechanism with payments, the mechanism implements a* **Nash equilibrium** *if reporting truthfully is always the optimal strategy for an agent given that other agents' are reporting truthfully. Meaning, for any involved agent i, if its true type is θ_i, its utility when reporting θ_i is greater than or equal to its utility when reporting any other type $\hat{\theta_i}$ (other than its true type θ_i), for any chosen global plan π_g, given that other agents are truthfully reporting their types.*

Of the known and most used family of mechanisms are the Vickrey-Clarke-Groves (VCG) mechanisms [16, 5, 8] defined as follows:

Definition 4: *A direct revelation mechanism is called a VCG mechanism if: 1) f maximizes social welfare (i.e. utilitarian function); and 2) Agent i payment to the mechanism is determined by $p_i(\theta) = h_i(\theta_{-i}) - \sum_{j \in \alpha, j \neq i} v_j(\pi_g, \theta_j)$, where $h_i(\theta_{-i})$ is an arbitrary function based on the declared types of all the agents except agent i, i.e. $\theta_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_m)$.*

VCG mechanisms assume that the agents have *quasi-linear* preferences, where the agent's utility can be represented in terms of its valuation of the selected outcome and its payment, which make it straightforward to transfer utilities across agents. VCG mechanisms are *allocative-efficient* as they maximize a social welfare function. The selection of the function $h_i(\theta_{-i})$ leads to the description of the family of mechanisms. According to the VCG mechanism payment function - with $h_i(\theta_{-i}) = 0$, the mechanism will pay each agent an equivalent amount to all the other agents' valuations of the selected outcome. The effect of the VCG payment function p_i is to *internalize the externalities* placed on the other agents due to the reported information of agent i. This aligns the agents' incentives with the system's goal of achieving efficient allocation, where an agent wants the mechanism to select the system's best outcome based on all the reported information in order to maximize its received payment.

3. RELATED WORK AND MODIFIED DEPOSIT-VCG

In informationally decentralized multiagent planning settings, where agents are selfish, hold their private information and usually have conflicting interests, the central authority aims to propose a mechanism for the agents that leaves no incentives for them to lie about their private information in the declaration phase, and makes sure that the determined global plan is implemented in the execution phase. In what follows, we will discuss previous efforts in controlling multiagent planning using mechanism design, the considered assumptions and concerns. Then, we will propose our Coordination-VCG mechanism, illustrating the conceptual and technical properties that the mechanism possesses.

van der Krogt et al. [15] investigated the usage of a VCG mechanism where a central authority (using an optimal global planner) will plan for all the involved agents, aiming to maximize the sum of all the agents' valuations, while the payments will be determined according to the VCG mechanism with $h_i(\theta_{-i}) = 0$ (Definition 5). It was shown [15] that the VCG mechanism is incentive compatible for traditional lying but not for over-reporting. Obviously, for traditional lying (which may affect the selected global plan, and thus, decreases other agents' valuations), the agent will have a corresponding decrease in its received payment from the mechanism, as its payment is equivalent to other agents' valuations.

The over-reporting type of lying opens debate on the suitable modeling assumptions to be considered for the execution phase. If an agent over-reports its actions, and these over-reported actions were included in the global plan which will be infeasible, then some agents will fail in executing their plans. This opens the question of whether the payments will be calculated and paid before or after the execution phase. If the payments are made before the global plan execution, then an agent may have some incentive to over-report, as this may increase other agents' valuations of the resultant "infeasible" global plan, and thus, an increase in the payment received by the agent from the mechanism. However, if the mechanism determines the global plan based on the agents' reported information, under the assumptions that the agents are guaranteed to make their payments after the execution, and any failure that may occur during the execution phase can be detected (i.e. the agents' reported information are verified), then calculating the payments and handling them after the execution is sufficient to prevent agents from over-reporting. Similar assumptions were used in a task scheduling context [13].

It was assumed in [15, 17] that the agents' payments are calculated and paid before the execution phase, and thus, a VCG mechanism cannot prevent over-reporting. To avoid over-reporting, van der Krogt et al. [15] proposed the Deposit-VCG mechanism in which each agent will pay a deposit equivalent to all the rewards gained by all agents (including itself) from achieving their goals, determined by $r(G) = \sum_{i \in \alpha} \sum_{g_i \in G_i} r_i(g_i)$, where $r_i(g_i)$ is the reward that agent i will get from achieving one goal g_i that belongs to its set of goals G_i. The amount of this deposit is the minimum [15] when considering the worst case scenario where none of the agents succeed in achieving any goals due to the untruthful

reported information. If the execution of the global plan fails due to some agent's untruthful information, this agent will not get its deposit back, assuming that lying can be detected during the execution. It was stated that: *"Since the separate deposit stage does not enlarge the strategy space of the agents, it is straightforward to see that if the agents are truthful under the VCG mechanism, they will not be better off by lying under the Deposit-VCG mechanism" ... [15].* This would entail that the deposit-VCG is truthful whenever the VCG mechanism is truthful, and is truthful for multiagent planning settings as it can prevent over-reporting (Proposition 7 and Theorem 8 [15]).

Although that the idea of using deposits has its merits, the way of computing the deposits is contradicting to the VCG definition, as any additional part of the VCG payment function must follow the $h_i(\theta_{-i})$ function definition, in which, it must depend on the declared types of all the agents except agent i. But, it is clear that the deposit depends on the declared type of agent i, as it depends on the declared goals and their associated reward functions of agent i. Thus, when an agent considers the deposit while maximizing its utility function, this additional deposit term (whether these deposits are made in a separate stage or not) will alter the declaration strategy space of the agent, and violates the VCG payments' definition, and thus, the VCG truthfulness property. This leads us to the following proposition:

Proposition 1: *The Deposit-VCG mechanism [15] is not truthful when the VCG mechanism is truthful.*

However, to still make use of the deposits idea, the deposits can be assumed to be huge (as in [17]) enough to overcome any harm caused to the system due to over-reporting. Or alternatively, we can (re)-define the deposit differently as $r(G) = \sum_{j\in\alpha, j\neq i} \sum_{g_j\in G_j} r_j(g_j)$, which satisfies the VCG payments' definition as it will not depend on the declaration of agent i. This modified Deposit-VCG is truthful at the Nash equilibrium, i.e., reporting truthfully is always the optimal strategy for an agent assuming that other agents will report truthfully as well.

Theorem 1: *A modified Deposit-VCG mechanism prevents traditional and over-reporting types of lying, and thus, truthful for multiagent planning settings.*

Proof. The utility of agent i under a VCG mechanism is

$$u_i(\pi_g, \theta_i) = v_i(\pi_g, \theta_i) - p_i(\theta) \tag{1}$$

where $p_i(\theta) = h_i(\theta_{-i}) - \sum_{j\in\alpha, j\neq i} v_j(\pi_g, \theta_j)$ according to the VCG definition. By substituting $p_i(\theta)$ in the agent's utility function, setting $h_i(\theta_{-i}) = 0$ and adding the modified deposit term, it becomes

$$\begin{aligned} u_i(\pi_g, \theta_i) &= v_i(\pi_g, \theta_i) \\ &+ \sum_{j\in\alpha, j\neq i} v_j(\pi_g, \theta_j) - \sum_{j\in\alpha, j\neq i} \sum_{g_j\in G_j} r_j(g_j) \end{aligned} \tag{2}$$

Recalling that the mechanism will choose a global plan that maximizes all the agents' valuations based on their reported types, agent i has control only over its declared type θ_i, and wants to maximize its utility function (eq.2). This can be done by maximizing the second term (i.e. its payments from the mechanism equivalent to other agents' valuations), and getting back (eliminating) the third one (i.e. the deposit term). The agent will have no incentive to practise traditional lying, as this may minimize other agents' valuations of the selected global plan, and thus, minimizes the potential payment received by the agent from the mechanism. This is obvious as the modified Deposit-VCG will be truthful whenever the VCG mechanism (which prevents traditional lying) is truthful.

For over-reporting, agent i knows that if it decided not to over-report, it will eliminate the deposit term (i.e. it will get deposit amount back after the execution phase). Note that agent i knows that its over-reported actions (if considered in the global plan) will be used (from the global plan perspective) in either helping other agents (case 1), or helping itself (case 2), or both. The agent simply needs to consider these cases in order to decide whether to over-report or not. For case 1, if agent i over-reports its actions, it will lose the deposit amount, but in the same time, it may increase the second term (i.e. other agents' valuation of the infeasible global plan). Any increase in the second term (i.e. the received payment) will be less than the deposit amount. This is obvious as the mechanism is maximizing the agents' valuations, and the deposit amount serves as an upper bound for the agents' valuations (i.e. the goals' reward of all other agents). While for case 2, over-reporting will not increase the received payments (the second term), and the agent still loses the deposit amount as it failed in executing its part in the global plan, even if this part is related to the agent itself. The only thing that over-reporting will cause in this case is to increase the agent's value of the selected global plan from the mechanism perspective (as the agent will be achieving more goals from the mechanism view), which will increase the mechanism payments (those depend on the value of agent i) for other agents. Thus, over-reporting in case 2 will not increase the agent utility. □

4. COORDINATION-VCG MECHANISM

A mechanism that uses deposits is still hard to implement for the two following reasons: *1)* it assumes implicitly that the agents' reported information can be verified (i.e. lying agents can be detected) during the execution phase in order to determine whether the agents will get their deposits back or not, which is a difficult assumption to hold especially for large number of agents; and 2- the deposit amount may be too large to be afforded by the participating agents. By assuming that the payments will be determined after the execution phase - similar to [13] in task scheduling - handles over-reporting, but again we are assuming the verifiability of the reported information, and moreover, the agents' payments after the execution must be guaranteed. In this section, we will introduce a new mechanism for controlling multiagent planning, which differs from the previously discussed efforts conceptually and technically.

For illustrating the intuition behind our proposed mechanism, we will consider the following general view for the multiagent planning problem [4]. Each involved agent has its own set of actions, which can be classified into local actions (i.e. those affect only the agent) and global actions (i.e. those affect other agents). Any global plan for a certain multiagent planning setting actually boils down to the coordinate points between the global actions of the involved agents, which encapsulate exploited positive interactions and/or resolved conflicts. However, it is worth mentioning that agents' local actions remain an influential factor in determining an overall optimal global plan. The coordination points between agents (when determined) can be expressed at the agent level by some commitment points in its local plan, in which this agent will be executing its global actions (those affect other agents) according to the agreed upon coordination scheme. Generally speaking, different approaches for solving the multiagent planning problem are in way or another related to how the coordination points between the involved agents are to be determined. Recalling that we assume that the global plan can be determined efficiently in a centralized manner, this is the only suitable approach to consider here while investigating direct revelation mechanisms (which assume that agents are only allowed to communicate with the central authority) for controlling multiagent planning.

Keeping this general view in mind, there are two possible scenarios, either the central authority - based on the agents' reported information - will determine a complete global plan which dictates a complete plan for each agent (scenario 1), or ideally [2], will determine only the coordination points (i.e. the commitment points for each agent) - if possible! (scenario 2). In the first scenario, and for the central authority to guarantee a successful execution of the determined global plan, it actually wants to assure that each agent will execute a certain complete local plan (on the agent's level), which conveys its role in the global plan. However, for the second scenario, the central authority only cares that the agents' global actions are executed according to the coordination points those represent the determined optimal global plan (i.e. each agent will execute the agreed upon actions at the commitment points of its local plan). Knowing that any actions that will be executed between these commitment points will only affect the agent, and that the agent is selfish (i.e. will maximize its value from its local plan), the central authority shouldn't care about what the agent will execute between these commitment points.

Our proposed mechanism belongs to the VCG family of mechanisms, where its essence idea is to map the agents' received payments (in form of some reward functions) to their local state space, and then, let the agents determine their own local plans considering these reward functions. Mechanism design - by definition - aims to define the rules that governs the agents' interactions by reshaping their strategy spaces, and under the rationality assumption, agents - while maximizing their own utilities - will behave in a desirable way from the system's point of view. By using a direct revelation mechanism, the agents' strategy space for the declaration phase is narrowed down to report their types. While for the execution phase, the reward functions can be viewed as the mechanism's influential tool that will control the agents during execution, and will guide them to local plans which are optimal from the global view point. For the second scenario, the agent's received payment will be mapped to its commitment points, then the agent can plan between these commitment points by filling up his local actions, which reduces the computational burden over the central authority. However, and up to our knowledge, such sophisticated procedures for determining the coordination points within a multiagent planning setting without generating the complete global plan are still under investigation [4]. Thus, we will consider the first scenario for our study [3], assuming that the mechanism will guide the agents through their complete local plans, which are optimal from the system view point based on the complete global plan. For this scenario, the agent re-calculation for its local plan is considered redundant, however, determining this plan is a single agent planning problem that will be solved on the agent level.

For budget balance issues (to be discussed later), we use the Clarke pivot rule [5] for defining $h_i(\theta_{-i})$ as $\sum_{j\in\alpha,j\neq i} v_j(\pi_g^{-i},\theta_j)$, where π_g^{-i} is the global plan computed without the reported information of agent i, and maximizes the valuations of all other agents (i.e. $\pi_g^{-i} = argmax_{\pi_g^{-i}\in\Pi_G^{-i}} \sum_{j\in\alpha,j\neq i} v_j(\pi_g^{-i},\theta_j)$). This $h_i(\theta_{-i})$ value will be paid by agent i to the mechanism. We can define our proposed mechanism, denoted by a Coordination-VCG mechanism, as follows:

Definition 5: *A* **Coordination-VCG mechanism** *is defined by the following steps:*

1. *The mechanism asks the agents to report their types* $\theta = (\theta_1,\ldots,\theta_m)$.
2. *The mechanism computes the global plan* π_g *that maximizes all agents' valuations based on* θ*, where* $\pi_g = argmax_{\pi_g\in\Pi_G} \sum_{i\in\alpha} v_i(\pi_g,\theta_i)$.
3. *The mechanism computes the payment for each agent* i *using* $p_i(\theta) = h_i(\theta_{-i}) - \sum_{j\in\alpha,j\neq i} v_j(\pi_g,\theta_j)$*, where* $h_i(\theta_{-i}) = \sum_{j\in\alpha,j\neq i} v_j(\pi_g^{-i},\theta_j)$ *and* $\pi_g^{-i} = argmax_{\pi_g^{-i}\in\Pi_G^{-i}} \sum_{j\in\alpha,j\neq i} v_j(\pi_g^{-i},\theta_j)$.
4. *The mechanism computes a reward function* r^i *for each agent* i *- defined on the state space of the local planning problem of the agent, where its cumulative amount (over the state space) is equivalent to the received payment (i.e.* $\sum_{j\in\alpha,j\neq i} v_j(\pi_g,\theta_j)$*) by the agent.*
5. *The mechanism reports back the computed reward functions to the agents.*

In the next section, we will discuss the truthfulness property of the proposed Coordination-VCG mechanism, and related budget balance issues. Then, we will provide a detailed possible implementation of our mechanism for multiagent planning, focusing mainly on how the agents' payments will be expressed by the reward function r^i (step 4) that will be used by the Coordination-VCG mechanism to influence the local plan of each agent i. Generally speaking, this function will distribute the amount of the payment received by an agent over its local state space, in a way that the agent's local optimal plan will be optimal from the system's view point.

5. COORDINATION-VCG TRUTHFULNESS AND BUDGET BALANCE

We will show that the Coordination-VCG is truthful at nash equilibrium for multiagent planning settings where traditional lying or over-reporting may occur.

Theorem 2: *A Coordination-VCG mechanism is truthful for traditional lying and/or over-reporting in multiagent planning.*

Proof. The utility of agent i under a Coordination-VCG mechanism is given by equation 1, where $p_i(\theta) = \sum_{j\in\alpha,j\neq i} v_j(\pi_g^{-i},\theta_j) - \sum_{j\in\alpha,j\neq i} v_j(\pi_g,\theta_j)$. By substituting $p_i(\theta)$ in equation 1, the utility function becomes

$$u_i(\theta_i,\pi_g) = v_i(\theta_i,\pi_g) - \sum_{j\in\alpha,j\neq i} v_j(\pi_g^{-i},\theta_j) + \sum_{j\in\alpha,j\neq i} v_j(\pi_g,\theta_j). \quad (3)$$

Agent i wants to maximize its utility, and thus, wants to maximize the third term (eq.3) that represents the payment portion it will be receiving from the mechanism. Obviously - and as a VCG mechanism, the mechanism prevents traditional lying, as it may decrease the third term. For over-reporting - where agent i will claim actions that it doesn't possess, this may increase the agent's received payments if these over-reported actions were included in the global plan. However, the agent knows that its received payments will be mapped to (i.e. distributed over) its local state space, which implies that some portion of its payment will be linked to some imaginary states that it will never reach, simply as it doesn't have the actions to reach there. Given that agent i doesn't know how its reward function will be designed (i.e. what amount will be linked to each state), then the Coordination-VCG prevents over-reporting and any combination of both types of lying. □

We found it convenient - for budget balance issues - to define the $h_i(\theta_{-i})$ following Clarke pivot rule [5] for the following reasons.

[2]From the computational view point.

[3]Without loss of generality that our mechanism can be applied in the second case as well.

First, it reduces the mechanism's required budget for operating the system, as the mechanism will be receiving the value of $h_i(\theta_{-i})$ from each participating agent i. *Second*, the payment term $p_i(\theta)$ is more expressive, as it corresponds to the difference between other agents' valuation without and with agent i, and thus, agent i is paying an amount equivalent to its effect on the system. An agent may end up by paying the mechanism, which will be the case if the agent existence harms the system more than benefiting it, for instance, by introducing conflicts that require more costly actions from other agents which in turn will decrease their values. Or on the other hand, an agent may end up by receiving payments from the mechanism, if its benefits to system are more than its harm, for instance, by helping other agents to achieve their goals and/or resolve conflicts on its expense. *Finally*, it allows us shedding the light on the budget balance property of the proposed Coordination-VCG mechanism. Budget balance is a desirable property that indicates that there is no net transfer of payments out of the system or into the system. Whether the Coordination-VCG mechanism is budget balanced or not, this will depend mainly on the characteristics possessed by the multiagent planning setting in hand. Considering a multiagent planning setting with three agents (A, B and C), their payments can be expressed as follows:

$$p_A = v_B(\pi_{B,C}) + v_C(\pi_{B,C}) - v_B(\pi_{A,B,C}) - v_C(\pi_{A,B,C}) \quad (4)$$

$$p_B = v_A(\pi_{A,C}) + v_C(\pi_{A,C}) - v_A(\pi_{A,B,C}) - v_C(\pi_{A,B,C}) \quad (5)$$

$$p_C = v_A(\pi_{A,B}) + v_B(\pi_{A,B}) - v_A(\pi_{A,B,C}) - v_B(\pi_{A,B,C}) \quad (6)$$

For the first payment equation (eq.4), agent A will pay an amount equivalent to the values of agents B and C from the global plan that only considers both of them, and will receive an amount equivalent to the values of agents B and C from the global plan that include the three agents. The first and third terms capture the effect of agent A on agent B due to its existence in the system (i.e. $\mathit{Eff}_{A\to B} = v_B(\pi_{B,C}) - v_B(\pi_{A,B,C})$), while the second and fourth terms capture its effect on agent C (i.e. $\mathit{Eff}_{A\to C} = v_C(\pi_{B,C}) - v_C(\pi_{A,B,C})$). Note that the effect of agent A on agent B (i.e. $\mathit{Eff}_{A\to B}$) is considered while agent C dually influences and is affected by this relation. Similar interpretation holds for agent B and agent C equations. The system's payments then can be expressed by the agents' effects on one another as follows $\mathit{Eff}_{A\to B} + \mathit{Eff}_{A\to C} + \mathit{Eff}_{B\to A} + \mathit{Eff}_{B\to C} + \mathit{Eff}_{C\to A} + \mathit{Eff}_{C\to B}$. By assuming that the effect of one agent on another has nothing to do with other agents in the system, the system will be budget balanced as the effect of any two agents on one another (e.g. $\mathit{Eff}_{A\to B}$ and $\mathit{Eff}_{B\to A}$) will offset each other. Thus, if the multiagent planning setting in hand possesses the characteristic that the effect of one agent on another has nothing to do with other agents in the system, meaning, the interactions within the system can be described by independent pairwise interactions between the agents, then the budget balance property will be attained under the Coordination-VCG mechanism. Achieving budget balance along with the allocative efficiency property (which is already attained in the Coordination-VCG) implies pareto-optimality. Such settings can describe, for instance, a service provider that has sufficient resources to serve all his potential clients, his relation with one client is not affected by his relation with others.

Proposition 2: *The Coordination-VCG mechanism is budget balanced and pareto-optimal for multiagent planning settings where pairwise interactions between agents are independent.*

In the following section, we will illustrate an implementation of the proposed Coordination-VCG, assuming that both the agents and mechanism are using Markov decision process as their planning model. We want to emphasize that this is *not* the only possible implementation for the proposed mechanism, and that this is not a general purpose implementation which may not be suitable for some planning problems because of the assumed planning scenario. However, this detailed implementation narrows down the gap between the theoretical contribution of this study and its practical purposes.

6. MARKOV DECISION PROCESS FOR PLANNING UNDER A COORDINATION-VCG

Decision-theoretic planning (DTP) is an extension of the classical AI planning paradigm that allows modeling of problems with incomplete information, in which uncertainty is associated with the actions' effects. The main aim in this context is to determine a plan/policy that has the highest expected utility. Most of the sequential decision problems can be viewed as a Markov decision processes (MDPs) [3], and thus, many studies are carried out on DTP while adopting MDPs as their underlying model.

MDPs are normally used for modeling stochastic dynamical systems, where at any given point of time, the system will be in one of its distinct states. A state provides a description of the system at some point of time, and it is commonly assumed (Markov assumption) that it captures all the agent's required information to carry out its decision-making process (i.e. $Pr(s^{t+1}|s^t, s^{t-1}, \ldots, s^0) = Pr(s^{t+1}|s^t)$). MDPs assume extensional representation of the system, where the whole system's state space is provided with each state explicitly named. However, in AI research, intensional representation is often used, where the system's states are described using sets of multi-valued features.

The system state evolves over time in response to the occurring events, these events can be viewed as the agent's own actions - totally controlled by the agent (what actions are taken and when), or/and some partially predictable exogenous events that may occur beyond the agent's control (e.g. evolution of a natural process or other agents' actions). Both kinds of events may - but not necessarily - stochastically cause state transitions. Putting exogenous events aside for now, given the current state of the system and the agent's action, the influence is determined according to a probability distribution over the system's possible next states. Assuming a stationary system, this probability distribution is independent from the MDP stage - concerning time steps.

Both the state space and the probability distribution governing possible state transitions describe the discrete-time stochastic process, which determines how the system will evolve in response to the events which may not be perfectly predictable. Starting from some initial state, the agent's goal(s) can be expressed by some system state(s) which the agent wants to find its way to by deciding on its policy, while avoiding undesirable states along the way. The agent can assess its plan according to whether the goal state(s) can be reached with sufficient probability, while optimizing some objective function associated by the plan execution (e.g. maximizing value). The above description can capture classical planning problems (goal-oriented, deterministic and complete knowledge), and extensions such as conditional and probabilistic planning problems.

6.1 Single Agent MDP

The fully-observable MDP planning problem of single agent i can be described as follows: *i)* A finite set $S^i = \{s_1, \ldots, s_{n_i}\}$ describing the possible n_i states of the planning problem of agent i; *ii)* A finite set $A^i = \{a_1, \ldots, a_{k_i}\}$ describing the k_i possible actions of agent i; *iii)* A probability distribution Pr_a^i associated with each of the agent's actions, describing the transition probabilities

upon taking action a in state s; *iv)* A discount factor $\gamma \in [0,1]$ used for discounting the value of future transitions; *v)* The agent's initial state denoted by $I^i \in S^i$; *vi)* The agent's goal state $G^i \in S^i$; *vii)* A cost function $c^i : A^i \rightarrow \mathbb{R}$ that assigns a cost for each action; *viii)* A goal reward function $rg^i : G^i \rightarrow \mathbb{R}$ that assigns a reward for achieving the goal, the reward function can be defined over the whole state space while assigning the reward 0 for other states than the goal state. Thus, the planning MDP of agent i can be described by the tuple $\theta_i = \langle S^i, A^i, Pr^i_a, \gamma, I^i, G^i, c^i, rg^i \rangle$, where θ_i determines the type of agent i.

The plan/policy of agent i is denoted by $\pi_i : S \rightarrow A$, that defines which action to be taken in each state. Starting from the initial state I^i, the agent simply wants to reach its goal G^i with the minimum cost for its undertaken actions. In other words, the agent wants to maximize its value function, expressed by $v_i(\pi_i) = rg^i(\pi_i) - c^i(\pi_i)$, where $v_i(\pi_i)$ denotes its value given that the agent adopts the plan π_i, while $rg^i(\pi_i)$ and $c^i(\pi_i)$ denote its reward and cost according to policy π_i, respectively.

Normally for MDPs, we can define the value [4] of agent i in a state s with an action a as $V_i(s,a) = RG^i(s) - C^i(s,a)$. However, under a Coordination-VCG mechanism, agent i will receive a payment from the mechanism (the third term in eq.3) expressed by a reward function r^i that maps the payment amount to the agent's state-space. The reward function r^i is denoted at the state level by R^i, and this changes the value of agent i in state s with an action a to be

$$V_i(s,a) = RG^i(s) + R^i(s) - C^i(s,a) \quad (7)$$

The above equation captures the utility of agent i under the Coordination-VCG mechanism at the state level, while ignoring the payment that the agent will pay to the mechanism (i.e. the second term in eq.3 which the agent can't influence). For simplicity, we will denote this utility as the agent's valuation that considers the received payment from the mechanism. This equation can easily be generalized to histories, let $h = [s_0, s_1, \ldots]$ be one arbitrary history sequence, then the agent's valuation for h induced by policy π_i is defined as: $v_i(h|\pi_i) = \sum_{t \geq 0} \gamma_t (RG^i(s_t) + R^i(s_t) - C^i(s_t, \pi_i(s_t)))$, where t indicates the time step. The expected value of policy π_i is computed over the space of all possible histories H, according to the probability of their possible occurrence, defined as: $E(\pi_i) = \sum_{h \in H} Pr(h|\pi_i) v_i(h|\pi_i)$. The agent's optimal policy/plan π^*_i is the one that has the maximal expected value, i.e. $E(\pi^*_i) \geq E(\pi_i)$ for any other policy π_i. Let $E(s)$ be the expected value at state s, and thus, the expected value at state s when taking action a is defined by: $Q_i(s,a) = V_i(s,a) + \gamma \sum_{s' \in S^i} Pr^i_a(s'|s) E(s')$. The optimal (maximum) value $E(\pi^*_i)$ must satisfy Bellman's fixed point equation: $E(s) = argmax_{a \in A^i} Q_i(s,a), \forall s \in S^i$ where a is the action to be taken in state s according to policy π^*_i (i.e. $a \equiv \pi^*_i(s)$).

6.2 Multiagent MDP and Local Rewards Implementing Global Plan

In the previous section, we described an agent's private planning problem. The mechanism will ask the m agents - in the α set of agents - to report their types, where an agent i will report its type $\theta_i = \langle S^i, A^i, Pr^i_a, \gamma, I^i, G^i, C^i, RG^i \rangle$. Using the agents' reported information, the mechanism can start formulating the multiagent planning problem as a multiagent MDP (MMDP) [2] described by the tuple $\langle \alpha, S, \{A^i\}_{i \in \alpha}, Pr, \gamma, \{I^i\}_{i \in \alpha}, \{G^i\}_{i \in \alpha}, \{c^i\}_{i \in \alpha}, \{rg^i\}_{i \in \alpha} \rangle$, where $\{A^i\}_{i \in \alpha}, \{I^i\}_{i \in \alpha}, \{G^i\}_{i \in \alpha}, \{c^i\}_{i \in \alpha}$ and $\{rg^i\}_{i \in \alpha}$ are the sets of the agents' reported actions, initial states, goal states, cost functions and goal reward functions, respectively. The mechanism needs to create a global state space S, and a probability distribution Pr over the joint action spaces of all the agents which will describe the transition probabilities upon taking a joint action $\langle a_1, \ldots, a_m \rangle$ in state $s \in S$. We will not discuss here the relation between the global state space S and the agents' state space S^i, as this relation is problem dependent and will vary from one problem to another.

After constructing the MMDP, the global optimal policy/plan π_g can be determined in a similar way as discussed in the previous subsection for a MDP. This global plan will maximize the summation of all the agents' valuations using their reported cost and goal reward functions. Till here, we have shown the implementation of the first two steps in the Coordination-VCG mechanism. For calculating the agents' payments (step 3), the previously constructed MMDP can be used in computing the third term in the payment function (eq.3) for each agent. However, for computing the second term (eq.3) for each agent i, the mechanism needs to remove agent i from the system and construct a new MMDP - in a similar way - using the information reported by all the other agents.

The global optimal plan π_g is actually m local optimal plans, π^i_g for each agent i, those that will yield to the best (optimal) global performance when executed. In some cases, there may be several optimal plans π^i_g for each agent i from the system's view point, meaning, there are several optimal global plans that correspond to different local optimal plans for agent i. In such cases, the mechanism needs to decide exactly on which optimal global plan to be considered, and thus, an exact plan π^i_g to be followed by agent i. The mechanism wants the optimal plan π^*_i decided by agent i to be equivalent to its optimal plan π^i_g from the system's perspective. To do this, the mechanism will design the reward function r^i in a way that the local plan π^i_g - which is optimal from the global view - will become the unique optimal plan (i.e. π^*_i) that agent i will determine while maximizing its expected value, and undertake it in execution. Ng and Russell (theorem 3 in [12]) stated that for π^i_g to be the unique optimal policy for a finite state space MDP, the value vector $\mathbf{V_i}$ of agent i over its state space must satisfy the following constraint (without much details)

$$(Pr^i_{a_1} - Pr^i_a)(I - \gamma Pr^i_{a_1})^{-1} \mathbf{V_i} > 0 \quad (8)$$

where $a_1 \equiv \pi^i_g$ (the action corresponding to π^i_g at some state), I is an identity matrix, and a represents any possible action other than a_1. Recalling that $\mathbf{V_i} = \mathbf{RG^i} + \mathbf{R^i} - \mathbf{C^i}$ (which is eq.7 expressed by vectors over the state space), and that the cumulative rewards mapped to the state space $\|R^i\|$ must be equivalent to the payment amount received by agent i from the mechanism, then this second constraint must hold:

$$\|\mathbf{V_i} + \mathbf{C^i} - \mathbf{RG^i}\| = \sum_{j \in \alpha, j \neq i} v_j(\pi_g, \theta_j) \quad (9)$$

where $\mathbf{C^i}$ and $\mathbf{RG^i}$ are considered constants known by the mechanism from the declared information by agent i, and $\sum_{j \in \alpha, j \neq i} v_j(\pi_g, \theta_j)$ is the payment amount received by agent i. The mechanism simply wants to find the $\mathbf{V_i}$ vector that satisfies both constraints simultaneously, which guarantees that π^i_g will be the unique optimal policy,

[4] We are using capital variables (i.e. V_i, C^i, RG^i) to express the value, cost and reward of agent i at the state level, while bold capital variables (i.e. $\mathbf{V_i}, \mathbf{C^i}, \mathbf{RG^i}$) to express vectors over the whole state space (i.e. each value in the vector is mapped to some state). Small variables $i.e. v_i, c^i, rg^i$ are used to express the value, cost and reward of agent i for trajectories (i.e. sequence of states).

and that the mapped rewards are equivalent to the agent's received payment. After determining $\mathbf{V_i}$, the vector $\mathbf{R^i} = \mathbf{V_i} + \mathbf{C^i} - \mathbf{RG^i}$ shows the exact map of the agent's received payment to its state space, which express the reward function r^i that the mechanism will report back to agent i. When agent i starts determining its own plan in order to maximize its valuation (using eq.7), taking into consideration the reward function r^i reported by the mechanism, the determined optimal plan will also be optimal from the system's perspective. And thus, the mechanism succeeded in controlling agent i during the execution phase using its received payment.

7. CONCLUDING SUMMARY AND FUTURE WORK

In informationally decentralized multiagent planning, rational agents usually have conflicting interests, and thus, they may strategically misrepresent their private information in order to maximize their own utility. Mechanism design can aid in extracting the agents' private information truthfully. However, multiagent planning differs from the classical applications of mechanism design by having an additional execution phase. In this study, we discussed previous efforts for designing mechanisms for controlling multiagent planning, and provided a modified Deposit-VCG. We contributed a novel Coordination-VCG mechanism which is truthful in both phases for all types of lies, and proved its correctness. We showed that the proposed mechanism is budget balanced and pareto-optimal for planning settings where the agents' pairwise interactions are independent. We provided a possible implementation for the proposed Coordination-VCG, where we assumed MDPs to be the planning model used by the agents and the mechanism. And we showed how the agents' payments can be projected back to their local planning problems. More efficient ways of handling the global planning process, and investigating the computation and communication aspects appear fruitful avenue of pursuit. As well as investigating how individual rationality can be achieved under the suitable MAP models.

8. ACKNOWLEDGMENT

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. The author is indebted to Jussi Rintanen, Patrik Haslum and Scott Sanner for helpful discussions and suggestions for improving the exposition of the material in this paper.

9. REFERENCES

[1] S. Botelho and R. Alami. Cooperative plan enhancement in multi-robot context. *Intelligent Autonomous Systems*, 6:131–138, 2002.

[2] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 478–485, Stockholm, 1999.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, July 1999.

[4] R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS 2008*, pages 28–35, September 2008.

[5] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17Ű33, 1971.

[6] J. Dix, H. Muñoz-Avila, D. S. Nau, and L. Zhang. IMPACTing SHOP: Putting an AI planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence*, 37(4):381–407, April 2003.

[7] E. H. Durfee. Scaling up agent coordination strategies. *IEEE Computer*, 34(7):39–46, July 2001.

[8] T. Groves. Incentives in teams. *Econometrica*, 41:617Ű631, 1973.

[9] L. Hurwicz. Optimality and informational efficiency in resource allocation processes. In K. J. Arrow, S. Karlin, and P. Suppes, editors, *Proceedings of the First Stanford Symosium on Mathematical Methods in the Social Sciences, 1959*, volume IV of *Stanford Mathematical Studies in the Social Sciences*, pages 27–46. Stanford University Press, 1960.

[10] L. Hurwicz. On informationally decentralized systems. In C. B. McGuire and R. Randner, editors, *Decision and Organization, A Volume in Honor of Jacob Marschak*, volume 12 of *Studies in Mathematical and Managerial Economics*, chapter 14, pages 297–336. North-Holland Publishing Company, Amsterdam, 1972.

[11] R. B. Myerson. Incentive compatibility and the bargaining problem. *Econometrica*, 47(1):61–73, January 1979.

[12] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663 – 670. Morgan Kaufmann Publishers Inc., 2000.

[13] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

[14] F. Pecora and A. Cesta. Planning and scheduling ingredients for a multi-agent system. In *Proceedings of UK PLANSIG*, pages 135–148, Delft (The Netherlands), November 2002.

[15] R. P. van der Krogt, M. M. de Weerdt, and Y. Zhang. Of mechanism design and multiagent planning. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*, pages 423–427. IOS Press, 2008.

[16] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16:8Ű37, 1961.

[17] Y. Zhang and M. M. de Weerdt. Creating incentives to prevent execution failures: an extension of VCG mechanism. In J. Dix, E. H. Durfee, and C. Witteveen, editors, *Planning in Multiagent Systems*, number 08461 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Point-Based Policy Generation for Decentralized POMDPs

Feng Wu
School of Computer Science
Univ. of Sci. & Tech. of China
Hefei, Anhui 230027 China
wufeng@mail.ustc.edu.cn

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

Xiaoping Chen
School of Computer Science
Univ. of Sci. & Tech. of China
Hefei, Anhui 230027 China
xpchen@ustc.edu.cn

ABSTRACT

Memory-bounded techniques have shown great promise in solving complex multi-agent planning problems modeled as DEC-POMDPs. Much of the performance gains can be attributed to pruning techniques that alleviate the complexity of the exhaustive backup step of the original MBDP algorithm. Despite these improvements, state-of-the-art algorithms can still handle a relative small pool of candidate policies, which limits the quality of the solution in some benchmark problems. We present a new algorithm, Point-Based Policy Generation, which avoids altogether searching the entire joint policy space. The key observation is that the best joint policy for each reachable belief state can be constructed directly, instead of producing first a large set of candidates. We also provide an efficient approximate implementation of this operation. The experimental results show that our solution technique improves the performance significantly in terms of both runtime and solution quality.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coherence and coordination, Multi-agent systems*

General Terms

Algorithms, Experimentation, Performance

Keywords

Teamwork, Coordination, Multi-Agent Planning, Decision-Theoretic Planning, Decentralized POMDPs

1. INTRODUCTION

Cooperative multi-agent decision making arises naturally in many real-world applications such as cooperative robots, planetary exploration, distributed sensor networks, and disaster response. These problems are difficult or impossible to solve using centralized decision making frameworks. In the RoboCup domain, for example, a group of robots with noisy sensors and inaccurate actuators must cooperate with each other to play soccer and win the game. With only partial view of the environment, each robot must reason about the choices of the others and how they may affect the environment. There are many sources of uncertainty in this problem and the state space is very large. Developing decision-theoretic techniques that can cope with this complexity is thus an important challenge.

The Markov decision process (MDP) and its partially observable counterpart (POMDP) have proved useful in planning and learning under uncertainty. A natural extension of these models to cooperative multi-agent settings is provided by the decentralized POMDP (DEC-POMDP) framework, in which agents have different partial knowledge of the environment and other agents. The DEC-POMDP framework is very expressive and can model many practical problems including the ones mentioned above. Unfortunately, solving it optimally has been shown to be NEXP-complete [9]. Thus, optimal algorithms [5, 14, 18, 19, 26, 27] can only solve very small problems. In recent years, researches proposed several approaches to improve the ability to solve larger problems. Examples include algorithms that exploit the structure of interaction in subclasses of DEC-POMDPs such as Transition Independent DEC-MDPs (TI-DEC-MDPs) [6] and Network Distributed POMDPs (ND-POMDPs) [17]. In other efforts researchers managed to address the complexity of the general model by considering communication explicitly [13, 15, 21]. However, not all real-world problems exhibit the necessary independence conditions, and communication is often costly and sometimes unavailable in the case of robots that operate underground or on other planets.

More general algorithms that compute approximate solutions have shown great promise using either offline methods [2, 3, 10, 11, 16, 23, 24] or online techniques [12, 22, 29]. Online algorithms must often meet tight time-constraints and the solution quality highly depends on the heuristics they use. There has also been substantial work on solving approximately DEC-POMDPs with infinite horizons [1,7,8]. In this paper, we focus on approximate offline algorithms for finite-horizon DEC-POMDPs. Currently, the state-of-the-art solution techniques still suffer from limited scalability.

The approach that is closest to our work is called Memory-Bounded Dynamic Programming (MBDP) [24]. It combines top-down and bottom-up components to build and optimize policies. The top-down component is used to generate a set of reachable belief states, usually guided by some heuristics. The bottom-up dynamic programming component is then used to build a set of possible policies based on the policies of the previous step. At the end of each step, only the best policies for the reachable belief states are kept as the building blocks for the next step. Since the number of policies kept at each step is bounded by a parameter called $maxTrees$, MBDP has a linear time and space complexity with respect

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

to the horizon. However, the exhaustive backup that MBDP uses to build a set of possible policies is very inefficient. Several successor algorithms have been developed to alleviate this problem by performing a partial backup with respect to the observations, by focusing on the most likely observations in IMBDP [23] or compressing the set of observations in MBDP-OC [10]. More recently, there have been several moderately successful attempts to further improve the exhaustive backup. One example is PBIP [11] that replaces the backup step with a branch-and-bound search in the space of joint policies. Another example is IPG [3] that can prune useless policies before actually generating them. While these ideas have produced very significant computational savings, the resulting algorithms can still handle a relative small pool of candidate policies (measured by $maxTrees$). A small pool of policies is sometimes sufficient to obtain near optimal results, but in other cases it leads to a significant loss of value. Our goal in this paper is to introduce an algorithm that can operate more efficiently with a significantly larger pool of candidate policies. The objective is to produce better quality solutions much faster and to improve the overall scalability of approximate DEC-POMDP algorithms in order to solve large problems.

We present a new algorithm called Point-Based Policy Generation for DEC-POMDPs, which combines more efficiently the top-down and bottom-up components of the MBDP family of algorithms. MBDP produces a bounded pool of policies that are optimized with respect to a set of reachable belief states. The key observation behind the development of the new algorithm is that *the best policy for each reachable belief state can be constructed directly*, instead of producing first a large set of candidates. Thus, we first construct the joint policies based on a given belief state and each joint action, then select the best one for the given belief point. Consequently, we avoid altogether performing backup for all possible policies or searching over the entire joint policy space. We prove that when performed optimally, our procedure is equivalent to what MBDP does, resulting in the same policy value. We also provide an approximate implementation of this procedure that can solve the problem efficiently–much faster than the optimal version. The experimental results show that our solution technique works well. It significantly improves the performance of existing algorithms in terms of both runtime and solution quality over several DEC-POMDP benchmark problems.

The rest of the paper is organized as follows. We first introduce the DEC-POMDP model and the MBDP family of algorithms. Then we describe the main algorithm and analyze its properties. We then present and discuss the approximation technique. Finally, we examine the performance of the algorithm on several benchmark problems and demonstrate its efficiency. We conclude with a summary of the contributions and future work.

2. DECENTRALIZED POMDPS

We adopt here the DEC-POMDP framework and notation [9], however our approach and results apply to equivalent models such as MTDP [21] and POIPSG [20].

Definition 1. A finite-horizon Decentralized Partially Observable Markov Decision Process (DEC-POMDP) is defined as a tuple $\langle I, S, \{A_i\}, \{\Omega_i\}, P, O, R, b^0\rangle$ where

- I is a finite set of agents indexed $1, \cdots, n$.

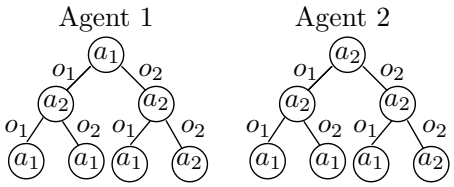


Figure 1: Example of a joint policy for 2 agents.

- S is a finite set of system states.
- A_i is a finite set of actions available to agent i and $\vec{A} = \times_{i\in I} A_i$ is the set of joint actions, where $\vec{a} = \langle a_1, \cdots, a_n\rangle$ denotes a joint action.
- Ω_i is a finite set of observations available to agent i and $\vec{\Omega} = \times_{i\in I}\Omega_i$ is the set of joint observations, where $\vec{o} = \langle o_1, \cdots, o_n\rangle$ denotes a joint observation.
- P is a Markovian state transition table. $P(s'|s, \vec{a})$ denotes the probability that taking joint action $\vec{a}$ in state s results in a transition to state s'.
- O is a table of observation probabilities. $O(\vec{o}|s', \vec{a})$ denotes the probability of observing joint observation $\vec{o}$ after taking joint action $\vec{a}$ and reaching state s'.
- $R : S \times \vec{A} \to \Re$ is a reward function. $R(s, \vec{a})$ denotes the reward value obtained from taking a joint action $\vec{a}$ in state s.
- $b^0 \in \Delta(S)$ is the initial belief state distribution.

In this paper we focus on the general DEC-POMDP problem with a finite horizon T and start state distribution b^0. Solving this problem can be seen as finding policies that maximize the expected joint reward for b^0 over T. While execution is inherently distributed, planning is performed offline and can be centralized.

In DEC-POMDPs, the policy of an agent is represented as a tree and a joint policy as a vector of trees, one for each agent. As shown in Figure 1, each node of the policy tree is labeled by an action to take and each edge is labeled by an observation that may occur. This continues until the horizon T is reached at the leaf nodes. When executing a policy tree at runtime, the agent follows a path from the root to a leaf depending on the observations it receives as it performs the actions at the nodes. The value function of a joint policy $\vec{q}^{\,t+1}$ is defined recursively as follows:

$$V^{t+1}(\vec{q}^{\,t+1}, s) = R(s, \vec{a}) + \sum_{s', \vec{o}} P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a}) V^t(\vec{q}^{\,t}_{\vec{o}}, s') \quad (1)$$

where $\vec{a}$ is the joint action at the root nodes of $\vec{q}^{\,t+1}$ and $\vec{q}^{\,t}_{\vec{o}}$ is the joint subtree of $\vec{q}^{\,t+1}$ after $\vec{o}$ is observed. The belief state used in this paper is a probability distribution over states $b \in \Delta(S)$. The value of a joint policy $\vec{q}$ for a belief state b is defined as $V(\vec{q}, b) = \sum_{s\in S} b(s) V(\vec{q}, s)$. A survey of the DEC-POMDP model and algorithms is available in [25].

3. MBDP AND ITS SUCCESSORS

Memory-Bounded Dynamic Programming (MBDP) [24] was the first algorithm to combine top-down and bottom-up solution techniques for DEC-POMDPs. In this section we describe previous work on MBDP and its successors, which are the best existing solution techniques for finite-horizon DEC-POMDPs. In MBDP, policies are constructed incrementally using bottom-up dynamic programming. A parameter, $maxTrees$, is chosen to ensure that a full backup with

Algorithm 1: The MBDP Algorithm

$\vec{Q}^1 \leftarrow$ initialize all 1-step policy trees
for $t = 1$ **to** $T - 1$ **do**
  $\vec{Q'}^{t+1} \leftarrow$ perform full backup on $\vec{Q}^t$
  $\vec{Q'}^{t+1} \leftarrow$ prune dominated policies in $\vec{Q'}^{t+1}$
  $\vec{Q}^{t+1} \leftarrow \{\}$
  for $k = 1$ **to** $maxTrees$ **do**
    $b \leftarrow$ generate a belief using a heuristic portfolio
    $\vec{q} \leftarrow$ select the best joint policy in $\vec{Q'}^{t+1}$ for b
    $\vec{Q}^{t+1} \leftarrow \vec{Q}^{t+1} \cup \{\vec{q}\}$
return the best joint policy in $\vec{Q}^T$ for b^0

this number of policies for the current step does not exceed the available memory. At each iteration, a full backup of the policies from the last iteration is performed. Then, top-down heuristics are selected from the portfolio to compute a set of reachable belief states. Finally, the best joint policies for these belief states are added to the new sets of policies. After the Tth backup, the best joint policy for the initial belief state is returned. The *best* joint policy is a joint policy with the highest value for a certain belief point. The main procedure is shown in Algorithm 1.

3.1 Backup Operations

The original MBDP uses *exhaustive backups* (or full backups) to construct policy trees. This operation generates every possible depth-t+1 policy tree for each action and each possible observation to the root node of some depth-t policy tree. If an agent has $|Q_i|$ depth-t policy trees, $|A_i|$ actions, and $|\Omega_i|$ observations, there will be $|A_i||Q_i|^{|\Omega_i|}$ depth-t+1 policy trees. This is inefficient because the number of possible depth-t+1 policy trees is still exponential in the size of the observation space. Thus, an improved version of MBDP (IMBDP) [23] was introduced to use *partial backups*. It first identifies the set of most likely observations for every agent bounded by a predefined number of observations *maxObs*, and then performs a backup with only these observations for each agent. The missing observation branches are filled up by using local search. Another approach to this problem is implemented in MBDP with *observation compression* (MBDP-OC) [10]. Instead of ignoring observations that are less probable, MBDP-OC merges certain sets of observations guided by the value lost. It seeks to reduce the exponential generation of new policies by forcing different branches of the new root policies to contain the same subtrees. The observation branches are merged so as to minimize the loss of value. Although the methods above can alleviate the complexity of the one-step backup operation, they are still time-consuming. We show in this paper that there is much to be gained if the bottom-up policy construction considers the reachable belief states generated by the top-down heuristics *from the very beginning*. The reason is that only the best policy trees for the reachable belief states are kept at the end of each iteration and most of them are useless.

3.2 Pruning Techniques

In order to reduce the the number of policy trees, the MBDP algorithm does pruning by using *iterated elimination of dominated policies* after each backup. A policy tree is dominated if for every possible belief state there is at least one other policy tree which is as good as or better than it. This test for dominance is performed using a linear program. Removing a dominated policy tree does not reduce the value of the optimal joint policy. Unfortunately, even with this pruning technique, the number of policy trees still grows quickly. To alleviate this problem, a new approach called *point-based incremental pruning* (PBIP) [11] was proposed, which uses branch-and-bound search in the space of joint policy trees instead. PBIP computes upper and lower bounds on the partial depth-t+1 joint policy trees using heuristics, and prunes dominated trees at earlier construction stages. The bounds are calculated by considering the belief states and the depth-t policy trees. PBIP prunes depth-t+1 policy trees that are outside the upper and lower bounds, but it does not exploit the reachability of policies.

3.3 Reachability Analysis

Recently, a new method called *incremental policy generation* (IPG) [3] was proposed to generate policies based on a state space reachability analysis. Intuitively, the action taken and observation seen may limit the possible next states no matter what actions the other agents perform. This allows only policies that are useful for some possible states to be retained. This approach may generate a smaller number of policies without losing value. It first generates all possible sets of depth-t trees for each observation with a fixed action, one for each agent. Then, it creates all possible depth-t+1 trees that begin with the fixed action followed by choosing any trees from the depth-t set after an observation is obtained. Once all depth-$t+1$ trees for each action are generated, it takes the union of the sets and produces the set of depth-t+1 trees. This approach can be incorporated with any DEC-POMDP algorithm that performs dynamic programming backups. While it exploits the state space reachability, this approach does not consider the reachable belief states generated by the top-down heuristics.

4. POINT-BASED POLICY GENERATION

As mentioned above, a better way to perform the bottom-up dynamic programming step is to construct the best joint policy for each belief state only–avoiding either full or partial backups. In this section, we propose a new method which generates directly the best joint policy for each belief state, namely *Point-Based Policy Generation* (PBPG).

4.1 Problem Formulation

Given a belief state b, the basic idea is as follow: for every joint action $\vec{a}$, we first find the best sub-policy trees for every possible observation branch after taking action $\vec{a}$; then, we choose the best joint action for b and build the best joint policy. Formally, this problem can be defined as follow:

Definition 2. Given a belief state b, a joint action $\vec{a}$, depth-t policy tree sets $\vec{Q}^t = \langle Q_1^t, Q_2^t, \cdots, Q_n^t \rangle$ and a value function $V^t : \vec{Q}^t \times S \rightarrow \Re$, find mappings $\delta_i : \Omega_i \rightarrow Q_i^t, \forall i \in I$ which maximize the depth-t+1 value function

$$V^{t+1}(\vec{a}, b) = R(\vec{a}, b) + \sum_{s', \vec{o}} Pr(\vec{o}, s' | \vec{a}, b) V^t(\vec{\delta}(\vec{o}), s') \quad (2)$$

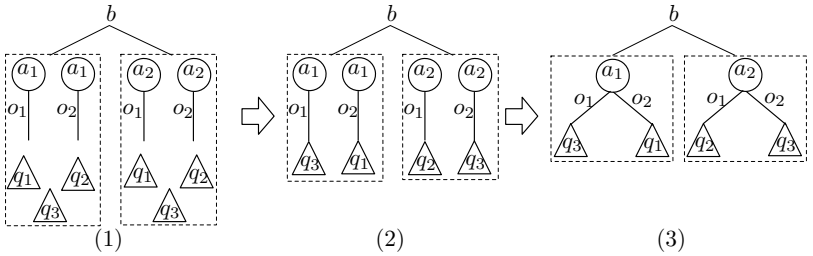


Figure 2: Example of the policy-tree construction process for two agents with two observations: (1) shows the input, which includes a belief state b, a joint action $\langle a_1, a_2 \rangle$, and sets of depth-t policy trees represented by the triangles; (2) shows the best mappings for the given belief state and joint action from each observation to a depth-t policy tree; (3) shows a joint policy tree built using the mappings.

where $\vec{\delta}(\vec{o}) = \langle \delta_1(o_1), \delta_2(o_2), \cdots, \delta_n(o_n) \rangle = \langle q_1^t, q_2^t, \cdots, q_n^t \rangle$, the probability $Pr(\vec{o}, s'|\vec{a}, b) = O(\vec{o}|s', \vec{a}) \sum_s P(s'|s, \vec{a}) b(s)$, and the reward function $R(\vec{a}, b) = \sum_s b(s) R(s, \vec{a})$.

This problem is essentially a subtree selection problem, where the goal is to choose depth-t subtrees in response to observations to maximize the depth-$t+1$ value function (Equation 2). However, the subtrees must be chosen in a decentralized manner. That is, subtree q_i^t is chosen based only on an observation of o_i for agent i. Our goal is to choose mappings or selection rules $\delta_i : \Omega_i \rightarrow Q_i^t, \forall i \in I$ to maximize the depth-$t+1$ value function and build the best depth-$t+1$ policy trees *given* the belief b and the joint action $\vec{a}$.

With the joint action $\vec{a}$ and the best mappings $\delta_i, \forall i \in I$, it is quite straightforward to construct the depth-$t+1$ policy trees. For agent i, the depth-$t+1$ policy tree can be built by using a_i as the root node and assigning the sub-policies for each observation branch based on δ_i. Figure 2 shows an example of the construction process with the resulting mappings $\delta_1 : o_1 \rightarrow q_3, o_2 \rightarrow q_1$ for agent 1 and $\delta_2 : o_1 \rightarrow q_2, o_2 \rightarrow q_3$ for agent 2. After the joint policy trees for b and every joint action are generated, we can find the best joint policy for the belief state b by choosing the one with the root nodes of $\vec{a}\,* = \langle a_1^*, \cdots, a_n^* \rangle$ computed as follows:

$$\vec{a}\,* = \arg\max_{\vec{a} \in \vec{A}} V^{t+1}(\vec{a}, b) \tag{3}$$

PROPOSITION 1. *For a given belief state b, the joint policy chosen by the method mentioned above yields the same value as the one selected by the MBDP algorithm.*

PROOF. Note that the depth-1 policy trees for both methods are the same with a single node of every possible action. Assume that the depth-t policy trees for both methods are also the same. For a belief state b, the MBDP algorithm first generates all possible depth-$t+1$ policy trees by exhaustive backup of the depth-t policies and then selects the joint policy $\vec{q}^{*t+1}$ which maximizes the value function

$$V^{t+1}(\vec{q}^{*t+1}, b) = \sum\nolimits_{s \in S} b(s) V^{t+1}(\vec{q}^{*t+1}, s) \tag{4}$$

where $V^{t+1}(\vec{q}^{*t+1}, s)$ is computed by Equation 1. The method described above first constructs a set of joint policy trees which maximize Equation 2 for every possible joint action and then chooses the joint policy with the joint action computed by Equation 3. Generally, we have

$$\begin{aligned} V^{t+1}(\vec{q}^{*t+1}, b) =& \sum_s b(s)[R(s, \vec{a}) + \sum_{s', \vec{o}} P(s'|s, \vec{a}) \cdot \\ & O(\vec{o}|s', \vec{a}) V^t(\vec{q}_{\vec{o}}^{\,t}, s')] \qquad Eq.4 \\ =& \sum_s b(s) R(s, \vec{a}) + \sum_{s', \vec{o}} [O(\vec{o}|s', \vec{a}) \cdot \\ & \sum_s P(s'|s, \vec{a}) b(s)] V^t(\vec{q}_{\vec{o}}^{\,t}, s') \\ =& R(\vec{a}, b) + \sum_{s', \vec{o}} Pr(\vec{o}, s'|\vec{a}, b) V^t(\vec{q}_{\vec{o}}^{\,t}, s') \quad Eq.2 \\ =& V^{t+1}(\vec{a}, b) \end{aligned}$$

where $\vec{a}$ is the root nodes of $\vec{q}^{*t+1}$ and $\vec{q}_{\vec{o}}^{\,t}$ is the sub-policy trees of $\vec{q}^{*t+1}$ for observation branches $\vec{o}$.

Thus, the joint policies selected by both methods yield the same value for belief state b at depth-$t+1$. Therefore, the proposition holds for every depth by induction. □

4.2 Approximate Solution

The key question is how to compute the best mappings $\delta_i, \forall i \in I$. Note that the number of possible mappings is $(|Q_i^t|^{|\Omega_i|})^{|I|}$ for a fixed joint action $\vec{a}$. Therefore, the straightforward way to enumerate all possible mappings is very inefficient. Actually, this problem is equivalent to the *decentralized decision making* problem studied by Tsitsiklis and Athans, which has been proved to be NP-hard even for two agents [28]. In this paper, we propose an efficient approximate method which solves the problem using a linear program and produces suboptimal solutions.

The approximation technique uses stochastic mappings instead of deterministic ones. They are defined as follows:

$$\pi_i : \Omega_i \times Q_i^t \rightarrow \Re, \forall i \in I.$$

That is, $\pi_i(q_i^t|o_i)$ is a probability distribution of subtrees q_i, given observation o_i for agent i. Similar to $\vec{\delta}$, we denote the joint stochastic mapping $\vec{\pi} = \langle \pi_1, \pi_2, \cdots, \pi_n \rangle$. Note that given b and $\vec{a}$, $R(\vec{a}, b)$ is a constant in Equation 2. Therefore, maximizing Equation 2 is equivalent to maximizing the following function:

$$V^{t+1}(\vec{\delta}, b) = \sum_{s', \vec{o}} Pr(\vec{o}, s'|\vec{a}, b) V^t(\vec{\delta}(\vec{o}), s') \tag{5}$$

With the stochastic mappings, Equation 5 can be rewritten as follows:

$$V^{t+1}(\vec{\pi}, b) = \sum_{s', \vec{o}} Pr(\vec{o}, s'|\vec{a}, b) \sum_{\vec{q}^t} \prod_i \pi_i(q_i^t|o_i) V^t(\vec{q}^{\,t}, s') \tag{6}$$

Table 1: The linear program for optimizing π_i

Variables: $\varepsilon, \pi_i'(q_i^t|o_i)$
Objective: maximize ε
Subject to:
Improvement constraint:
$V^{t+1}(\vec{\pi}, b) + \varepsilon \leq \sum_{s',\vec{o}} Pr(\vec{o}, s'|\vec{a}, b) \sum_{\vec{q}} \pi_i'(q_i^t|o_i) \cdot \pi_{-i}(q_{-i}^t|o_{-i}) V^t(\vec{q}^{\,t}, s')$
Probability constraints:
$\forall o_i \in \Omega_i, \sum_{q_i^t \in Q_i^t} \pi_i'(q_i^t|o_i) = 1;$
$\forall o_i \in \Omega_i, q_i^t \in Q_i^t, \pi_i'(q_i^t|o_i) \geq 0.$

There are multiple ways to solve Equation 6 and calculate the solution $\pi_i, \forall i \in I$. Our approximate method computes a suboptimal solution by choosing initial parameters for $\pi_i, \forall i \in I$ and iteratively optimizing the parameters of one agent while leaving the parameters of the other agents fixed, until no improvement is obtained. This process always terminates after a finite number of iterations if a threshold of minimum improvement is set. The suboptimal solution computed in this way can be considered analogous to a Nash equilibrium where no agent can benefit unilaterally.

To start, each local stochastic mapping $\pi_i, i \in I$ is initialized to be deterministic, by selecting a random $q_i^t \in Q_i^t$ with a uniform distribution. Then, each agent is selected in turn and its policy is improved while keeping the other agents' policies fixed. This is done for agent i by finding the best parameters $\pi_i'(q_i^t|o_i)$ satisfying the following inequality:

$$V^{t+1}(\vec{\pi}, b) \leq \sum_{s',\vec{o},\vec{q}^{\,t}} Pr(\vec{o}, s'|\vec{a}, b) \pi_i'(q_i^t|o_i) \pi_{-i}(q_{-i}^t|o_{-i}) V^t(\vec{q}^{\,t}, s')$$

where $\pi_{-i}(q_{-i}^t|o_{-i}) = \prod_{k \neq i} \pi_k(q_k^t|o_k)$.

The linear program shown in Table 1 is used to find the new parameters. The procedure terminates and returns $\vec{\pi}$ when ε becomes sufficiently small for all agents. Random restarts are used to avoid local maxima.

4.3 The PBPG Algorithm

Once the stochastic mapping π_i is computed for agent i, q_i^t is selected for observation branch o_i according to the distribution $\pi_i(q_i^t|o_i)$. The main steps of PBPG are shown in Algorithm 2. The number of joint policy trees generated at each iteration is bounded by $(|\vec{A}|maxTrees)$, much less than $(|A_i|maxTrees^{|\Omega_i|})^{|I|}$ produced by the full backup in the original MBDP. It is worth pointing out that the policy evaluation is very time-consuming and can easily run out of memory, especially for large problems. It computes the value for every *joint policy* at every state as shown in Equation 1. Note that the number of joint policies actually evaluated here is $(|\vec{A}|maxTrees + maxTrees^{|I|})$, not the total enumeration $(|A_i|maxTrees)^{|I|}$. We evaluate the joint policy for each joint action and prune the dominated ones at an early stage. This makes the algorithm more efficient. Efficiency can also be improved by exploiting the sparsity of the transition matrix, observation matrix and belief vector, which often have many zero elements. This property can be used to solve the necessary equations more quickly.

We also use a heuristic portfolio to generate the belief b at the beginning of each iteration, just as MBDP does. A heuristic portfolio is a set of heuristics which can be used to compute a set of belief states. Each heuristic is used to select a subset of the policy trees. In our implementation, we use two types of heuristics: the MDP heuristic and the random heuristic. The MDP heuristic is based on solving the underlying MDP in a centralized manner and executing the resulting policy to create sample belief states. The random heuristic produces random reachable belief points using random policies. Sometimes, several belief points select the same policy tree because the sampled beliefs are quite close to each other. When that happens, we re-sample to generate different belief points. Unlike MBDP, it is not necessary to run our algorithm *recursively* to obtain good results. The experimental results show significant improvement over all the existing algorithms simply by using a portfolio containing the above two simple heuristics in our algorithm.

Algorithm 2: Point-Based Policy Generation

```
T ← horizon of the DEC-POMDP model
maxTrees ← max number of trees at each step
Q⃗^1 ← initialize and evaluate all 1-step policy trees
for t = 1 to T − 1 do
    Q⃗^{t+1} ← {}
    for k = 1 to maxTrees do
        b ← generate a belief using a heuristic portfolio
        ν* ← −∞
        for a⃗ ∈ A⃗ do
            π⃗* ← compute the best mappings with b, a⃗
            q⃗ ← build a joint policy tree based on a⃗, π⃗*
            ν ← evaluate q⃗ by given the belief state b
            if ν > ν* then q⃗* ← q⃗, ν* ← ν
        Q⃗^{t+1} ← Q⃗^{t+1} ∪ {q⃗*}
    evaluate every joint policy in Q⃗^{t+1} with Equation 1
q⃗*^T ← select the best joint policy from Q⃗^T for b^0
return q⃗*^T
```

4.4 Summary and Discussion

To summarize, MBDP does an exhaustive backup for all depth-t policy trees before selecting a joint policy for each belief state. IMBDP ignores less-likely observation branches and MBDP-OC merges observations while minimizing the loss of value. PBIP prunes depth-$t+1$ policy trees using upper and lower bounds at the early stage, and IPG limits the number of depth-t policies using state reachability.

Unlike these existing algorithms, which try to improve the performance of the *backup* operation by limiting the number of observations or policies, we completely replace the backup step with an efficient policy generation method. Instead of generating a large set of policy trees, our approach constructs only a small set of possible candidates for each belief state using an efficient linear program. The number of policy trees generated for each belief state is bounded by the number of joint actions.

It is possible to incorporate previously developed methods with our algorithm, particularly observation compression and incremental policy generation, to further improve its performance. Merging equivalent observations and limiting the size of possible subtrees can immediately reduce the number of variables in the linear program and improve its scalability. Implementing these improvements, however, is beyond the scope of the paper and is left for future work.

5. EXPERIMENTAL EVALUATION

We tested our algorithm on the hardest existing benchmark problems. PBPG shares the same linear time and

space properties of MBDP with respect to the horizon. Therefore, our experiments focus mainly on scalability with respect to $maxTrees$, and the value gains that result from increasing $maxTrees$. We compared our results mainly to PBIP-IPG [3] – the latest algorithm, which has outperformed the other existing algorithms such as MBDP, IMBDP, MBDP-OC and PBIP. The purpose of these experiments is to show that our algorithm can solve problems with a much larger number of $maxTrees$ with good runtime and solution quality. Actually, the $maxTrees$ parameter presents a good way to tradeoff between runtime and solution quality, and it makes it possible to design a *contract* anytime algorithm for DEC-POMDPs [30]. The experiments with different values of $maxTrees$ illustrate the advantage of our algorithm as a contract algorithm.

5.1 Experimental Setting

In the experiments, we use two types of heuristics – the random policy heuristic and the MDP policy heuristic. With the random policy heuristic, the agents act randomly top-down to the current step and sample a set of belief states. With the MDP policy heuristic, the agents act according to a pre-computed MDP policy of the model. For the fairness of comparison, we used the same heuristic portfolio as the first recursion of other MBDP-based algorithms (MDP: 45%, Random: 55%). They run recursively using the complete solution of the previous recursion as a new part of the heuristic portfolio for the next run. However, as we demonstrate below, the quality of results produced by our algorithm could be further improved by using a better heuristic portfolio.

We performed re-sampling up to 10 times if the sizes of both agents' policies were less than $maxTrees$. Due to the randomness of the sampling, we ran the algorithm 10 times per problem and reported the average runtime and value. All timing results are CPU times with a resolution of 0.01 second. Many of the parameter settings we used are too large for PBIP-IPG to be able to produce an answer with a reasonable amount of time. An "x" in Table 2 means that the algorithm cannot solve the problem within 12 hours. As a reference, we also provide for each test problem an upper bound on the value based on solving the underlying full-observable MDP (V_{MDP}). Notice that this is a loose bound [18]. When the results we obtain are close to this value, we can be sure that they are near optimal. But when there is a big difference, it is not possible to know how close to optimal we are. PBPG was implemented in Java 1.5 and ran on a Mac OSX machine with 2.8GHz Quad-Core Intel Xeon CPU and 2GB of RAM available for JVM. Linear programs were solved using lp_solve 5.5 [1].

5.2 Test Problems and Results

The Meeting in a 3×3 Grid domain [8] is a classical benchmark for DEC-POMDP algorithms. In this domain, two robots navigate on a grid world and try to stay as much time as possible in the same grid location. We adopted the version of the problem used by Amato *et al.* [3], which has 81 states, 5 actions and 9 observations per robot. As shown in Table 2, our algorithm took much less time than PBIP-IPG with the same number of $maxTrees$ and got competitive values. Even with a relatively large number of $maxTrees$, e.g. $maxTrees$=20, our algorithm still ran faster than PBIP-IPG with $maxTrees$=3 and produced better value as expected.

[1] http://lpsolve.sourceforge.net/5.5/

Table 2: Experimental Results (10 trials)

$maxTrees$	PBIP-IPG		PBPG	
	Time	Value	Time	Value
Meeting in a 3×3 Grid, $\|S\| = 81,\ \|O\| = 9,\ T = 100$				
3	3084s	92.12	27.21s	87.01
10	x	x	201.50s	93.46
20	x	x	799.90s	93.90
50	x	x	8345.13s	94.79
100	x	x	52231.85s	95.42
$V_{MDP} = 96.08$				
Cooperative Box Pushing, $\|S\| = 100,\ \|O\| = 5,\ T = 100$				
3	181s	598.40	11.34s	552.79
10	x	x	69.12s	715.95
20	x	x	287.42s	815.72
50	x	x	2935.24s	931.43
100	x	x	19945.56s	995.50
$V_{MDP} = 1658.25$				
Stochastic Mars Rover, $\|S\| = 256,\ \|O\| = 8,\ T = 20$				
3	14947s	37.81	12.47s	41.28
10	x	x	59.97s	44.30
20	x	x	199.45s	45.48
50	x	x	987.13s	47.15
100	x	x	5830.07s	48.41
$V_{MDP} = 65.11$				
Grid Soccer 2×3, $\|S\| = 3843,\ \|O\| = 11,\ T = 20$				
3	x	x	10986.79s	386.53
$V_{MDP} = 388.65$				

PBPG could solve this problem with $maxTrees$=100 while PBIP-IPG ran out of time with $maxTrees \geq 10$. Note that this problem has 9 observations, which makes it more difficult to solve than the following two benchmark problems. The result for $maxTrees$=100 was actually near-optimal considering the loose upper bound V_{MDP}.

The Cooperative Box Pushing problem [23] is another common benchmark problem for DEC-POMDPs. In this domain, two agents are pushing three boxes (1 large and 2 small) in a 3×4 grid. The agents will get a very high reward if they cooperatively push the large box into the goal area together. This domain has 100 states and each agent has 4 actions and 5 observations. The results have been shown in Table 2. With $maxTrees$=3, our algorithm ran ten times faster than PBIP-IPG but got competitive value. As the number of $maxTrees$ increases, the solution became better as expected. In the experiments, we observed that with $maxTrees$=100, 99% of the iterations would do re-sampling about 10 times. It means that the heuristic portfolio has reached its limit to construct different policy trees when $maxTrees$=100. The frequent re-sampling in this case contributed to an increase in the runtime of PBPG. It is quite likely that 999.53, the highest value we got with $maxTrees$=100 in these experiments, is quite close to the optimal value for this problem with horizon 100.

The Stochastic Mars Rover problem [4] is a larger domain with 256 states, 6 actions and 8 observations for each agent. The runtime of PBIP-IPG is substantially larger than in the previous benchmarks due to the larger state space. We used in these experiments T=20 because it takes too much time for PBIP-IPG to solve problems with T=100. The horizon we used is the same as in the original PBIP-IPG paper [4]. In contrast, our algorithm scales better over the state space as well as the action and observation spaces. Surprisingly, our

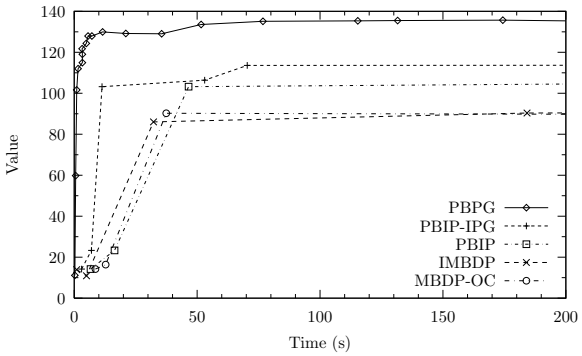


Figure 3: Value vs. Runtime for Cooperative Box Pushing with T=10. In these experiments, different algorithms used different numbers of $maxTrees$.

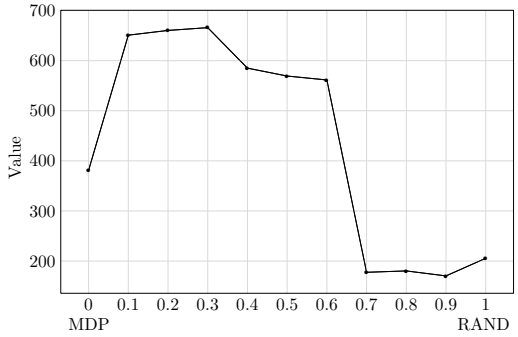


Figure 4: Values with different heuristic portfolios for the Cooperative Box Pushing problem with T=100, $maxTrees$=3. The composition of the portfolio changes gradually from the MDP heuristic only (left) to the random heuristic only (right).

algorithm can solve this problem rather quickly. Compared with the results in the Cooperative Box Pushing domain, runtime did not grow up substantially despite the fact that there are twice as many states. The main reason is that the transition and observation matrixes of this problem are quite sparse. Our algorithm with $maxTrees$=100 is still faster than PBIP-IPG with $maxTrees$=3. Again, the larger number of $maxTrees$ helps produce better solutions.

To test scalability, we also tried a more challenging problem introduced in [29], namely Grid Soccer 2×3. This problem has 3843 states, 6 actions and 11 observations. Currently, this problem can only be solved approximately by online algorithms. This is the first time that a problem of this size is solved successfully by an offline algorithm. Although online algorithms are generally very fast, offline algorithms can often provide higher solution quality. Besides, online algorithms often require the ability to communicate at runtime, which may be very costly or impossible in some domains. Offline algorithms have the added advantage of producing a coordinated joint policy prior to execution, reducing the need to communicate. In fact, our algorithm could get a value of 386.53 without any communication, which is near optimal. In comparison, the leading online algorithm called MAOP-COMM can only produce a value of 290.6 (without considering the cost of communication) while using communication 14.8% of the time on average [29].

The parameter $maxTrees$ does not only limit the usage of memory but also provides a tradeoff between the solution quality and runtime. Intuitively, an algorithm with larger $maxTrees$ will produce better value but also take longer time to execute. In our experiments, we tried different $maxTrees$ and recoded the runtime and value of each algorithm for the Cooperative Box Pushing domain with T=10. As shown in Figure 3, our algorithm got better value by given the same amount of time. Notice that the MDP upper bound [18] (V_{MDP}) of this problem with T=10 is 175.13. Thus, our algorithm performs quite well with respect to the value of the solutions. It is worth pointing out that all the algorithms we compared have linear time and space complexity with respect to the horizon. We chose a short horizon (T=10) here to make it possible for the other algorithms to solve the problem with different $maxTrees$ (1 to 8 as we tested, runtime >200s is not shown) in a reasonable amount of time; otherwise, they would run out of time very quickly.

In Figure 4 we show the results for the Cooperative Box Pushing problem with different heuristic portfolios. The x-coordinate indicates the frequency in which the random heuristic is used. The leftmost point (0) indicates that the random heuristic is not used at all and the MDP heuristic is used all the time. The rightmost point (1) indicates that the random heuristic is used all the time and the MDP heuristic is not used at all. We can see that the best heuristic portfolio for this domain is obtained at x=0.3. That is, the best results are obtained when 30% of the policies are selected using the random heuristic, and 70% using the MDP heuristic. The default portfolio used in the main experiments is (MDP: 45%, Random: 55%), which is also the portfolio used by the first recursion of other MBDP-based algorithms. Therefore, we could actually further improve the performance in Table 2 by using a better heuristic portfolio.

6. CONCLUSIONS

We present the point-based policy generation algorithm for finite-horizon DEC-POMDPs. Similar to previous MBDP-based algorithms, it also combines top-down heuristics and bottom-up dynamic programming to construct joint policy trees for an initial belief state. Our approach also uses the parameter $maxTrees$ to limit the usage of memory, thus it shares the same linear time and space complexity with respect to the horizon. The main contribution is a new point-based policy generation technique that builds the joint policies directly at each iteration, instead of performing a complex backup operation. By using this technique, many more policy trees can be kept as building blocks for the next iteration compared to the state-of-the-art algorithms. With a larger number of candidate subtrees, the solution quality can be further improved. Even when used with the same number of $maxTrees$, our algorithm runs orders of magnitude faster and produces competitive values in all the domains we tested. One important characteristic of the new algorithm is that it scales better over the state space and it can solve larger problems than currently possible with existing offline techniques. The experimental results show that the new PBPG algorithm significantly outperforms all the state-of-the-art algorithms in all the domains we tested.

In future work, we plan to incorporate other general methods such as observation compression and state reachability analysis into our algorithm to solve even larger problems. Currently, one limitation of our algorithm is that ev-

ery joint observation must be considered in order to model the problem as several linear programs. For some problems with large observation sets, additional techniques will have to be employed to further improve scalability. We are also investigating ways to learn the best heuristic portfolio automatically for different domains. The algorithm proposed in this paper eliminates much of the time- and space-consuming backup operation and opens up new research directions for developing effective approximation algorithms for multi-agent planning under uncertainty.

Acknowledgments

We thank Christopher Amato for providing the code of PBIP-IPG and the anonymous reviewers for their helpful comments. This work was supported in part by the China Scholarship Council, the Air Force Office of Scientific Research under Grant No. FA9550-08-1-0181, the National Science Foundation under Grant No. IIS-0812149, the Natural Science Foundations of China under Grant No. 60745002, and the National Hi-Tech Project of China under Grant No. 2008AA01Z150.

7. REFERENCES

[1] C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems*, 2009.

[2] C. Amato, A. Carlin, and S. Zilberstein. Bounded Dynamic Programming for Decetralized POMDPs. In *AAMAS 2007 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 2007.

[3] C. Amato, J. S. Dibangoye, and S. Zilberstein. Incremental Policy Generation for Finite-Horizon DEC-POMDPs. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling*, pages 2–9, 2009.

[4] C. Amato and S. Zilberstein. Achieving goals in decentralized POMDPs. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 593–600, 2009.

[5] R. Aras, A. Dutech, and F. Charpillet. Mixed Integer Linear Programming for Exact Finite-Horizon Planning in Decentralized Pomdps. In *Proc. of the 17th Int. Conf. on Automated Planning and Scheduling*, pages 18–25, 2007.

[6] R. Becker, S. Zilberstein, V. R. Lesser, and C. V. Goldman. Transition-independent Decentralized Markov Decision Processes. In *Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 41–48, 2003.

[7] D. S. Bernstein, C. Amato, E. A. Hansen, and S. Zilberstein. Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, 34:89–132, 2009.

[8] D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded Policy Iteration for Decentralized POMDPs. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence*, pages 1287–1292, 2005.

[9] D. S. Bernstein, S. Zilberstein, and N. Immerman. The Complexity of Decentralized Control of Markov Decision Processes. In *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, pages 32–37, 2000.

[10] A. Carlin and S. Zilberstein. Value-based observation compression for DEC-POMDPs. In *Proc. of the 7th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 501–508, 2008.

[11] J. S. Dibangoye, A. Mouaddib, and B. Chaib-draa. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 569–576, 2009.

[12] R. Emery-Montemerlo, G. J. Gordon, J. G. Schneider, and S. Thrun. Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 136–143, 2004.

[13] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 137–144. ACM, 2003.

[14] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic Programming for Partially Observable Stochastic Games. In *Proc. of the 19th National Conf. on Artificial Intelligence*, pages 709–715, 2004.

[15] R. Nair, M. Tambe, M. Roth, and M. Yokoo. Communication for Improving Policy Computation in Distributed POMDPs. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 1098–1105, 2004.

[16] R. Nair, M. Tambe, M. Yokoo, D. V. Pynadath, and S. Marsella. Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence*, 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 *Proc. of the 20st National Conf. on Artificial Intelligence*, pages 133–139, 2005.

[18] F. A. Oliehoek and N. Vlassis. Q-value functions for decentralized POMDPs. In *Proc. of the 6th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 833–840, 2007.

[19] F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 577–584, 2009.

[20] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Conference on Uncertainty in Artificial Intelligence*, pages 489–496, 2000.

[21] 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.

[22] M. Roth, R. G. Simmons, and M. M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *Proc. of the 4th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 786–793. ACM, 2005.

[23] S. Seuken and S. Zilberstein. Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs. In *Proc. of the 23rd Conf. in Uncertainty in Artificial Intelligence*, 2007.

[24] S. Seuken and S. Zilberstein. Memory-Bounded Dynamic Programming for DEC-POMDPs. In *Proc. of the 20th Int.l Joint Conf. on Artificial Intelligence*, pages 2009–2015, 2007.

[25] S. Seuken and S. Zilberstein. Formal Models and Algorithms for Decentralized Decision Making under Uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[26] D. Szer and F. Charpillet. Point-based Dynamic Programming for DEC-POMDPs. In *Proc. of the 21st National Conf. on Artificial Intelligence*, pages 1233–1238, 2006.

[27] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *Proc. of the 21st Conf. on Uncertainty in Artificial Intelligence*, pages 576–590, 2005.

[28] J. Tsitsiklis and M. Athans. On the complexity of decentralized decision making and detection problems. *IEEE Transaction on Automatic Control*, 30:440–446, 1985.

[29] F. Wu, S. Zilberstein, and X. Chen. Multi-Agent Online Planning with Communication. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling*, pages 321–328, 2009.

[30] S. Zilberstein. Optimizing Decision Quality with Contract Algorithms. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1576–1582, 1995.

Quasi Deterministic POMDPs and DecPOMDPs

Camille Besse & Brahim Chaib-draa
DAMAS Laboratory
Department of Computer Science and Software Engineering
Laval University, G1K 7P4, Quebec (Qc), Canada
{besse,chaib}@damas.ift.ulaval.ca

ABSTRACT

In this paper, we study a particular subclass of partially observable models, called quasi-deterministic partially observable Markov decision processes (QDET-POMDPS), characterized by deterministic transitions and stochastic observations. While this framework does not model the same general problems as POMDPS, they still capture a number of interesting and challenging problems and have, in some cases, interesting properties. By studying the observability available in this subclass, we suggest that QDET-POMDPS may fall many steps in the complexity hierarchy. An extension of this framework to the decentralized case also reveals a subclass of numerous problems that can be approximated in polynomial space. Finally, a sketch of ε-optimal algorithms for these classes of problems is given and empirically evaluated.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation

Keywords

partial observability, determinism, coordination

1. INTRODUCTION

AI planning was initially conceived as a deterministic problem where a sequence of actions has to be decided in order to achieve a goal state with desirable values from an original state. This problem was thoroughly studied in AI with important contributions as A*, GRAPHPLAN, and others [13].

However, this deterministic model has strong limitations on the type of problem that can be represented. Thus, one cannot represent situations where actions have non-deterministic outcomes or where states are not completely observable. In such cases, one must resort to Markov Decisions Processes (MDPS [15]) when the state is fully observable and Partially Observable Markov Decisions Processes (POMDPS [6]) otherwise. However, with this expressiveness comes an increase of complexity, specially for POMDPS, and thus this gain in generality involves a cost in the ability to solve the final problem. For instance, POMDPS offer one of the most expressive frameworks and are thus widely used for sequential decision making under partial observability [11], but the current known algorithms scale very poorly as the planning horizon grows. This reality is even more true in a decentralized setting where each agent has to anticipate every possible past and future of other agents.

Indeed, a major difficulty of decision-theoretic domains mainly depends on the definition of observability of agents in the problem. For example, considering the different markovian models (MDPS, POMDPS) and their various cooperative multiagent extensions (MMDPS [5], MTDP [17], DEC-POMDPS [3]), the difference between *fully* and *partially* observable models truly shows the exponential increase in worst-case complexity.

Nevertheless, numbers of problems that involve partial observability have a common characteristic: they have actions with deterministic outcomes and the observation generated is also deterministic. Indeed, these problems have recently been used in many proposals for planning with incomplete information, e.g. [14], and are used for learning partially observable models [1].

These models were briefly discussed in [12], under the name of deterministic POMDPS (DET-POMDPS) for which some important theoretical results were obtained. Littman first showed that a DET-POMDP can be mapped into an MDP with an exponential number of states and then be solved with standard algorithms for MDPS. Second, he showed that optimal non-stationary policies of polynomial size can be computed in non-deterministic polynomial time and finally that optimal stationary policies can be computed in polynomial space. Since then, up to our knowledge, no paper was published on this subject except [4] that extends these results by defining a specific subclass of DET-POMDPS, that have the so-called *polynomial diameter* property, that can be solved in non-deterministic polynomial time. Bonet also linked the DET-POMDP framework to the AND/OR tree search algorithms, arguing that this type of algorithm is more efficient than standard POMDP algorithms for this subclass of POMDPS.

Given this role of DET-POMDPS in recent research and motivated by the quest of amenable models for decision making under partial observability, we extend the work of Littman and Bonet in order to bridge a part of the gap between DET-POMDPS and POMDPS, by studying the subclass of POMDPS with deterministic transitions by actions and particular stochastic observations. We thus present a specific subclass of widely used POMDPS, called quasi-DET-POMDPS (QDET-POMDPS) and two particular subclasses of partial observability. A theoretical analysis suggests that ε-approximating these subclasses falls many steps in complexity in the polynomial hierarchy. We also extend these results to the multiagent case revealing a drastic improvement of the complexity in case of decentralized decisions.

This paper is organized as follows. First, examples of challeng-

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

ing problems are given (in the next section) that motivate our research, as mono or multiagent problems. In Sect. 3, a formal definition of the models and the variants of observability are given. In Sect. 4, main theoretical results are described and the complexity of the subclass is presented for both mono and multiagent models in Sect. 5. Finally, some experimental results are presented in Sect. 6 before discussing the significance of this work in Sect. 7.

2. EXAMPLES

Many problems have been modeled as POMDPs and DET-POMDPs and had been used for developing and evaluating various algorithms for planning under uncertainty and partial information. For space reasons, we present only few examples of some problems that may be modeled as a QDET-POMDP:

Diagnosis: The aim of diagnosis is to identify one of the m states of a system (e.g. a patient) using n noisy binary tests. An instance consists of a $m \times n$ stochastic matrix T where each T_{ij} represent the probability that test j is positive in the state i. The goal is to find the sequence of tests that will identify almost surely the state of the studied system [16]. In this example, the model is quasi-deterministic since the transition is deterministic (only one state) and observations are results of each test and are thus stochastic.

Indoor Robot: Consider an indoor arm robot working on a supply chain that moves boxes from one conveyor to another. Even if its actuators are nearly deterministic, its captors of surrounding activity may be noisy and may provide stochastic observations. In this domain we may also want to coordinate several robots and humans on complexe assembling tasks.

Fault Detection Consider a synthetic deterministic system that evolves through the interaction of an agent (e.g. a coffee machine). The aim of the problem is to identify the sequence of interactions that leads the system to fail while only receiving noisy or partial observations of its internal state. This problem is also extendable at the case where multiple agents interact at the same time with the system. Here again, transitions are deterministic while observations are stochastic.

All of these problems can be modeled as QDET-POMDPs or QDET-DEC-POMDPs. Let us now see the formal definition of the deterministic POMDP and its variants.

3. MODEL AND VARIANTS

Deterministic POMDPs were initially defined as follows [12]:

DEFINITION 1. *A Deterministic Partially Observable Markov Decision Process (*DET-POMDP*) is a tuple* $\langle \mathcal{S}, \mathcal{A}, \Omega, \mathcal{T}, \mathcal{O}, \mathcal{R}, \gamma, \boldsymbol{b}^0 \rangle$*, where:*

- $\mathcal{S}$ *is a finite set of* states $s \in \mathcal{S}$*;*
- $\mathcal{A}$ *is the finite set of* actions *of the agent and* $a \in \mathcal{A}$*, denotes an action;*
- Ω *is the finite set of* observations *of the agent and* $z \in \Omega$*, denotes an observation;*
- $\mathcal{O}(z, a, s') : \Omega \times \mathcal{A} \times \mathcal{S} \mapsto \{0, 1\}$ *is the* deterministic observation function *indicating wether or not the agent gets observation* z *when the world falls in state* s' *after executing action* a*;*
- $\mathcal{T}(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \{0, 1\}$ *is the* deterministic transition function *indicating wether or not making action* a *in state* s *results in state* s'*;*
- $\mathcal{R}(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ *is the* reward *perceived by the agent when the world falls into state* s *after executing action* a*;*
- γ *is the discount factor;*
- $\boldsymbol{b}^0$ *is the a priori knowledge about the state, namely the initial belief state, assumed non-deterministic, i.e. where no state has probability one.*

Note that the initial *belief state* $\boldsymbol{b}^0$, which describes the different possibilities for the initial state, is crucial. Indeed, if the initial state were known, and since the transition function is deterministic, then all the future states will also be known, and the DET-POMDP is then reduced to the well studied problem of deterministic planning in AI [13].

Compared to deterministic POMDPs, our proposed extended model presents changes on the observability function, it is called Quasi-deterministic Partially Observable Markov Decision Process and it is defined as follows:

DEFINITION 2. *A Quasi-deterministic Partially Observable Markov Decision Process (*QDET-POMDP*) is a tuple* $\langle \mathcal{S}, \mathcal{A}, \Omega, \mathcal{T}, \mathcal{O}, \mathcal{R}, \gamma, \boldsymbol{b}^0 \rangle$*, where:*

- $\mathcal{S}, \mathcal{A}, \Omega, \mathcal{T}, \mathcal{R}, \gamma, \boldsymbol{b}^0$ *are the same as in Definition 1;*
- $\mathcal{O}(z, a, s') : \Omega \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ *is the* observation function *indicating the probability of getting observation* z *when the world falls in* s' *after executing* a*;*
  Moreover, $\forall\, s' \in \mathcal{S}$, $a \in \mathcal{A}$, $\exists\, z \in \Omega$, *s.t.* $\mathcal{O}(z, a, s') \geqslant \theta > \frac{1}{2}$*, i.e. the probability of getting one of the observations is lower bounded in each state by at least one half;*

First, let us notice that θ is just a lower bound on the probability of observing each state and thus can be eventually greater in some states. Notice also that the planning horizon is not set *a priori*. This is due – as we will see in Section 4 – to an interesting convergence property of this model with some other assumptions to a very low entropy belief state after a fixed number of steps.

Second, to handle the multiagent case in both definitions, simply consider a set of agents where each agent i has its own action set $\mathcal{A}_i$ and where the joint action set $\boldsymbol{\mathcal{A}}$ is the product of all the agents' action sets. The transition and the observation functions are then just defined over the joint action set, and the condition on minimal observability is defined for all joint action $\boldsymbol{a}$ in $\boldsymbol{\mathcal{A}}$.

However, assuming that all agents have the same observability capacity (and hence the same observation space), and considering that in a QDET-DEC-POMDP there exist a most likely observation of the state whatever the chosen joint action is, in the same way as in the monoagent case, only the study of the QDET-POMDP is necessary from which we will extend to the multiagent case. For the ease of explanations, we will thus restrain the theoretical study to the monoagent case and then later in the paper, extend the results to the multiagent case.

Let us now see the optimality criteria and the variants of these models.

Optimality Criteria and Variants

As our goal is to compute a policy that permits an agent to perform *optimally*, we consider the **maxexp** optimality criteria that maximizes the expected discounted reward of a policy. The value of a

policy π is thus computed using:

$$V_\pi(\boldsymbol{b}^0) = \mathop{\mathrm{E}}_{s\sim\boldsymbol{b}^0}\left[\sum_{t=0}^{\infty}\gamma^t\mathcal{R}(s^t,\pi(s^t))\,\middle|\,s^0=s,\ \pi\right]$$

The considered variants of this model are related to the observation model as follows:

Unobservable models in which $|\Omega| = 1$ and thus no information is retrieved about the state. This class is a subclass of the so-called conformant problem in planning [9].

Fully Observable models in which $\Omega = \mathcal{S}$ and $\mathcal{O}(z,a,s') = 1$ iff $z = s'$. This class is exactly the classic fully observable MDPs where only the initial state is unknown (e.g. qMDP [11]).

Non-unobservable models in which $|\Omega| > 1$. This class is exactly the complement of unobservable problems. Among this class of problems, we distinguish:

- **Enough-observable models** in which $\Omega = \mathcal{S}$. This class regroups all the linear but noisy observation problems where the state itself is perceived but with an additive noise. This class regroups for example all control problems where the state is perceived through noisy sensors.
- **Factored-observable models** in which $|\Omega| = |\mathcal{X}| \times |\mathcal{D}_{\mathrm{X}}|$. Where $\mathcal{X}$ is the set of state variables and $\mathcal{D}_{\mathrm{X}}$ is the domain of variable X. The state space is then given by $\mathcal{S} = \bigtimes_{\mathrm{X}\in\mathcal{X}}\mathcal{D}_{\mathrm{X}}$. This class is similar to the previous one using additive noise but restricting the number of observations along the "dimensions" of the state space. Indeed, as the state space is assumed structured, the agent can use this structure to learn about at least one dimension at each time step. The previous class is equivalent to this class but with only one dimension.

General models which include previous cases, do not assume anything on the observation function.

As the fully observable, the unobservable and the general cases were extensively studied in the literature [13, 11], we will not consider them in the remaining of the paper. However, the enough-observable and the factored-observable cases present an interesting avenue since many of the quasi-deterministic problems mentioned earlier are very often factored or at least enough-observable.

We will show in the next section that these problems actually are easier than the general problems by bounding the history needed to identify the underlying state with high probability. Such bounds will indeed induce a complexity reduction of the problem and we will present this result in section 5.

4. THEORETICAL ANALYSIS

In this section, a lower bound on the number of steps to ensure convergence to a certain belief is given.

As mentioned earlier in the paper, a way to represent compactly the full history of observations during the planning process is the *belief state* [20]. This is a probability distribution over the states that represents the belief of the agent to be in each state through probabilities. We denote by $\boldsymbol{b}^t(s) = \Pr(s|z^t,a^t,\boldsymbol{b}^{t-1})$ the probability of being in state s at step t given that observation z^t was perceived and action a^t was performed in the belief state $\boldsymbol{b}^{t-1}$. This probability is computed using Bayes' rule:

$$\boldsymbol{b}^t(s) = \frac{\mathcal{O}(z^t,a^t,s)\sum_{s'\in\mathcal{S}}\mathcal{T}(s',a^t,s)\boldsymbol{b}^{t-1}(s')}{\sum_{s''\in\mathcal{S}}\mathcal{O}(z^t,a^t,s'')\sum_{s'\in\mathcal{S}}\mathcal{T}(s',a^t,s'')\boldsymbol{b}^{t-1}(s')} \tag{1}$$

Using a matrix representation, Equation (1) can be rewritten:

$$\boldsymbol{b}^k(s) = \frac{D_kT_{a^k}\cdots D_1T_{a^1}\boldsymbol{b}^0}{\mathbb{1}^\top D_kT_{a^k}\cdots D_1T_{a^1}\boldsymbol{b}^0} \tag{2}$$

Where $\boldsymbol{b}^0$ is the initial belief, T_{a^t} are transition matrices according to action a^t, D_i are diagonal matrices with the terms on the diagonal corresponding to the probability to observe z_i given each state, and $\mathbb{1}$ a $|\mathcal{S}|$-dimensional vector of ones.

In order to show the convergence of the belief state to a single state with high probability, let us first state that this probability depends on the number n of succeeded observations among k steps in a non-unobservable context. Nevertheless, non-unobservability is not a sufficient condition to ensure this convergence. Let us now study how n varies regarding to the proposed variants on the observability.

4.1 Enough-Observable models

Enough-observable models ensure that there is only one most likely observation (MLO) in each state and that each state's MLO is not the MLO of any other state:

DEFINITION 3. *An enough-observable* QDET-POMDP *is a* QDET-POMDP *where following assumptions holds:*

$$\exists o_1\in\Omega,\ \forall a\in\mathcal{A},\ \forall s\in\mathcal{S}^{o_1},$$
$$\textit{with } \mathcal{S}^{o_1} = \{s\in\mathcal{S}, o_1\in\Omega | P(o_1|s,a) > P(o|s,a), \forall o\neq o_1\},$$
$$\textit{then } |\Omega| = |\mathcal{S}| \textit{ and } |\mathcal{S}^{o_1}| = 1$$

Here, $\mathcal{S}^{o_1}$ is the set of states where o_1 is the MLO.

Considering this definition, one can state our first main result:

THEOREM 1. *Under the enough-observability assumption,* $\boldsymbol{b}^k(s) \geqslant 1-\varepsilon$ *iff*

$$n \geqslant \frac{1}{2\ln\frac{\nu\theta}{(1-\theta)}}\ln\left[\frac{1-\varepsilon}{\varepsilon}\left(1+\nu^{1-\frac{k}{2}}\right)\right] + \frac{k}{2} \tag{3}$$

Where $\nu = \max_{s,a}\sum_{z\in\Omega} I(\theta > \mathcal{O}(z,a,s) > 0) < |\Omega|$ *the maximum number of "bad" observations that can be perceived in a state.*

PROOF SKETCH. In the worst case, the probability of observing the real underlying state is always minimal and equals to θ at each step. Moreover, if the failed observations obtained always support the second most likely state, it results in an increasing of the probability to potentially be in this state. According to Equation (2) and using determinism of transitions, which induces that transition matrices are permutation matrices, one must show that:

$$\frac{\theta^n\frac{(1-\theta)^p}{\nu^p}}{\theta^n\frac{(1-\theta)^p}{\nu^p} + \theta^p\frac{(1-\theta)^n}{\nu^n} + (\nu-1)\frac{(1-\theta)^k}{\nu^k}} \geqslant 1-\varepsilon \tag{4}$$

Where n is the number of successful observations of the real underlying state and $p = k - n$ the number of failures. The numerator is obtained by obtaining n times a "good" observation and p times a "bad" one during the execution. The denominator sum over all states the same sequence of observation where the first term if for the most likely state, the second term for the second most likely state and the third term for the rest of possible states according to the number of "bad" observations ν. We assume here that the probability to get a "bad" observation is uniform. This assumption is justified by the *maximum-entropy principle* which states that according to the current knowledge, the highest entropy distribution – the uniform in our case – is the most appropriate one. Solving this inequality leads to Equation (3). The extensive derivation of the equations is given in Appendix A. □

Roughly speaking, ν represents also the way the error spreads over the false states and Theorem 1 states that if the observation is good enough (with a large probability θ to observe the real underlying state) and if the error spreads over many states (with a large ν), then it suffices to have one half of the observations plus one to be the real underlying state to converge to a deterministic belief state.

The uniform assumption on bad observations is justified by the maximum entropy principle but can be omitted without loss of generality. Indeed, taking ν equals to 1 induces that the agent observes either the true underlying state or the second most likely state of his belief which is truly the worst case for our agent. The resulting equation would be exactly the same as equation (3) but where the term $\nu^{1-\frac{k}{2}}$ would be also equals to one which would then slightly loosen the bound.

4.2 Factored models

In a more general way than enough-observable models, factored-observable models ensure that each value of each variable is sufficiently often observed so that the factored state can de determined in a finite number of steps:

DEFINITION 4. *A factored-observable* QDET-POMDP *is a* QDET-POMDP *where following assumption holds:*

- *The state space is* factored *in μ state variables: $\mathcal{S} = \bigtimes_{\mathrm{x}\in\mathcal{X}} \mathcal{D}_{\mathrm{x}}$ and observations possible are $\Omega = \bigcup_{\mathrm{x}\in\mathcal{X}} \mathcal{D}_{\mathrm{x}}$.*
- *The sum of probabilities of observing one state's variables' real values is lower bounded by $\theta > \frac{1}{2}$.*

This definition implies that, in the worst case, for each state variable, there is a probability $\frac{\theta}{\mu}$ to observe its real value and a probability $\frac{1-\theta}{|\Omega|-\mu}$ to observe anything else. Note also that this definition is a generalization of Definition 3 which is the case $\mu = 1$. This statement leads to the following theorem:

THEOREM 2. *Under the factored-observability assumption, $\boldsymbol{b}^k(s) \geqslant 1-\varepsilon$ iff*

$$n \geqslant \frac{1}{2\ln\frac{(|\Omega|-\mu)\theta}{\mu(1-\theta)}} \ln\left[\frac{1-\varepsilon}{\varepsilon}\left(1+|\mathcal{S}|-\mu\right)\right] + \frac{k}{2} \qquad (5)$$

PROOF SKETCH. The proof follows exactly the same arguments as in Theorem 1. □

Once the number n of most likely observations is lower bounded, finding the probability to achieve at least this number is simply an application of the binomial distribution to have at least n successes on k trials:

COROLLARY 3. *In any* QDET-POMDP *under enough-observability* ***or*** *factored-observability assumptions, the probability that a belief state $\boldsymbol{b}^k(s)$ is ε-deterministic after k steps is:*

$$\exists s, \Pr(\boldsymbol{b}^k(s) \geqslant 1-\varepsilon) = \sum_{i=n}^{k} \binom{k}{i} \theta^i (1-\theta)^{k-i} \qquad (6)$$

In other words, this indicates that to be certain (with a small δ) to have a deterministic belief state (with a small ε) we may have to explore a large horizon if θ is too small (e.g. near 0.5).

Let us now derive the worst case complexity from these bounds.

5. COMPLEXITY ANALYSIS

In this section, new complexity results induced from Theorems 1 and 2 are given.

5.1 Mono-agent case

A major implication of Theorems 1 and 2 is the reduction of the complexity of general POMDPs problems when a QDET-POMDP is encountered. Indeed, [15] have shown that finite-horizon POMDPs are PSPACE-complete. This roughly speaking rests on the fact that the agent has to choose an action that, given any observation, leads to the choice of another action and so on, on a polynomially bounded horizon T. However, fixing the horizon T to be constant, causes to complexity to fall down many steps in the polynomial hierarchy [21]. Polynomial hierarchy consists in the generalized class of problems that uses oracles. Stockmeyer [21] defined $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$ as the class of decision problems that can be solved in polynomial time by a non-deterministic Turing machine using a NP-oracle. The "canonical" problem for this complexity class (which is SAT for NP) is 2-QBF for Σ_2^{P}; 2-QBF is the problem of deciding whether the following quantified boolean formula is true: $\exists\vec{a}\forall\vec{b}\,\phi(\vec{a},\vec{b})$. Stepping up in the polynomial hierarchy (e.g. Σ_3^{P}) means adding another quantifier for another set of variables of the Boolean formula (e.g. verifying if $\exists\vec{a}\forall\vec{b}\exists\vec{c}\,\phi(\vec{a},\vec{b},\vec{c})$ is true); and so on. Thus, in the case of constant horizon POMDP, one can state:

PROPOSITION 4. *Finding a policy for a finite-horizon-k* POMDP, *that leads to an expected reward at least C is $\Sigma_{2k-1}^{\mathrm{P}}$.*

PROOF. To show that the problem is in $\Sigma_{2k-1}^{\mathrm{P}}$, the following algorithm using a $\Sigma_{2k-2}^{\mathrm{P}}$ oracle can be used: guess a policy for $k-1$ steps with the oracle and then verify that this policy leads to an expected reward at least C in polynomial time by verifying the $|\Omega|^k$ possible histories, since k is a constant. □

As QDET-POMDPs are a subclass of POMDPs and since fixing $1-\delta$, the wanted probability to be in a ε-deterministic belief state, induces a constant horizon under enough-observability or factored-observability assumptions:

COROLLARY 5. *Finding a policy for an infinite horizon* QDET-POMDP, *under enough-observability or factored-observability assumptions, that leads to an expected reward at least C with probability $1-\delta$, is $\Sigma_{2k-1}^{\mathrm{P}}$.*

PROOF. To show that this problem is in $\Sigma_{2k-1}^{\mathrm{P}}$, the following algorithm gives the ε-optimal policy in polynomial time assuming a $\Sigma_{2k-2}^{\mathrm{P}}$ oracle: guess a policy for $k-1$ steps with the oracle and then verify that this policy leads to an expected reward at least C by computing the belief in each leaf of the tree of observations and by adding the optimal expected value of the underlying MDP since the belief is deterministic in each of these leaves. □

Practically, finding a probably approximatively correct ε-optimal policy for a QDET-POMDP thus implies using a k-QMDP algorithm that computes exactly k exact backups of a POMDP and that then uses the policy of the underlying MDP for the remaining steps (eventually infinite).

To sum up, by fixing the wanted probability (1-δ) to be in a ε-deterministic belief state, one can upper-bound the horizon on which it is necessary to plan, from which one can ensure that following the optimal policy of the underlying POMDP will perform well. Now, let us see how can this result can be extended to decentralized decision making.

5.2 Multi-agent case

Concerning the DEC-POMDPs, the improvement is much greater. Indeed, DEC-POMDPs are known to be exceptionally hard to solve optimally in the finite horizon case (NEXP-complete [3]) and even to approximate [18]. Moreover, these approximate solutions of

θ	ν	k	$n \geqslant$
0.6	3	75	40
0.6	10	59	31
0.6	100	50	26
0.7	3	22	13
0.7	10	19	11
0.7	100	14	8
0.8	3	9	6
0.8	10	6	4
0.8	100	6	4

Table 1: Enough-Observable bound.

θ	μ	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{S}\rvert$	k	$n \geqslant$
0.6	2	10	100	84	44
0.6	3	5	125	98	52
0.6	10	6	10^6	112	60
0.7	2	10	100	30	17
0.7	3	5	125	33	19
0.7	10	6	10^6	39	23
0.8	2	10	100	13	8
0.8	3	5	125	16	10
0.8	10	6	10^6	20	13

Table 2: Factored-observable bound.

the infinite horizon general case, usually based on finite state controllers [2], converge to local optima without any guarantee on the solution found.

By restricting the model to be quasi-deterministic, and assuming that all agents still have enough-observability or factored-observability, one can also find a great complexity reduction:

COROLLARY 6. *Finding a policy for an infinite horizon* QDET-DEC-POMDP*, under enough-observability or factored-observability assumptions, that leads to an expected reward at least C with probability $1-\delta$, is* PSPACE.

PROOF. To show that this problem is PSPACE, the following algorithm gives the ε-optimal policy in polynomial space: expand all possible action-observation history up to horizon k (done in space $\mathcal{O}((|\boldsymbol{\mathcal{A}}||\boldsymbol{\Omega}|)^k)$), and then compute for each leaf of the constructed tree the expected value of the reached quasi-deterministic belief using the underlying MMDP infinite horizon optimal policy. Finally, propagate the value back to the root to verify if an expected reward of C is obtained. □

Note that the assumption of enough-observability means here that each agent perceives the same *complete* state while not necessarily obtaining the same observation. On one hand, this assumption seems less applicable in DEC-POMDPs than in POMDPs since many internal values of the agents are also in the joint state of the DEC-POMDP and thus are not necessarily observable as the example in the next Section will illustrate. On the other hand, assuming a quasi-reliable communication system between agents is not so restrictive and induces naturally the enough-observability assumption by encompassing the communication noise into the observation noise.

In fact, the enough-observability assumption relates closely to the work of Goldman *et al.* [8] on DEC-MDPs where agents, when communicating their observations, have access to the real underlying state. This nonetheless differs on one point; while in DEC-MDPs each agent observes completely its own part of the state, granting the set of agents the complete observability of the state through communication, enough-observability assumption only assure that each agent *may* observe the true underlying state with probability at least θ. In other words, a DEC-MDP with complete communication is a DEC-POMDP under the enough-observability assumption with $\theta = 1$.

Let us now put figures on Theorems 3 and 4 and present a new decentralized fire fighting problem that meets our assumption requirements.

6. EXPERIMENTAL ANALYSIS

In this section we first provide examples of horizon that can be induced by the proposed bounds and then provide an example where such bounds could be applied.

6.1 Bounds' efficiency

To give an idea of the efficiency of the proposed bounds, we define $\delta > 0$ such that $\Pr(\boldsymbol{b}^K(s) \geqslant 1-\varepsilon) \geqslant 1-\delta$. Table 1 and 2 give, for $\varepsilon = 10^{-3}$, $\delta = 10^{-1}$ and different values of θ, the probability of observation, ν, the error spreading factor, μ, the number of state variables, and the domains' size of variables, the value of the bound on the horizon k and the number of successes needed n given that the probability of having both is above $1-\delta$.

As expected, horizons needed to converge are greater in the factored case than in the enough-observable case for similar state and observation spaces since the agent, at each time step, gets less information about the current state. Actually, observations discriminate among subsets of states but not among states themselves like in the previous case. However, as the number of observations is much less than in the previous case, current algorithms may have less difficulty in this type of problems. An empirical study of their difference should be interesting as a research avenue[1].

Furthermore, it is interesting to notice that, as soon as the probability θ to observe the real underlying state is above 0.8 in the enough-observable case, it suffices to compute the optimal policy for the first four steps to have a ε-deterministic belief state with at least 90% probability.

6.2 Infinite Horizon QDET-DEC-POMDP

As suggested by the complexity proofs of corollary 5 and 6, an algorithm that would solve ε-optimally the infinite horizon QDET-DEC-POMDP problem under enough observability or factored observability consists in computing a regular policy for the finite k-horizon problem and then use the optimal policy of the underlying MMDP for the remaining of the task. Such an algorithm would use for example the Dynamic Programming method [10]. However, in practice, solving a k-horizon DEC-POMDP even quasi-deterministic is still very hard and some other approximation should be used. We thus present in this section some results of a specific fire fighting problem where agents have to coordinate to make a bucket chain from a fire hydrant to a fire (e.g. Figure 4).

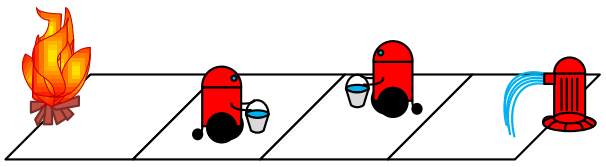


Figure 4: The bucket chain problem.

[1]Comparing results for the fire fighting problem between factored and enough observable models will be available in the final version of the paper.

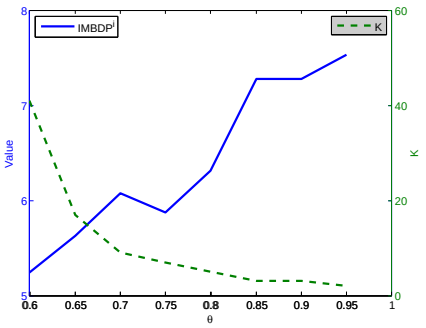


Figure 1: Expected value and estimated horizon for different value of θ.

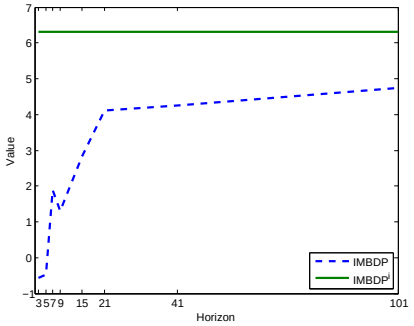


Figure 2: Finite horizon discounted expected value for various horizon lengths.

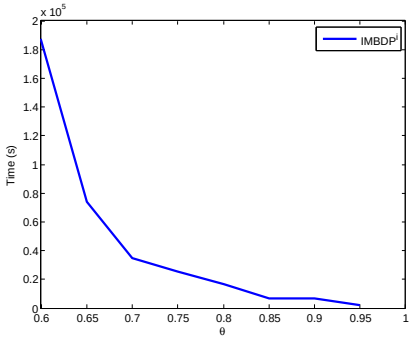


Figure 3: Computational time for different value of θ.

The general version of this problem is stated as follows: two agents are located on a linear grid and can carry a bucket. They can go *right*, *left*, or *throw water*, each action incurring a small penalty (-0.1 per agent). As soon as they go on the rightmost place, they automatically refill their bucket with probability φ. Moving actions have also a probability φ to succeed, the action *throw water* is always successful. Agents can give a bucket one to each other when on an adjacent place through the same *throw water* action. Each time a bucket is emptied on the most left place, a reward is obtained (1 per agent). Agents are initially placed at random without any water. Agents are assumed to have a noisy observation on their position (and have a θ probability to observe it and $\frac{1-\theta}{2}$ to observe one of the adjacent state) as well as they receive a noisy communication of the other agent indicating its position and its bucket state. As the problem could be an infinite horizon problem a discount factor $\gamma = 0.95$ has been used in the experiments. A value $\varphi = 1$ was used to meet the deterministic transition requirements. When $\varphi = 1$, the number of reachable states is 49 (7 for each agent) and thus 49 observations can be obtained.

Concerning the algorithm, we used an adapted version of Improved Memory Bounded Dynamic Programming (IMBDP) [19] so that it used the optimal expected value of the underlying MMDP at the leaves of the tree search. This algorithm is denoted by IMBDP^i on figures.

We ran several simulations using various parameters. Figure 1 shows the expected of IMBDP^i on the bucket chain problem for various values of θ ranging from 0.6 to 0.95. Parameters of IMBDP^i were $maxTree = 5$ and $maxObs = 1$. Increasing these parameters does not significantly improve the expected value while significantly deteriorating the time and space performances. As expected, as long as θ increases to one, the necessary planning horizon decreases to two and the infinite horizon expected value increases near to the value of the underlying MMDP's optimal policy.

Figure 2 shows the finite horizon discounted expected values for $\theta = 0.8$ at various horizons ranging from 3 to 101 showing that the algorithm tends to the infinite horizon expected value. The standard IMBDP algorithm was used with the same parameters as above ($maxTree = 5$ and $maxObs = 1$).

Finally, Figure 3 shows the computational time needed to compute an approximate infinite horizon policy using the exact same parameters for the algorithm. As expected, the time decreases as θ grows since the needed horizon also decreases.

Let us now discuss the different assumptions of the proposed models and their significance.

7. DISCUSSION

Quasi-deterministic models encompass numerous decision problems where the environment is well defined and controlled but just partially observed. The presented results can then be applied in a large number of applications ranging from web agents to fault detection systems. However, the approach still has some limitations.

First of all, presented results are probabilistic. Agents are guaranteed to converge to an ε-deterministic belief state with probability $1 - \delta$. Diminishing the probability δ induces an exponential increase of the planning horizon k and thus an increase of the computational needs.

Second, deterministic transition functions are not the mostly used transition functions in the Markovian community since its probabilistic roots. However, many real problems are in fact deterministic but do not use Markovian models for the same reason.

Third, We did not talk about policies nor actions throughout the paper, this relies on the fact that we assume that observation perceived are the same whatever the action is chosen. Relaxing this assumption leads to the problem of balancing information gathering and reward gathering like the *exploration-exploitation* dilemma in reinforcement learning. A concern that is beyond the scope of this paper but that we intend to prospect in future researches.

Last, it is not clear how the assumptions on the observability of the agents restrain the field of applications of this work although it is clear that it may be applied in many indoor robotics applications where each sensor is very often dedicated to one component of the state of the robot, independently of the policy chosen.

Concerning the observability assumptions, they are the key point of this paper. The majority of problems in the literature assume either full observability or partial observability. This paper proposes two other classes of observability that fill in the gap between these two extremes. Indeed, many of real problems are enough-observable or even factored-observable and one usually has to assume partial observability in these cases. This proposition of a new subclass of partial observability may stimulate the development of specific algorithms for these subclasses based on and/or graphs for example [4].

Moreover, in the multiagent case, the proposed work assumed some sort of communications between agents, allowing them to exchange their partial view of the world in order to provide each agent a noisy but global view of the state of the system. Even if this kind of communication is very often employed in true applications of multiagent systems, all previous works on communication in decentralized POMDPs (e.g. [7]) focused on the act of communicating and not on the goals of doing so nor on the content of the

exchanged messages. Indeed, a natural insight of communicating is to inform others what they do not know. By communicating its own observations to all other agents, the problem of decentralized POMDP would reduce to a "simple" multiagent POMDP where every single agent shares the same belief state and thus does not have to plan for all over possible policies of other agents. This work thus opens many interesting communication modeling avenues in multiagent Markovian models where multiagent does not necessarily means NEXP-complete.

8. CONCLUSION AND FUTURE WORK

To summarize, we proposed in this paper an extension of the DET-POMDP framework to stochastic observability, called QDET-POMDP, that bridges a part of the gap between DET-POMDPs and general POMDPs. A further extension to multiagent systems has also been proposed. A study of their convergence properties leads to a significant improvement in terms of computational complexity. Empirical performances are also presented through a new decentralized problem of fire fighting with communication between agents.

Concerning future work, we are currently working on extending current bounds on the horizon to problems where transitions are stochastic but can still be lower bounded. Model and algorithms that take communication of agents into account are also one of our research avenues. Finally, in our opinion, the problem of balancing information-reward gathering should be one of the next research areas in the community studying partially observable environments and models.

9. REFERENCES

[1] E. Amir and A. Chang. Learning Partially Observable Deterministic Action Models. *J. Artif. Intell. Res.*, 33:349–402, 2008.

[2] D. S. Bernstein, C. Amato, E. A. Hansen, and S. Zilberstein. Policy Iteration for Decentralized Control of Markov Decision Processes. *J. of AI Res. (JAIR)*, 34:89–132, 2009.

[3] D. S. Bernstein, S. Zilberstein, and N. Immerman. The Complexity of Decentralized Control of Markov Decision Processes. In *Proc. of UAI*, pages 32–37, 2000.

[4] B. Bonnet. Deterministic POMDPs Revisited. In *Proc. of Uncertainty in AI*, 2009.

[5] C. Boutilier. Sequential Optimality and Coordination in Multiagent Systems. In *Proc. of the Inter. Joint Conf. on AI*, pages 478–485, 1999.

[6] A. Cassandra, L. Kaelbling, and M. Littman. Acting Optimally in Partially Observable Stochastic Domains. In *Proc. of Assoc. for the Adv. of AI*, pages 1023–1028, 1994.

[7] C. V. Goldman, M. Allen, and S. Zilberstein. Decentralized Language Learning through Acting. In *Proc. of the Inter. Joint Conf. on AAMAS*, pages 1006–1013, 2004.

[8] C. V. Goldman and S. Zilberstein. Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis. *J. Artif. Intell. Res.*, 22:143–174, 2004.

[9] R. P. Goldman and M. S. Boddy. Expressive planning and explicit knowledge. In *Proc. of the 3rd Inter. Conf. on Artif. Intel. Planning Systems*, pages 110–117, 1996.

[10] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic Programming for Partially Observable Stochastic Games. In *Proc. of Assoc. for the Adv. of AI*, pages 709–715, 2004.

[11] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[12] M. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Dept. of Comp. Sc., Brown University, 1996.

[13] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[14] H. Palacios and H. Geffner. From Conformant into Classical Planning: Efficient Translations that May Be Complete Too. In *Proc. of the 17th Int. Conf. on Automated Planning and Scheduling*, pages 264–271, 2007.

[15] C. Papadimitriou and J. Tsisiklis. The Complexity of Markov Decision Processes. *Math. Oper. Res.*, 12(3):441–450, 1987.

[16] K. Pattipati and M. Alexandridis. Application of Heuristic Search and Information Theory to Sequential fault Diagnosis. *IEEE Trans. on Sys., Man and Cyber.*, 20(4):872–887, 1990.

[17] D. V. Pynadath and M. Tambe. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *J. Artif. Intell. Res.*, 16:389–423, 2002.

[18] Z. Rabinovich, C. V. Goldman, and J. S. Rosenschein. The Complexity of Multiagent Systems: the Price of Silence. In *Proc. of AAMAS*, pages 1102–1103, 2003.

[19] S. Seuken and S. Zilberstein. Improved Memory-Bounded Dynamic Programming for Dec-POMDPs. In *Proc. of Uncertainty in AI*, 2007.

[20] E. J. Sondik. *The optimal control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

[21] L. J. Stockmeyer. The Polynomial-Time Hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

10. APPENDIX A

PROOF OF THEOREM 1.

$$\frac{\theta^n \frac{(1-\theta)^p}{\nu^p}}{\theta^n \frac{(1-\theta)^p}{\nu^p} + \theta^p \frac{(1-\theta)^n}{\nu^n} + (\nu-1)\frac{(1-\theta)^k}{\nu^k}} \geqslant 1-\varepsilon$$

$$\Leftrightarrow \quad \frac{\nu^n\theta^n(1-\theta)^p}{\nu^n\theta^n(1-\theta)^p + \nu^p\theta^p(1-\theta)^n + (\nu-1)(1-\theta)^k} \geqslant 1-\varepsilon$$

$$\Leftrightarrow \quad \frac{\nu^p\theta^p(1-\theta)^n}{\nu^n\theta^n(1-\theta)^p} + \frac{(\nu-1)(1-\theta)^k}{\nu^n\theta^n(1-\theta)^p} \leqslant \frac{1}{1-\varepsilon} - 1$$

$$\Leftrightarrow \quad \nu^{k-2n}\theta^{k-2n}(1-\theta)^{2n-k} + (\nu-1)\frac{\theta^{-n}\nu^{-n}}{(1-\theta)^{-n}} \leqslant \frac{\varepsilon}{1-\varepsilon}$$

$$\Leftrightarrow \quad \frac{\nu^{k-2n}\theta^{k-2n}}{(1-\theta)^{k-2n}}\left[1 + (\nu-1)\frac{\nu^{-p}\theta^{-p}}{(1-\theta)^{-p}}\right] \leqslant \frac{\varepsilon}{1-\varepsilon}$$

$$\Leftrightarrow \quad (k-2n)\ln\frac{\nu\theta}{(1-\theta)} + \ln\left[1 + (\nu-1)\frac{\nu^{-p}\theta^{-p}}{(1-\theta)^{-p}}\right] \leqslant \ln\frac{\varepsilon}{1-\varepsilon}$$

$$\Leftrightarrow \quad (k-2n)\ln\frac{\nu\theta}{(1-\theta)} \leqslant \ln\frac{\varepsilon}{1-\varepsilon} - \ln\left[1 + (\nu-1)\frac{(1-\theta)^p}{\nu^p\theta^p}\right]$$

$$\Leftrightarrow \quad (2n-k)\ln\frac{\nu\theta}{(1-\theta)} \geqslant \ln\frac{1-\varepsilon}{\varepsilon} + \ln\left[1 + (\nu-1)\frac{(1-\theta)^p}{\nu^p\theta^p}\right]$$

$$\Leftrightarrow \quad (2n-k) \geqslant \frac{\ln\frac{1-\varepsilon}{\varepsilon}}{\ln\frac{\nu\theta}{(1-\theta)}} + \frac{1}{\ln\frac{\nu\theta}{(1-\theta)}}\ln\left[1 + (\nu-1)\frac{(1-\theta)^p}{\nu^p\theta^p}\right]$$

$$\Leftrightarrow \quad n \geqslant \frac{\ln\frac{1-\varepsilon}{\varepsilon}}{2\ln\frac{\nu\theta}{(1-\theta)}} + \frac{1}{2\ln\frac{\nu\theta}{(1-\theta)}}\ln\left[1 + (\nu-1)\frac{(1-\theta)^p}{\nu^p\theta^p}\right] + \frac{k}{2} \quad (7)$$

but since $2 \leqslant \nu \leqslant |\mathcal{S}| - 1$, $n > \frac{k}{2}$ and $\frac{1-\theta}{\theta} < 1$,

$$\begin{aligned}\ln\left[1 + \frac{|\mathcal{S}|-2}{\nu^{k-n}}\frac{(1-\theta)^{k-n}}{\theta^{k-n}}\right] &\leqslant \ln\left[1 + \frac{|\mathcal{S}|-2}{\nu^{\frac{k}{2}}}\left(\frac{1-\theta}{\theta}\right)^{\frac{k}{2}}\right] \\ &\leqslant \ln\left[1 + \nu^{1-\frac{k}{2}}\right]\end{aligned}$$

From which, ensuring Eqn. (3) also ensures Eqn. (7), □

Sequential Bayesian Decision Making for Multi-Armed Bandit

R. E. McInerney, S. J. Roberts
Pattern Analysis and Machine Learning Research Group
Department of Engineering Science
University of Oxford
Oxford OX1 3PJ UK
{robmc,sjrob}@robots.ox.ac.uk

I. Rezek
Clinical Neuroscience Department
Imperial College London
London
SW7 2AZ
UK
i.rezek@imperial.ac.uk

ABSTRACT

Most approaches to multiple agent decision making, or single agent multi-action decisions, tacitly assume a stationary reward environment in which ϵ-greedy approaches can asymptotically give valid information regarding action-value policies. We extend this approach by taking a Bayesian methodology to multi-armed bandits in which natural balance between exploitation and exploration takes place. Crucially we do not collapse information until all unknowns can be integrated out together. Uncertainty at all levels is automatically incorporated into the decision process, occasionally resulting in decisions being made which differ from the standard maximum likelihood approach. This framework can be extended with ease to incorporate utility functions which combine expected future reward and penalise model ignorance as well as offering a game theoretic framework in which multiple agents interact.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence-Learning

General Terms

Algorithms, Reliability, Theory

Keywords

Sequential Bayesian Decisions, Uncertainty, Bandit problems, expected reward, Greedy strategies, St. Petersburg paradox

1. INTRODUCTION

Any agent operating in a system must make decisions in order to select a course of action, whether that be deciding which direction to travel or how to classify a set of information. As such, decision making is an integral part in the design of autonomous systems, linking information and inference to outputs. Sequential decision making is simply decision making applied over time. Typically an agent will, at a certain time period, take an action in order to maximise an expected utility using the information and observations it has available at that point. Once an action has been executed an expected reward is earned and the process repeats itself. The problem of interest is to choose a decision strategy that will lead to high long term rewards. As the agent will often be faced with limited information and uncertainty we use the tools of probability theory, specifically Bayesian probability theory, to generate our solutions.

The multi-armed bandit, originally introduced in the statistics literature under the title "the sequential design of experiments" [12, 13, 10], is a sequential decision problem analogous to a traditional slot machine, or one-armed bandit, except that there are n levers to choose from instead of one. The agent, or player, is faced repeatedly with a choice between n different actions, $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ analogous to lever pulls. When an action is taken, a reward is drawn from a probability distribution associated with that lever. The problem is formally equivalent to a one-state Markov Decision Process.

In the *opaque* bandit, which we consider here, the agent receives a unique reward after each action it takes drawn from the probability distribution associated with that action. This is distinct from the *transparent* bandit [5] in which the agent observes all rewards including those it would have received had it taken alternative actions. We regard this opacity as typical of the majority of real-world problems.

It is normally the task of the agent to sample sequentially from the n hidden reward distributions in order to maximise the expected cumulative reward; defined as:

$$R = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots] = \mathbb{E}\left[\sum_{i=0}^{\mathcal{H}} \gamma^i r_{t+i}\right] \quad (1)$$

where r_t is the reward received after choosing an action a_k at time t, and $\gamma \in [0, 1]$ is a discount factor used to weight near term reward more heavily than distant future reward. The horizon $\mathcal{H}$ is the total number of actions the agent is able to take.

The agent begins with no knowledge about the levers but through repeated play can make inferences about the structure and parameters of the hidden reward distributions. As the agent has uncertainty associated with these distributions it must at each iteration decide between *exploitation* of the actions that have the highest expected rewards according to its current knowledge and *exploration* of other actions in order to reduce the associated uncertainty and perhaps find a more lucrative reward distribution. This is known in the decision making literature as the exploitation versus

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

exploration tradeoff.

Most bandit literature uses one of two forms of reward distribution: Bernoulli or Gaussian. In the Bernoulli Bandit reward is either termed success or failure, with success being achieved with a certain probability for each action (a Bernoulli process). All successes have equal value, as do failures. The expected reward is equivalent to the probability of success, rather than the expected value of reward received. This form is analogous to flipping weighted coins with unknown biasing with success being defined as receiving a head (or tail). In the second form, the Gaussian Bandit, reward is distributed according to a Gaussian distribution. Each reward received has a value, with higher values being favourable to lower values. It is of course possible to define negative rewards in this framework. Gaussian distributions are used to simplify analysis, but in reality any continuous probability distribution could be used within a similar strategy framework. Different techniques are required to solve and generate strategies for each form of reward distribution. We concentrate on the Bernoulli Bandit here as it allows us to focus on how an agent manages uncertainty without the complication of continuous reward. To our knowledge [6] provides the only fully Bayesian strategy for the Gaussian Bandit. That paper still only managed a two step look-ahead implementation due to the inherent complexity of the problem.

2. INADEQUACY OF EXPECTED REWARD

In order to design a policy an agent usually evaluates the likely outcome of actions, assigning a *value* to each action available. In sequential decision making literature a large proportion of research makes use of expected rewards or utilities to evaluate actions, making decisions by favouring higher expected values.

An expected value is the integral of a random variable with respect to its probability measure. It is important to note that expected value is *not* the same as most probable value, and in general may not be equal to a value the random variable can in fact take.

As designers of autonomous agents we must be aware of the difference between expected and real resources. To illustrate the importance of this point consider a simple problem in which a decision maker is offered the choice of taking £10 with certainty or £1000 with probability 1/80. The expected payout when taking the gamble is £12.50, and as such a decision maker using a policy that favoured higher expected payouts would choose this option. However, it is not possible to receive a payout of £12.50, only the payouts £0, £10 and £1000 are possible. Only if this problem is repeated many times does the expected payoff become closer to actual payoff, so how should one make a decision in a one-shot game?

2.1 The St. Petersburg Paradox

The St. Petersburg Paradox is a paradox introduced by Nicolas Bernoulli in 1713 [2] in which a theoretical lottery possesses an infinite expected payoff, but would still only be worth a small amount of money. It has been used as a means to demonstrate how a naive decision criterion, which only considers expected payoff maximisation, would suggest a course of action that a rational thinking person would never be willing to take.

To enter the lottery a player pays a fixed entry fee, at which point a fair coin is repeatedly tossed until a tail appears signalling the end of the game. Initially the player starts with 1 Ducat[1] in the bank which is doubled every time a head appears. At the end of the game the player wins whatever is in the bank. So if the coin is tossed n times before a tail appears the player wins 2^{n-1} Ducats. The problem is to decide what a fair price of entry is to such a lottery. It could be argued that a sensible gambler would enter the lottery if and only if the entrance fee was less than the expected payout, however the expected payout evaluates as follows:

$$\begin{aligned}\mathbb{E}[r] &= \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{8}\cdot 4 + \dots \\ &= \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \dots \\ &= \sum_{i=1}^{\infty} \frac{1}{2} = \infty \end{aligned} \tag{2}$$

Using the expected payout criterion, the sensible gambler would play the lottery no matter how large the entrance fee was because the expected return would always be higher. However, the probability of very high returns is very small. For example a run of 10 heads would payout over 1000 Ducats but the probability of that occurring is less than 0.001. There is a 50% chance the gambler would receive 2 Ducats from the game, and a 75% chance of receiving 4 Ducats or less. Hacking suggested that "few of of us would pay even 25 (Ducats) to enter such a game" [4].

Several approaches have been suggested as solutions to the St. Petersburg Paradox, the most famous of which, proposed by Nicolas Bernoulli's cousin Daniel Bernoulli, is the principle of *Expected Utility Hypothesis* and the presumption that money has a decreasing marginal utility [1]. Bernoulli suggested that a realistic measure of a gambler's utility would be a function of the gambler's wealth (w) and the logarithm of money. By this approach the utility for the above lottery would be defined as the log difference between the gambler's wealth before and after the event:

$$\mathbb{E}[U] = \sum_{n=1}^{\infty} \frac{\log(w + 2^{n-1} - c) - \log(w)}{2^n} < \infty \tag{3}$$

where c is the cost of entry.

Unfortunately expected utility theory, like all other proposed methods, does not entirely solve the St. Petersburg Paradox. For instance, if we alter the lottery such that instead of paying out 2^{n-1} Ducats after a run of n heads we now payout 10 Ducats to the power 2^n we find the expected utility is no longer finite and the paradox returns. An alternative argument, and in fact the only argument which completely 'solves' the problem, points out that such a lottery could never exist in the real-world: no casino could ever afford to offer the high payouts possible no matter how remote those possibilities are, and no human could ever flip a coin indefinitely. Taking this pragmatic approach, the St. Petersburg paradox becomes non-paradoxical and finds no fault in classical decision theory [7].

Although the St. Petersburg paradox does not prove the inadequacy of expected reward in a world limited in time and finances, it does provide us with some important results. To understand this let us return to our first example in which an agent must decide between a certain payoff and

[1]Gold coin

a higher uncertain payoff. Using expected utility theory we can evaluate the expected utilities of both lotteries to see how a gambler's wealth would affect our choice:

$$\mathbb{E}[U_1] = \frac{(\log(w+10) - \log(w))}{1} \quad (4)$$

$$\mathbb{E}[U_2] = \frac{(\log(w+1000) - \log(w))}{80} \quad (5)$$

where U_1 is the utility gained when taking £10 with certainty and U_2 is the utility gained when taking £1000 with probability 1/80. Evaluating $\mathbb{E}[U_1 - U_2]$ we find the following results:

$$\mathbb{E}[U_1 - U_2] < 0 \quad \text{if} \quad w \lesssim £20 \quad (6)$$

$$\mathbb{E}[U_1 - U_2] > 0 \quad \text{if} \quad w \gtrsim £20 \quad (7)$$

A person with wealth less than ∼£20 will favour taking £10 with certainty, whilst a person with wealth greater than ∼£20 will favour the gamble. This seems a reasonable result as we would expect a relatively poor person to gain much more from a small payout than a wealthy person, such that they would rather take with certainty what they may already regard as a significant amount of money over a gamble. In some ways this approach has incorporated risk-aversion into the decision process.

The point we wish to make is not that expected reward is necessarily wrong, but that an agent's particular utility must be carefully evaluated when making decisions. Expected reward alone does not include the possibility that an agent is risk averse, or that one agent's utility may differ from another seemingly identical agent. Additionally, expected utility of any form does not necessarily represent rewards that are actually possible to receive, especially in one-shot games. These points are of particular concern if the agent in question is human.

3. BAYESIAN DECISION MAKING

We begin this section by formulating a simple 2-armed bandit problem to act as a testbed for the comparison of various action selection strategies.

3.1 Definition of Bandit

There are two parts to the problem that require definition: the action space $\mathcal{A}$ and the hidden reward function R^* (here the suffix $*$ denotes the true rather than the inferred or expected reward function). As this is a two-armed formulation the action space contains only two actions, $\mathcal{A} = \{a_1, a_2\}$, which correspond to pulling lever 1 or 2.

We assume here the reward to be discrete and to take either the value 1 or 0, which is equivalent to the bandit paying out either the top prize or nothing. This results in two binary reward distributions from which the agent playing the bandit can sample. Parameter λ^* is defined for each of these distributions as the probability that the sampled reward will equal 1:

$$\lambda_1^* = Pr(r_t = 1 | a_{1,t}) \quad (8)$$

$$\lambda_2^* = Pr(r_t = 1 | a_{2,t}) \quad (9)$$

where $a_{k,t}$ indicates action a_k taken at time t.

At each time step the agent chooses which of the two binary reward distributions to sample from (which lever to pull) with the intention of maximising future cumulative reward. As only one action can be taken at a time, information will therefore only be received about that action. The agent accumulates samples of reward and failure for each action over time. Here $s_{k,t}$ and $f_{k,t}$ respectively represent the number of successes and failures at time t the agent has received after taking action a_k. Success is defined as the received reward equalling 1, whilst failure is defined as the received reward equalling 0. $\mathcal{D}_t = \{s_{1,t}, f_{1,t}, s_{2,t}, f_{2,t}\}$ represents the full history of samples for both actions at time t. (Unless needed for clarity we will from now on omit the t index to simplify nomenclature.) The agent uses $\mathcal{D}$ in order to decide which actions to take in the future. The decision method used is a free choice of the agent, and we discuss several different approaches here.

3.2 Maximum Likelihood

To begin the discussion on how to make decisions in the bandit problem we first look at a very simple approach using maximum likelihood estimates.

As stated earlier, the agent is only able to make decisions using the past history of samples $\mathcal{D}$. If we had an estimate for the value of λ_1^* and λ_2^*, we could make a decision which would maximise the expected future reward at the next step given these estimates. If $\hat{\lambda_k}$ is the estimate of the true reward probability for action k, the expected reward when taking action k is:

$$\mathbb{E}[r|\hat{\lambda_k}, a_k] = \int r\, p(r|a_k)\, \mathrm{d}r = \hat{\lambda_k} \quad (10)$$

In order to maximise the expected reward under this estimate, the agent simply needs to choose the action a_k with the largest value of $\hat{\lambda_k}$.

3.2.1 Estimating λ

The maximum likelihood (ML) estimate of $\hat{\lambda_k}$ is the ratio of the number of successes to the total number of samples for action k:

$$\hat{\lambda_k} = \frac{s_k}{s_k + f_k} \quad (11)$$

Maximum likelihood in the context of the multi-armed bandit (and indeed in many other cases) has particular problems which are discussed in the next subsection.

3.2.2 Problems with Maximum Likelihood

To understand why maximum likelihood is particularly problematic, we first consider situations in which at least one action has not been sampled from. If action a_k has not been sampled from, $\hat{\lambda_k}$ and therefore the expected reward from taking that action $\mathbb{E}[r|\hat{\lambda_k}, a_k]$, are undefined. If the expected reward is undefined for any action how is a decision made? One option is to artificially define a value for the estimate when it is otherwise undefined. In this particular problem a value of $\hat{\lambda_k} = 0.5$ might make sense indicating no preference in the likelihood of success or failure, however this can only be considered ad-hoc as it does not naturally arise. For larger more complex systems, any artificial choice of value will undoubtedly have a large effect on the decisions being made and therefore should be avoided.

The second problem is that uncertainty is not incorporated into maximum likelihood decision making. There should be greater uncertainty in our beliefs regarding actions which have been sampled more infrequently. This is because as more samples are collected for a particular action, the more

accurate the maximum likelihood estimate becomes. As an example consider two actions, a_1 and a_2: a_1 has been sampled once and one reward has been gained, a_2 has been sampled ten times and ten rewards have been gained. According to equation (11), $\hat{\lambda_1}$ and $\hat{\lambda_2}$ are both equal to 1. We know that the true values λ_1^* and λ_2^* cannot take the value 0, so our belief in what values the estimates can take is bounded $0 < \hat{\lambda_1}, \hat{\lambda_2} \leqslant 1$, however it is *more likely* that the true value λ_2^* is closer to 1 than λ_1^* given the samples we have taken. We should be more uncertain about our values for $\hat{\lambda_1}$ than $\hat{\lambda_2}$, and this uncertainty should affect the decision made. The maximum likelihood framework in no way represents this uncertainty.

A solution implemented in many maximum likelihood frameworks and applicable to both the above problems is to perform a *burn-in* period, referred to as ϵ-first strategies in the literature [3]. During a burn-in period the agent will attempt to sample from all the reward distributions to avoid any undefined estimates, and ideally to a point such that the uncertainty in the estimates is marginal. The major drawback with this method is its reliance on ad-hoc variables such as the length of burn-in or minimum estimate uncertainty. The agent may also not gain rewards it would otherwise have gained during this period if it was using a less naive decision method.

3.3 Bayesian Parameter Estimation

We now discuss a method in which we infer distributions over the parameters we are estimating using a Bayesian framework. Decisions are made using the expected values of the parameters which are calculated by integrating out the uncertainty in the distributions. This method avoids the problems associated with maximum likelihood methods by providing a means for the natural inclusion of prior knowledge and incorporation of uncertainty in the parameter estimations and subsequent decisions.

3.3.1 Using Bayes' Theorem

We now wish to generate probability distributions representing our belief in the value of the true parameters λ_1^* and λ_2^*. These distributions will contain all the information we have available to us and any prior knowledge we would like to include. Bayes theorem is used explicitly to form the distributions. As there are two independent parameters being estimated, we infer two independent *posterior* distributions. If there are cross couplings between the levers then a joint posterior distribution is required. The *likelihood* for each distribution is calculated using the history of past samples from the associated action and reward distribution.

To avoid confusion in the following discussion we present Bayesian inference for a general action a_k. It should be assumed that inference is performed for both actions using only the data relevant to that action, namely $s_{k,t}$ and $f_{k,t}$ (the number of successes and failures from $s_{k,t}+f_{k,t}$ samples of action a_k at time t) and any prior knowledge.

With regards to notation we use λ_k rather than λ_k^* (the true expected reward) to indicate our belief in λ_k^*. As λ_k^* is a singular value the distribution over it is a delta function at λ_k^*.

3.4 Bayesian Inference

3.4.1 Likelihood

Each sample can be thought of as a random variable drawn from a Bernoulli trial with unknown probability of success $\lambda \in [0, 1]$, where success is still defined as the sampled reward equalling 1. If this random variable is sampled n times with s successes the resulting probability distribution is binomial:

$$p(s|\lambda = x) = \binom{n}{s} x^s (1-x)^{n-s} \tag{12}$$

This equation is the likelihood for action a_k, $p(s_k|\lambda_k = x)$, if the number of rewards equalling 1 (s_k) equals s, and the total number of samples of action a_k's reward distribution ($s_k + f_k = n_k$) equals n.

3.4.2 Prior

The *conjugate prior* to the binomial distribution is the beta distribution which is a continuous probability distribution defined on the interval $[0, 1]$, specified by only two parameters α and β. The beta distribution is particularly useful as it possesses analytic solutions to (for example) expectation and variance. The probability density function of the beta distribution is:

$$p(\lambda = x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{13}$$

B, the beta function, acts as a normalisation constant and is defined as:

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{14}$$

where Γ is the gamma function:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} \mathrm{d}t \tag{15}$$

so we can see the Binomial is a special case of the Beta where α and β are integers.

3.4.3 Initial Prior

Before any samples are taken from action a_k we have no information pertaining to the value of λ_k^* and should therefore have equal preference across all values of $\lambda_{k,0}$ at time $t = 0$:

$$p(\lambda_{k,0} = x) = 1 \qquad \text{for all } x \tag{16}$$

This is a uniform prior representing maximal ignorance. To achieve this we set $\alpha_{k,0}$ and $\beta_{k,0}$ to 1, which can be verified by substituting these values into equation (13).

3.4.4 Posterior

Using Bayes' rule, equations (12) and (13), and evaluating the *evidence* as a normalising constant, the posterior can be calculated:

$$\begin{aligned} p(\lambda = x|\mathcal{D}) &= \frac{p(\mathcal{D}|\lambda = x) \times p(\lambda = x)}{\int_0^1 \left(p(\mathcal{D}|\lambda = x) \times p(\lambda = x)\right) \mathrm{d}x} \\ &= \frac{x^{s+\alpha-1}(1-x)^{n-s+\beta-1}}{B(s+\alpha, n-s+\beta)} \end{aligned} \tag{17}$$

where $\mathcal{D}$ represents the total information gained from sampling. As expected the posterior is another beta distribution with updated parameters. Using equation (17) the posterior parameters for action a_k at time t are:

$$\alpha_{k,t} = \alpha_{k,0} + s_{k,t} \tag{18}$$

$$\beta_{k,t} = \beta_{k,0} + n_{k,t} - s_{k,t} \tag{19}$$

where $\alpha_{k,0}$ and $\beta_{k,0}$ are the prior parameters defined earlier.

In non-sequential Bayesian inference equation (17) is evaluated completely at each time step. This means that for action a_k at time t, the posterior distribution is calculated using the initial values of α_k and β_k indicating a uniform prior, and s_k and f_k equal to the *total* number of successes and failures up to that point. It is also possible to form sequential Bayesian inference in which the last *posterior* distribution to be inferred becomes the *prior* at the next round of inference. We do not present this approach here, however the result is the same as for non-sequential inference.

3.4.5 Expected λ and reward

The distributions generated in this section represent the belief we have in the value of the underlying reward probabilities. We can use these to calculate the expected reward when taking each of the actions, conditioned on the past history of samples and the uniform prior, by integrating the uncertainty out in the distributions. The expected reward for taking action a_k is:

$$\begin{aligned}\mathbb{E}[r_t|a_{k,t},\mathcal{D}_k] &= \int r\, p(r|a_k,\mathcal{D}_k)\,\mathrm{d}r \\ &= \iint r\, p(r|\lambda_k,a_k)\, p(\lambda_k|\mathcal{D}_k)\,\mathrm{d}\lambda_k\,\mathrm{d}r \\ &= \int \lambda_k\, p(\lambda_k|\mathcal{D}_k)\,\mathrm{d}\lambda_k \\ &= \mathbb{E}[\lambda_k|\mathcal{D}_k] \end{aligned} \tag{20}$$

The expected reward for taking action a_k is simply equal to the expected value of λ_k. The agent must choose the action a_k with the largest expected λ_k to maximise expected reward at the next step. The agent will be indifferent between two actions with the same expected reward.

Marginalising the uncertain parameter λ_k is the only way to calculate expected reward. This does *not* mean the uncertainty in the parameter is completely discarded, the Bayesian framework naturally encodes this uncertainty in the expected value. Consider an action a_k which has never been sampled; the expected value of λ_k is equal to the expected value of our uniform prior, indicating complete ignorance. As more samples are taken we gain information about the underlying reward distributions and as such our expected value of λ_k moves away from our prior expectation. As the number of samples increases greatly the expected value tends toward the maximum likelihood estimate. In this situation the Bayesian paradigm can be seen to moderate our estimation between the uniform prior and maximum likelihood models based on the certainty of the λ_k distribution.

3.5 Alternative Bayesian Decisions

It is possible for two distributions with differing uncertainty to have the same expected value. That is to say that the certainty with which we make estimations can be different, even if the expected values are the same. Standard decision theory, which is equivalent to the method discussed in the last section, finds no reason to distinguish between expected rewards if they evaluate as the same. Clearly it *is* possible to distinguish between distributions with equal expected values if the uncertainties differ (even if this distinction later proves to be uninformative). We conjecture that we should in fact differentiate between two such distributions when making decisions. As we believe the Bayesian parameter estimation to be correct, the question we now ask is not 'how should we manage uncertainty?' but rather 'what measure should we be maximising through decisions?'.

What follows is a discussion of how both λ distributions can be combined onto a single distribution over a latent variable. When the single latent variable is marginalised non-standard decisions are made.

3.5.1 Understanding confidence

In this problem we compare two probabilities of rewards. As the reward is either 1 or 0 we will always favour actions which have a greater chance of reward. If we wanted the greatest chance of reward at the next step and we knew the underlying reward probabilities λ_1^* and λ_2^* we would *always* pick the action a_k with the highest probability λ_k^*. If the true reward probabilities were equal we would be indifferent between the two actions.

Let us temporarily assume that we can define some confidence or probability that one action is better than another. Instead of always choosing the 'best' action (the action with the highest estimate of reward probability) we choose actions with probability based on the confidence probability. That is to say we explicitly define a probability $P(a_k)$ that we choose action a_k. If later we do not want to choose actions stochastically we can still take the action with the highest selection probability (the Greedy action) with certainty; in the results section we take this approach (MOD-EX GREEDY).

We can justify defining a confidence probability by looking at what values we are comparing. λ_1^* and λ_2^* are both probabilities of the same result occurring (reward equalling 1) from different actions. As the result for each is stochastic a situation could arise such that if both actions were taken at the same time, an action with a lower value of λ could provide a reward when an action with a higher value of λ does not. The confidence probability provides a measure of how certain we are one action will not provide a reward when another one would have. For example, consider two reward probabilities: $\lambda_1 = 1$ and $\lambda_2 = 0$; in this case we will be completely certain that there will never be a situation when action a_2 will give a reward and action a_1 will not. If the values are closer, for example $\lambda_1 = 0.8$ and $\lambda_2 = 0.7$ we become much less sure that there will not be a situation when action a_2 gives a reward and a_1 does not.

3.5.2 Selection Probability

At the next step we want to receive a reward, so we condition our selection probability on us actually receiving this reward. The selection probabilities are also conditioned on the past history of samples $\mathcal{D}$, as such the selection probability for action a_k is $P(a_k|r=1,\mathcal{D})$. We also want to couple actions so that we always choose one action:

$$P(a_1|r=1,\mathcal{D}) + P(a_2|r=1,\mathcal{D}) = 1 \tag{21}$$

In order to use the Bayesian framework we also need to define the prior probabilities of choosing actions. The prior probability of choosing action a_k will be $P(a_k)$. As before:

$$P(a_1) + P(a_2) = 1 \tag{22}$$

As we should have no prior preference for choosing one action over another both prior probabilities should be equal, and both are hence assigned as 0.5.

The total probability of receiving a reward at the next step, conditioned on the past history of samples and an ac-

tion being taken, is now:

$$P(r=1|a,\mathcal{D}) = P(r=1|a_1,\mathcal{D})P(a_1)+P(r=1|a_2,\mathcal{D})P(a_2) \tag{23}$$

which is the normalising term in the Bayesian formulation. Using Bayes', the selection probability of action a_k is the proportion of this probability mass that a_k is responsible for. For action a_1 this is:

$$P(a_1|r=1,\mathcal{D}) = \frac{P(r=1|a_1,\mathcal{D})P(a_1)}{P(r=1|a_1,\mathcal{D})P(a_1) + P(r=1|a_2,\mathcal{D})P(a_2)} \tag{24}$$

or more generally:

$$P(a_1|r=1,\mathcal{D}) = \frac{P(r=1|a_1,\mathcal{D})P(a_1)}{\sum_k P(r=1|a_k,\mathcal{D})P(a_k)} \tag{25}$$

As both prior probabilities are equal this simplifies to:

$$P(a_1|r=1,\mathcal{D}) = \frac{P(r=1|a_1,\mathcal{D})}{\sum_k P(r=1|a_k,\mathcal{D})} \tag{26}$$

If the true parameter values λ_1^* and λ_2^* are known equation (26) becomes:

$$\begin{aligned} P(a_1|r=1,\lambda_1^*,\lambda_2^*) &= \frac{P(r=1|a_1,\lambda_1^*)}{\sum_k P(r=1|a_k,\lambda_k^*)} \\ &= \frac{\lambda_1^*}{\lambda_1^*+\lambda_2^*} \end{aligned} \tag{27}$$

Note that the probability of reward when action a_k is taken is only dependent on parameter λ_k^*.

In reality parameters λ_1^* and λ_2^* are not known, the agent only has belief in their value gained from $\mathcal{D}$, which is represented by the posterior distributions over λ_1 and λ_2. Therefore there is a distribution over the selection probability which may be integrated out via a Bayesian marginal integral:

$$\begin{aligned} P(a_1|r=1,D) &= \iint p(a_1|r=1,\lambda_1,\lambda_2)p(\lambda_1|D)p(\lambda_2|D)\,\mathrm{d}\lambda_1\,\mathrm{d}\lambda_2 \\ &= \iint \frac{\lambda_1}{\lambda_1+\lambda_2}\,p(\lambda_1|D)p(\lambda_2|D)\,\mathrm{d}\lambda_1\,\mathrm{d}\lambda_2 \end{aligned} \tag{28}$$

where $p(\lambda_1|D)$ and $p(\lambda_2|D)$ are the posterior distributions generated in section 3.3.

There is no simple analytical solution to (28), hence our solution is to approximate the integral by drawing samples from λ_1' and λ_2', where:

$$\lambda_1' \sim Beta(\alpha_1,\beta_1) \tag{29}$$

$$\lambda_2' \sim Beta(\alpha_2,\beta_2) \tag{30}$$

and then for each pair of samples calculate the ratio:

$$\frac{\lambda_1'}{\lambda_1'+\lambda_2'} \tag{31}$$

and evaluate the mean of all the sampled ratios to find (28):

$$P(a_1|r=1,D) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{\lambda_{1,i}'}{\lambda_{1,i}'+\lambda_{2,i}'} \tag{32}$$

where $\lambda_{k,i}'$ is the ith sample from λ_k', and N is the total number of samples.

3.5.3 The Log Odds

In order to see how the uncertainty in the λ posteriors affects (28) it useful to expand the above formulation. We explicitly define a latent variable using a log odds ratio, and marginalise this through the logistic function. Uncertainty in the latent variable can be seen to moderate the posterior action belief.

We define a latent variable b as the log odds of the action selection probability before marginalisation:

$$\begin{aligned} b &= \log\left(\frac{p(a_1|r=1,\lambda_1,\lambda_2)}{1-p(a_1|r=1,\lambda_1,\lambda_2)}\right) \\ &= \log\left(\frac{\lambda_1}{\lambda_2}\right) \end{aligned} \tag{33}$$

The logistic (or softmax) function is the inverse of the log odds:

$$p(a_1|r=1,b) = \frac{1}{1+\exp(-b)} \tag{34}$$

Using the above definitions it is simple to show that equation (28) is equivalent to:

$$P(a_1|r=1,D) = \int p(a_1|r=1,b)p(b|D)\mathrm{db} \tag{35}$$

As there is no analytic solution (other than approximations) to the integral, we sample λ_1' and λ_2' and define:

$$b' = \log\left(\frac{\lambda_1'}{\lambda_2'}\right), \tag{36}$$

hence

$$P(a_1|r=1,D) \simeq \frac{1}{N}\sum_{i=1}^{N}\frac{1}{1+\exp(-b_i')}. \tag{37}$$

The integration can be seen as the moderation of the logistic function by the latent variable. High uncertainty in either of the λ posteriors will result in high uncertainty in the latent variable. Higher uncertainty in the latent variable will result in a posterior probability through the logistic integration which is closer to the prior of 0.5, even if the expectation of b' is the same.

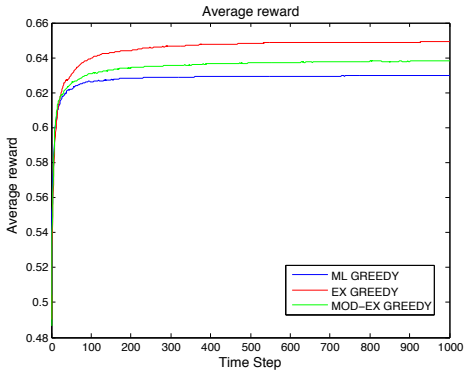
4. RESULTS

To assess the relative performance of different bandit strategies we compare them on a suite of test problems. The test suite consists of 2000 2-armed Bernoulli bandit tasks. For each task, the probability of reward when taking action a_k, $P(r_t|a_k,t) = \lambda_k^*$, is drawn from a uniform probability distribution between 0 and 1:

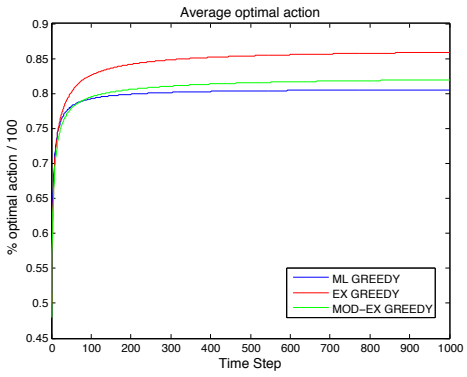
$$\lambda_k^* \sim \mathcal{U}(0,1) \tag{38}$$

Once drawn λ_k remains constant throughout the duration of each task representing a static system scenario. During each task all methods are tested using the same set of reward probabilities for consistency. The horizon of each task is $\mathcal{H} = 1000$ to allow enough time for the relative behaviours of each algorithm to emerge.

We analyse the performance of each algorithm using three measures: proportion optimal action taken, average reward received and information gained. All three measures are evaluated and plotted against time to see how performance varies as actions are taken. The optimal action is defined

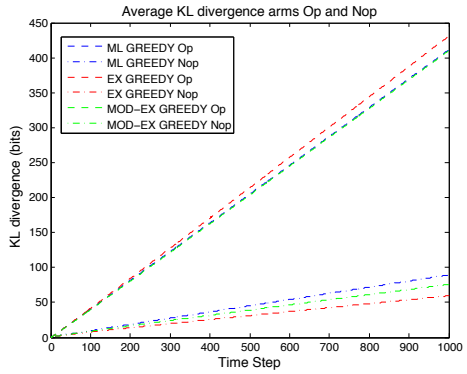


(a) Average reward gained

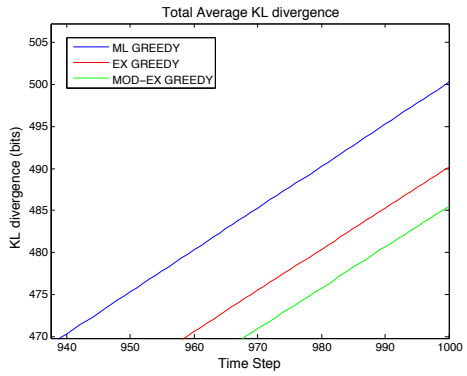


(b) % optimal action was selected

Figure 1: Average performance of Maximum Likelihood, Expectation and Moderated Expectation Greedy Maximisation on the 2-Armed Bernoulli Bandit Test Suite.



(a) Optimal Action (Op) and Non-Optimal Action (Nop) plotted separately



(b) Total information gained (for clarity only the last 60 time steps are presented)

Figure 2: Average information gained (KL-divergence) using Maximum Likelihood, Expectation and Moderated Expectation Greedy Maximisation on the 2-Armed Bernoulli Bandit Test Suite.

as the action with the highest true probability of reward. Information gained is calculated as the symmetric *Kullback-Leibler divergence* between a uniform prior, indicating complete uncertainty, and the inferred Beta distributions at each time step. As information is gained the inferred Beta distributions move away from the uniform prior and KL divergence increases. We calculate the information gained for the optimal action and the suboptimal action separately.

The three methods tested are those described in Section 3 which are: *Maximum Likelihood* maximisation (ML GREEDY - Section 3.2), *Expectation* maximisation (EX GREEDY - Section 3.4) and *Moderated Expectation* maximisation (MOD-EX GREEDY - Section 3.5). All three methods use a greedy action selection strategy. As the maximum likelihood method does not infer the posterior Beta distributions in its decision process, we use the successes and failures at each time step to establish what the posterior distributions would be if such a method was used, and use these distributions to calculate the KL divergence.

Figure 1 shows the average reward gained, the proportion the optimal action was taken and the average regret for each algorithm over the full test suite. EX GREEDY performed the best on all three measures, followed by MOD-EX GREEDY and finally ML GREEDY. This clearly illustrates the importance of correctly encoding uncertainty using a Bayesian formulation; Both Bayesian methods outperform the Maximum Likelihood method.

MOD-EX GREEDY moderates the expectation based action evaluation according to the uncertainty of both posterior distributions. This results in more information being gained about non-optimal actions. Figure 2 shows the information gained over the test suite and further illustrates this point. Interestingly, we can see from this figure that MOD-EX GREEDY gains more information about non-optimal actions than EX GREEDY but gains less total information than any other method. This figure also illustrates how poor ML GREEDY is at utilising information. ML GREEDY gains significantly more total information than either other method but still gains the least reward. Once again this is because uncertainty is not incorporated into the decision process.

Although on initial inspection it appears that EX-GREEDY outperforms MOD-EX GREEDY it is important to remember this is a static reward test suite with a small discrete action space. EX-GREEDY should perform well on such a test suite as it minimises the sampling of non-optimal levers whilst still encoding uncertainty, however if the action space became larger or the system was dynamic this algorithm would not necessarily perform comparatively as well. In contrast this test has shown that MOD-EX GREEDY utilises information well by gaining the smallest amount of total in-

formation but still performing better than ML GREEDY.

5. MULTI-AGENT SYSTEMS

Sequential decision making in multi-agent systems furnishes us with the additional problem that an agent's reward function could be a function of the actions of other agents acting in the system as well as its own. Each agent must be aware that there is a feedback between sensing, decision making and acting in such a system; any action made by agent A will likely affect the reward agent B receives, thus altering agent B's future actions and in turn changing the future reward agent A receives. Uncertainty can exist in the structure of an agent's own reward function as well as the structure of other agents' reward functions making it difficult to predict what actions others are likely to take. Even with perfect knowledge of all reward functions in a system, each agent can operate under its own set of unknown strategies adding yet another layer of uncertainty to the problem; this is especially the case in systems containing human agents.

The importance of incorporating uncertainty when dealing with multiple agents is demonstrated in [9]. In that paper a two-agent problem is defined with the following combined payoff matrix:

$$R = \begin{pmatrix} (1,1) & (0,r) \\ (r,0) & (10,10) \end{pmatrix} \tag{39}$$

where r is a factor determining the degree of risk dominance of the action pair 1/1 set to range from $r = -1, -2 \cdots -25$. It is clearly best for both agents to choose action 2 and receive a payoff of 10, however it was shown in that paper that many traditional learning approaches which did not incorporate uncertainty resulted in agents becoming convinced that playing action 2 was bad due to the penalty for playing 1 against 2 being high. Using a moderated fictitious play algorithm, similar in approach to the method presented in Section 3.5 and in which agents evaluated actions over a logistic function, they found convergence to the strategy with payoff $(10, 10)$ was more likely.

We have presented our work using the multi-armed bandit platform as it is a simple way to illustrate decision making under uncertainty. The methods discussed are equally applicable to systems with further levels of uncertainty; it is also simple to extend the multi-armed bandit itself such that an individual's success in making choices is dependant on the choices others make. How an agent should combine uncertainty over strategies as well as reward functions does however require future research.

6. CONCLUSION

The exploration vs. exploitation trade-off has been studied extensively, particularly in the field of *Reinforcement Learning*, as it could be applied to almost any system in which an individual seeks to gain some form of utility under uncertainty. Many methods, not just those implemented on the Multi-Armed Bandit, focus on a binary decision between exploitation and exploration.

As Bayesians we fundamentally disagree that an individual should be forced to make a decision between exploiting knowledge and exploring the system. This is not because we believe an agent does not need to reduce uncertainty. We believe that if an individual correctly incorporates the uncertainty it has about the system using a Bayesian formulation, it will gain all the information it needs to achieve maximum utility whilst always behaving in an exploitative manner. Certainly our results reinforce this idea, demonstrating how algorithms using Maximum Likelihood parameter estimation show decreased performance when compared to Bayesian methods.

Unfortunately formulating the full Bayesian behaviour can not only be mathematically complicated, but frequently intractable. This is one of the reasons many heuristic methods have enjoyed a great deal of attention and analysis. The most popular of these methods are based on ϵ-greedy action selection policies, first described in [14]. In this approach the agent chooses its greedy action with frequency $(1 - \epsilon)$, otherwise choosing an action at random. The ϵ parameter is ad-hoc and thus solutions of this form do not generalise well, although their performance is often hard to beat [11]. Ultimately we have to be aware of the tractability of our solutions whilst attempting to maintain a Bayesian approach.

7. REFERENCES

[1] D. Bernoulli. Exposition of a new theory on the measurement of risk. *Econometrica: Journal of the Econometric Society*, pages 23–36, 1954.

[2] N. Bernoulli. Correspondence of Nicolas Bernoulli concerning the St. Petersburg Game. Letter to Pierre Raymond de Montmart, 1713.

[3] E. Even-Dar, S. Mannor, and Y. Mansour. PAC bounds for multi-armed bandit and Markov decision processes. *Lecture notes in computer science*, pages 255–270, 2002.

[4] I. Hacking. Strange expectations. *Philosophy of Science*, pages 562–567, 1980.

[5] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and computation*, 108:212–212, 1994.

[6] W. Macready and D. Wolpert. Bandit problems and the exploration/exploitation tradeoff. *IEEE Transactions on evolutionary computation*, 2(1):2–22, 1998.

[7] R. Martin. The St. Petersburg Paradox. *Stanford Encyclopedia of Philosophy*, 2008.

[8] M. Osborne. *An introduction to game theory*. Oxford University Press New York, NY, 2004.

[9] I. Rezek, D. Leslie, S. Reece, S. Roberts, A. Rogers, R. Dash, and N. Jennings. On similarities between inference in game theory and machine learning. *Journal of Artificial Intelligence Research*, 33(1):259–283, 2008.

[10] H. Robbins. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc*, 58(5):527–535, 1952.

[11] R. Sutton and A. Barto. *Introduction to reinforcement learning*. MIT Press Cambridge, MA, USA, 1998.

[12] W. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, pages 285–294, 1933.

[13] W. Thompson. On the theory of apportionment. *American Journal of Mathematics*, 57(2):450–456, 1935.

[14] C. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

Exploiting Structure To Efficiently Solve Loosely Coupled Stochastic Games

Hala Mostafa
Computer Science Department
University of Massachusetts, Amherst
hmostafa@cs.umass.edu

Victor Lesser
Computer Science Department
University of Massachusetts, Amherst
lesser@cs.umass.edu

ABSTRACT

This paper is concerned with sequential decision making by self-interested agents when their decision processes are largely independent. This situation can be formulated as a stochastic game which would traditionally be represented in extensive form (a single game tree), a representation that fails to exploit the loose coupling in the game. We propose a new representation for 2-agent loosely coupled stochastic games that allows exploiting the sparsity and structure of agent interactions while still being able to capture a general stochastic game. We provide analytical and experimental results to show the representational and computational savings we obtain compared to extensive form in settings with different degrees of coupling. Our second contribution is a compact formulation of our problem as a Multi-Agent Influence Diagram, a first step towards the goal of solving problems with more than two agents. Finally, we investigate the challenges that need to be resolved to meet this goal.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Coherence and coordination

General Terms

Performance, Economics

Keywords

Game theory, Structured game representations

1. INTRODUCTION

In this section, we introduce the notion of loosely coupled stochastic games, provide an example to motivate it and provide a general characterization of this class of games. We also introduce the running example that will be used throughout this chapter.

1.1 Loosely coupled stochastic games

A *stochastic game* describes a situation where agents are self-interested and interact over a number of stages. Each stage begins with the game at some state. Agents take actions simultaneously and, in general, receive rewards based on the actions taken by all agents and the particular state the game was in. The game then probabilistically transitions to a new state based on the previous state and joint actions. The agents are therefore very tightly coupled; each action of each agent affects the rewards and next games of all others.

In contrast to this tight coupling, we introduce *loosely coupled stochastic games* where the agents are largely independent. Each agent has its local state and its decisions only affect its own reward and next state. In our games, an agent generally does not know about the other agent's states and actions and, typically, does not care. What ties the decision processes of the two agents, however, is that there is some interaction between the agents; some actions of an agent affect the others. As the number of interactions increases, we move from completely independent decision processes to the tightly coupled stochastic games discussed in the literature.

Loose interactions arise in many situations. Consider a set of cleaning robots owned by different companies which, between them, manage the cleanliness of a building. Each robot is responsible for a set of halls, but corridors are joint responsibility and the robots can get extra reward if they correctly coordinate their cleaning of this shared space. Other interactions stem from sharing the waste bins and potentially getting into each other's way in shared areas like corridors and elevators. Non-situated agents can also have loose interactions. Consider a set of self-intersted computational servers that offer their computational resources for a fee. Each server has its set of incoming computational tasks, resources, and fee policy. However, some tasks require more resources than is available to any single server, in which case the server receiving the task can propose to some of the others to complete the task for a fee. In addition to affecting each other's rewards, the servers can affect each other's state. Collaborating on certain tasks can result in a server gaining more experience on these tasks, causing it to transition to states where future tasks of this kind are executed more quickly.

Traditionally, the above situations would be represented in extensive form, which requires specifying an agent's rewards and next state for each of its actions *and* the other agents' actions. Clearly, this representations is overly verbose, since in most cases, the rewards and new states are independent of the other agents. Ignoring this fact results in game trees that are much larger than they need to be. Besides being representationally inefficient, a single game tree obscures the structured interaction among agents, making it hard to exploit to efficiently find a Nash equilibrium.

1.2 Characteristics of loosely coupled games

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

The above examples share these general characteristics:

- Each agent has its local states and actions (e.g. robots have different locations and grabbing actions). Most of an agent's action outcomes and rewards are independent of the other agent.
- Agents are generally unaware of each other's states and actions, unless some form of communication is specified.
- Few interactions among agents, compared to the number of actions they can take. They are also structured, meaning that an action can affect the other agent in a specific way. In a *reward* (resp. *transition*) *interaction*, an action of one agent affects the reward (resp. outcome probabilities) of certain actions of the others.
- Interactions are not only between actions taken at the same time. An action can be affected by something that happened in the past (e.g. a robot's dumping a large object in a waste bin will affect the outcome of using the bin at any later point). The fact that the affecting action happened is not necessarily encoded in an agent's state.

Self-interested Mars rovers

To ground future discussions, we use the following simple example that captures the above characteristics.

We use a self-interested variant of the Mars Rovers scenario [1]. Consider a situation where two countries send autonomous rovers to collect data from the surface of Mars. Each rover has a list of samples that it wants to gather. The rovers need to accomplish their missions within a limited time, so they cannot gather all the samples on their list. A rover's set of actions is the set of sites it has not visited yet, and each action has two outcomes, fast and slow, probabilistically chosen by nature. An agent's decision problem is therefore which sites to gather data from and in what order so as to maximize its expected reward. Even though the rovers are generally unaware of each other's actions and whereabouts, their decision processes are not totally independent. We define a *shared site* to be a site where one agent's visit can affect the reward or the outcome probability of the other agent when/if it visits that same site. For example, if both rovers want to gather data from the same site, they may help each other, thus getting a higher reward for the site, or they may get in each other's way, thus reducing one or both rover's reward.

One solution concept for the above situation is the Nash equilibrium; a pair of policies such that no agent is motivated to deviate from its policy. At an equilibrium, i.e., each agent's policy is a best response to the other's.

2. REPRESENTING AND SOLVING 2-PLAYER LOOSELY COUPLED GAMES

2.1 Existing representation: Extensive form

A stochastic game can be represented as an extensive form game (EFG), which is a tree capturing the order in which agents take actions, what they know when they take each action, and the probabilistic nature of actions. An EFG is a tuple $< I, V, E, P, H, u, p >$ where:

- I is the set of n players
- (V, E) is a finite directed tree with nodes V and edges E and Z is the set of terminal nodes
- $Player : V \setminus Z \rightarrow I$ determines which player moves at each decision node.
- $H = \{H_0, ..., H_n\}$ is the set of information sets, one for each player. Each H_i is a partition of $Player_i$.
- $A_i(h)$: the set of actions available at information set h
- $u : Z \rightarrow \mathbb{R}$ is the utility function defined over the set of terminal nodes. For $x \in Z$, $u_i(x)$ is the payoff to player i if the game ends at node x
- p is the transition probability of chance moves

In a game with imperfect information, an agent does not know exactly the state of the other agent (and thus the game played by the agents at any particular stage), but does have a probability distribution over it. In such games, an information set can contain more than one node, which the agent cannot tell apart. A policy should therefore make the same decision across all nodes belonging to the same information set. It is a mapping from information sets to probability distributions over actions.

2.2 Proposed representation: Self-Interested EDI-CR

In loosely coupled games, the rule is that agents' actions are independent and the exceptions are the reward and transition interactions among them. We would therefore like to represent the decision processes of the two agents separately and enumerate the (relatively few) interactions. This is exactly what our model Event-Driven Interactions with Complex Rewards (EDI-CR) does. In previous work [14], we used EDI-CR to capture structured interactions among cooperative agents. For self-interested agents, we slightly modify the original definition of EDI-CR to allow agents to have different reward functions. A 2-agent EDI-CR instance is therefore a tuple $< S_{1,2}, A_{1,2}, P_{1,2}, R_{1,2}, \rho, \tau, T >$ where:

- S_i, A_i, $R_i : S_i \times A_i \rightarrow \mathbb{R}$, $P_i : S_i \times A_i \times S_i \rightarrow [0, 1]$ are the local state space, action space, reward function and transition function of agent i. These elements specify the separate decision processes of the agents.
- $\rho = \{< (s^k_{k_1}, a^k_{k_1}), ..., (s^k_{k_2}, a^k_{k_2}), r^k >_{k=1..|\rho|}\}$ specifies reward interactions. The k^{th} entry specifies r^k, the additional reward/penalty obtained when each agent does its specified action in its specified state at any point during its execution.
- $\tau = \{< (s^k_{k_1}, a^k_{k_1}), (s^k_{k_2}, a^k_{k_2}), p^k >_{k=1..|\tau|}\}$ specifies transition interactions. The k^{th} entry specifies the new transition probability p^k of the state-action pair of agent k_2 when agent k_1 does its specified state-action pair before the affected agent makes its transition.
- T is the time horizon for the problem.

The individual states, actions and rewards describe the dynamics of each agent's decision process, while ρ and τ capture the interactions among them. We stress that the model can be used for general stochastic games; it would have entries in ρ and τ for every combination of actions and states to mirror the fact that all actions affect all agents.

Representing a game as an EFG is more compact than EDI-CR when modeling tightly coupled games. With 2 agents, each having 2 states and 2 actions, the joint representation has 32 entries (16 in each of the reward and

transition functions). If all actions participate in determining the game's next state and individual rewards, EDI-CR has 4 entries per local transition and reward function, in addition to 16 reward and 16 transition interactions, clearly less compact than EFG. At the other extreme, if the agents do not interact, then instead of having a *single* tree in which rewards and transitions depend on the *joint* states and actions of agents, EDI-CR has *separate* trees with transition and reward functions defined over *local* states and actions. In this case, EDI-CR only has entries in the local functions, for a total of 8 entries compared to EFG's 32. Between these two extremes, the extent to which EDI-CR's representation is more intuitive depends on the number of interactions (e.g., for 3 interactions, EDI-CR would have 8+3=11 entries, compared to EFG's 32).

2.3 Solution Method

In this section, we review an existing formulation of a stochastic game as a bilinear program (BLP) and discuss how this formulation is affected by having nearly separable decision processes as opposed to a single game tree. The original formulation of extensive form games was derived by Petrik and Zilberstein [15] and is as follows:

$$\begin{array}{ll} \text{Maximize} & x^\mathsf{T} r_1 + x^\mathsf{T}(C_1 + C_2)y + y^\mathsf{T} r_2 - \lambda_1^\mathsf{T} b_1 - \lambda_2^\mathsf{T} b_2 \\ \text{subject to} & A_1 x = b_1 \quad A_2 y = b_2 \\ & r_1 + C_1 y - A_1^\mathsf{T}\lambda_1 \le 0 \\ & r_2 + x^\mathsf{T} C_2 - A_2^\mathsf{T}\lambda_2 \le 0 \quad x, y \ge 0 \end{array}$$

We use the sequence form [10] to represent the agents' policies. x and y are vectors of realization weights of agent i and j's action sequences, respectively. r_1 (resp. r_2) is a vector representing the individual rewards of agent i (resp. j); those rewards that do not depend on what the other agent does. Each entry in r_i contains the expected reward of a sequence of player i. C_1 and C_2 are matrices with a row for each sequence of i and a column for each sequence of j. These matrices represent rewards of i and j whose attainment depends on what *both* agents did. A_1, A_2, b_1 and b_2 form constraints that guarantee that a solution indeed represents the realization weights of a legal policy; i.e. probabilities of actions at each state add up to 1 (for more details on the constraints over a policy in sequence form, see [10]). λ_1 and λ_2 are the variables of each agent's dual optimization problem. Their presence in the objective function reflects our interest not in a solution that maximizes the sum of rewards, but in one that is an equilibrium.

When dealing with loosely coupled games, there can potentially be many sequences of one agent's actions that neither affect, nor are affected by, the other agent. These sequence will have their own local rewards, but will have zero entries in the agent's C matrix. Using this observation, we can have much fewer rows and columns in C_1 and C_2 if we exclude such sequences. Denoting by $\bar{x}$ and $\bar{y}$ those elements of x and y that affect, or are affected by, actions of the other agent, we get the following program

$$\begin{array}{ll} \text{Maximize} & x^\mathsf{T} r_1 + \bar{x}^\mathsf{T}(C_1 + C_2)\bar{y} + y^\mathsf{T} r_2 - \lambda_1^\mathsf{T} b_1 - \lambda_2^\mathsf{T} b_2 \\ \text{subject to} & A_1 x = b_1 \quad A_2 y = b_2 \\ & r_1 + [C_1\bar{y}; \bar{0}] - A_1^\mathsf{T}\lambda_1 \le 0 \\ & r_2 + [\bar{x}^\mathsf{T} C_2; \bar{0}] - A_2^\mathsf{T}\lambda_2 \le 0 \quad x, y \ge 0 \end{array}$$

where $[v; \bar{0}]$ is vector v padded with enough zeros to make it of the desired length. The details of calculating C_1 and C_2 are similar to those of calculating the team's reward matrix C in the formulation of the cooperative case [14]. The resulting bilinear program is solved using an existing algorithm [15] to find a Nash equilibrium.

3. COMMUNICATION SCHEMES IN EDI-CR AND EXTENSIVE FORM

In this section, we investigate how the degree of coupling affects the relative savings of using EFG and EDI-CR. One simple way to change the degree of coupling is by introducing different amounts of communication among agents[1]. The more the agents communicate, the more they affect each other; sending a message affects what the receiver observes.

Analytical and experimental setup

We present three communication schemes; no, mandatory and optional communication. For each, we analyze the effect on the size of an instance when represented using EDI-CR, EFG and the MAID which will be discussed in the next section. For EFG and EDI-CR, we measure size as the number of states in the joint game tree and in each agent's decision process, respectively. For MAIDs, we look at the total size of the CPDs of all nodes. We express these quantities in terms of A actions, O outcomes per action, T time steps, k reward interactions and m transition interactions, with $k + m \le A$. The variables k and m allow us to investigate how the number of interactions and their nature affect the size of a representation. We stress that our analysis is not specific to the Mars rover example. It applies to any loosely coupled game that fits the characterization we give in Section 1.2. To simplify the analysis, we assume that an action takes one time unit and that actions can repeat.

We also look at the effect of communication on the time to find the first Nash equilibrium[2]. EFGs are solved using the `logit` solver in the game theoretic package Gambit [13] and reported as "Gambit" (we report results of `logit` because it performed better than `lcp`). EDI-CR is solved as a bilinear program reported as "BLP". We time out a solver after 30 minutes and report "N/A".

We present experimental results from 8 instances of the Mars rovers domain with $T \in [6, 8]$ and number of actions is 5 or 6 (unlike the analytical analysis, an action here can take more than one time unit). To avoid generating very large games that would not fit in memory regardless of the representation, we can specify restrictions over actions by having earliest start times before which they cannot proceed. We obtained results from a larger set of data which showed the same trend as in the results we report, so we omit them.

3.1 No communication

We first look at the case where communication is not allowed. An agent makes decisions based only on its local state, which keeps track of what actions have been done so far and the outcomes obtained for them. With EFG, each stage consists of actions and outcomes for both agents. The number of nodes is therefore $\mathcal{O}(A^{2T}O^{2T})$.

[1]Communication is a special kind of transition interaction; sending a message makes the recipient transition to a state where the message is observed, thereby affecting its transition probability. As such, communication can be handled by our solution method.

[2]Because it is hard to compare solution qualities in selfish settings, we are concerned with finding *any* equilibrium

Table 1: Size and performance comparison for the no-communication case (times in seconds)

EFG infosets	EFG size	EDI-CR size	% reduction	Gambit time	BLP time
49	1301	132	89.85%	9	2
68	1618	140	91.35%	18	9
100	3195	216	93.24%	63	2.4
151	7987	303	96.21%	306	2.5
173	11.2k	348	96.89%	1080	3
296	25.1k	610	97.57%	N/A	2.5
333	44.4k	695	98.43%	N/A	2.6
841	473k	2079	99.56%	N/A	8

Table 2: Size and performance comparison for the mandatory communication case (times in seconds)

EFG infosets	EFG size	EDI-CR size	% reduction	Gambit time	BLP time
82	1301	1107	14.91%	21	2.7
83	1618	377	76.70%	29	6
135	3195	600	81.22%	120	6.5
204	7987	1481	81.46%	460	3
201	11.2k	2600	76.80%	900	4
574	25.1k	3475	86.17%	N/A	5
630	44.4k	3000	93.24%	N/A	14.3
1438	473k	7823	98.35%	N/A	5.6

In EDI-CR, each stage in an agent's decision process consists of an action and outcome for this agent only, resulting in $\mathcal{O}(A^T O^T)$ states per agent. Because there are transition interactions, an agent needs to remember the outcome of affected actions, but our state representation remembers all outcomes, so states space size is independent of m.

While theoretically the sizes of EFG and EDI-CR are both exponential in the time horizon T, Table 1 shows that in practice, doubling the exponent results in game trees that are too large to build and/or solve.

3.2 Mandatory communication

We now model situations where communication is inherently part of the setting, rather than a conscious decision on the part of the agents. An agent i doing its part of a reward or transition interaction *involuntarily leaves a trace* that it has done this action. Consequently, the other agent j will see this trace upon doing *its* part of the action. An agent does not suffer a cost for this implicit communication, but cannot avoid it either. To allow an agent to make decisions based on the traces it sees, an agent's state keeps track of a flag denoting whether a trace was seen upon doing each reward or transition interaction.

In EFG, even though the state now stores the actions, outcomes and $k+m$ flags of each agent, the number of states is *not* $\mathcal{O}(A^{2T}O^{2T}2^{2k+2m})$. The reason is that the values of an agent's flags are fully determined by earlier actions of the other agent, so when an agent does an ineteraction, there is no branching over whether it will see a trace there. In fact, there is no more branching in this communication scheme than in the case without communication, and the number of nodes in the EFG tree is still $\mathcal{O}(A^{2T}O^{2T})$.

Even though they are of the same size, the EFG representation of the no communication case and the mandatory case are not the same. To see why, note that because of the additional flags, an agent has more information available to make its decisions when there is communication. This translates into the game tree having more information sets per agent; nodes that were indistinguishable in the absence of communication can now be told apart. Comparing Tables 1 and 2 shows how much the number of information sets in an EFG increased. Since a policy specifies a distribution over actions for each information set, mandatory communication increases the size of the policy space and makes finding a Nash equilibrium more difficult. Table 2 indeed shows that even though the size of EFG did not change, the solution time generally increased.

As for EDI-CR with mandatory communication, there *is* probabilistic branching in an agent's decision process over whether or not it sees a trace upon doing an interaction, since that depends on what the other agent has done. The size of each agent's process is therefore $\mathcal{O}(A^T O^T 2^{k+m})$.

It is interesting to note the effect of mandatory communication on the size gap between EFG and EDI-CR. Compared to no-communication, mandatory communication results in EDI-CR achieving less reduction in size. The increased coupling introduced by communication makes EFG less inadequate, compared to EDI-CR. If we increase the frequency and language of communication, at some point the decision processes will be so tightly coupled that EDI-CR's advantage of representing them separately will be lost.

3.3 Optional communication

We now look at optional communication where an agent can choose whether to leave a trace upon doing its part of an interaction. Even though communication here does have a cost, an agent may still decide to communicate if it knows that communication will cause the other agent to do something beneficial to it. For example, in the Mars rovers scenario, if rover j knows from i's policy that if i visits site s_1, then i will visit s_2, and if s_2 has a much higher value if visited by both rovers, then rover i will choose to leave a trace at s_1 as an inducement for j to go there too.

To represent optional communication in EFG, in addition to actions and outcomes for each agent, there is an action node with two branches (leave trace or not) after every decision to do part of an interaction. A state keeps track of the actions and outcomes of both agents, as well as at most $k+m$ binary communication decisions per agent, for a total of $\mathcal{O}(A^{2T}O^{2T}2^{2(k+m)})$ states. Note that even though in this scheme an agent can potentially have the same information to make its decisions as in the mandatory case, the number of decisions itself is much larger, because of the communication decisions, resulting in a larger number of information sets.

In EDI-CR, again, there are communication decision nodes, in addition to branching over whether an agent will see a marker upon visiting a site. The number of states is $\mathcal{O}(A^T O^T 2^{2(k+m)})$.

Table 3 shows that having communication decisions results in huge EFG trees, making it impossible for Gambit to solve them within a reasonable amount of time. However, the 4^{th} instance shows that solution time and size are not always correlated, which can be explained by the fact that we are searching for the *first* equilibrium we can find, and the time this takes depends on both the size of the problem and the structure of the search space.

Table 3: Size and performance comparison for the optional communication case (times in seconds)

EFG infosets	EFG size	EDI-CR size	% reduction	Gambit time	BLP time
547	21.1k	6213	70.58%	N/A	8.6
136	3777	671	82.23%	N/A	3
190	7511	1093	85.45%	N/A	2.8
602	51.6k	5651	89.06%	N/A	214
589	68.3k	5766	91.57%	N/A	11
2668	295k	13.9k	95.29%	N/A	35
2004	316k	10.2k	96.76%	N/A	32
N/A	2200k	21.8k	99.01%	N/A	195

4. COMPACTNESS OF MAIDS FOR LOOSELY COUPLED GAMES

EFG is a representation that does not exploit any structure in a game. In this section, we investigate a more structured representation and its suitability for loosely coupled games. We give a brief background on Multi-Agent Influence Diagrams (MAIDs) and discuss how we can represent our communication schemes using it.

4.1 Background on MAIDs

Multi-agent influence diagrams (MAIDs) [11, 3] are representations that have their origins in influence diagrams [7]. Like all alternatives to the extensive form representation, MAIDs try to explicitly capture a structural property of a game that would otherwise be obscured in extensive form. In the case of MAIDs, this property is that not all decision variables in a game are inter-dependent.

A MAID defines a directed acyclic graph in which nodes correspond to random variables of three types. For each agent i, there is a set of 1) decision variables, $\mathcal{D}_i$, whose domains are available actions and are represented as rectangles; 2) chance variables, χ_i, whose values are chosen by nature and are represented as ovals; and 3) utility variables, $\mathcal{U}_i$, which represent the agent's payoffs and are drawn as diamonds. A conditional probability distribution (CPD) specifies the conditional probability of the node's variable given an instantiation of its parents, $P(x|Pa_x)$.

A strategy profile for agent i is a set of decision rules, one for each node in $\mathcal{D}_i$. A decision rule specifies the probability of making a certain decision given values of its parents. To represent *perfect recall* (an agent does not "forget" decisions it made in the past), all earlier decisions and their parents are among the parents of a later decision node.

4.2 No communication MAID

At each stage, each agent i has a decision node D_i, a chance outcome node O_i (e.g., a Boolean representing the slow or fast outcome of visiting a site) and a utility node. To guarantee perfect recall, an agent's decision node depends on all its previous decisions and outcomes. The per-agent utility nodes U_i at each stage represent rewards from individual actions. Addition utility nodes U_i^ρ represent payoffs from reward interactions.

In a naiive representation, O_i and U_i^ρ nodes depend on *all* j's past decision nodes to account for the dependence of i's transitions and the shared rewards on j's actions.

To avoid the blow up in CPDs size that this results in, we introduce *helper nodes* that act as "memory" or storage, allowing us to break some of the dependencies on previous decision nodes and replace them with dependencies on helper nodes at the previous stage only. A helper node is a chance node representing a Boolean variable whose value deterministically depends on that of the corresponding helper node at the previous stage and on the decision node at the current stage. The helper node remembers whether a certain action was done in the past.

For the no-communication scheme, we add, for each agent, a helper node at each stage for each of the $k+m$ interacting actions. The variable of agent i's helper node at stage t indicates whether i did the associated action at or before stage t. The node for agent i's x^{th} action that is involved in a reward (resp. transition) interaction is denoted r_i^x (resp. t_i^x) and is referred to as *reward indicator* (resp. *transition indicator*). An indicator has the value True if it was True at the previous stage or the associated action was taken at the current stage. Figure 1 shows a no-communication Mars rovers instance with $A = 4, k = m = 2, T = 3$ represented as a MAID.

We now calculate the size of the MAID representation of the no-communication case.

- Decision nodes: because of perfect recall, an agent's decision node has as its parents all its own previous decisions and outcomes. The CPD of a decision node at level t has $A^{t-1}O^{t-1}$ instantiations of parents, for each of which it specifies the probability of A values, for a CPD size of $\mathcal{O}(A^T O^T)$ per agent.
- Because it has a decision node and a Boolean as its parents, a transition or reward indicator's CPD is of size $2A$.
- Outcome nodes: if an action is part of a transition interaction, its outcome probability depends on whether the affecting action was done by the other agent. An outcome node therefore depends on the current level decision node and the other agent's m transition indicators from the previous level. The CPD then specifies the probability of AO values for each of $A \times 2^m$ instantiations of its parents.[3]
- Utility nodes: the individual utility nodes U_i (labeled $u_{i/j}^x$ in the figure), specify a reward for each outcome of each action, resulting in A^2O^2 entries for each CPD. For shared reward nodes U_i^ρ (labeled b_x in the figure to denote bonus from the x^{th} interaction), the reward depends on whether each agent has done its part of the interaction as summarized in the last level reward indicators of the x^{th} interaction. Each CPD therefore has 2 Boolean parents, for a total size of $4k$ per agent for all reward interactions.

Even though the size of MAID's decision CPDs is the same order as EDI-CR, the MAID is overall larger because of the CPDs of the other kinds of nodes.

Problems with MAID representation

The above mapping highlights some problems MAIDs have in representing loosely coupled games. First, as can be seen in the figure, the structure in our loosely coupled games (the independence of most actions' rewards and transitions) is obscured because a decision node does not branch over

[3]Note that the outcome node needs the value of all m transition indicators because we do not know which of the m affected actions, if any, this agent will take.

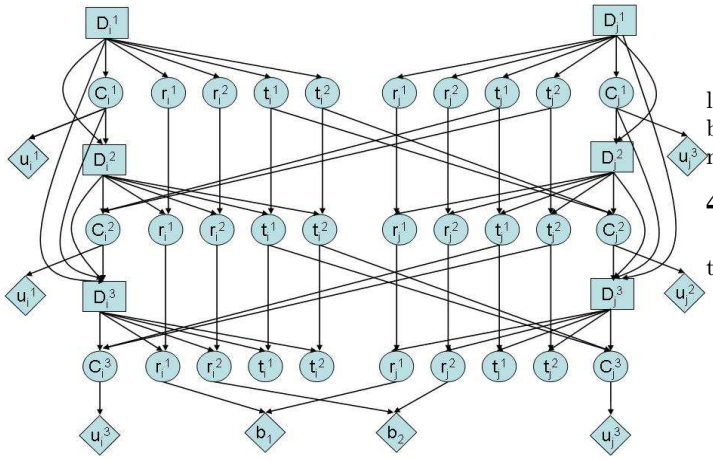


Figure 1: MAID representation of a no-communication Mars rovers instance

the possible decisions, so we cannot isolate a single action and represent its dependence on another. Second, MAIDs do not naturally capture dependencies that are temporally non-localized, forcing us to resort to constructs that "remember" actions done in the past and allow them to affect future actions without having the latter depend on all previous decisions. A MAID representation is essentially stateless, and trying to capture a game in which agents have local state that is affected by previous actions and affects the choice of future actions is problematic.

We also note that for simplicity, we assumed that actions can repeat. MAIDs are not good at representing domains where the set of available actions is context-sensitive, which is what would be needed. To disallow repeated actions, we would need, for every pair of an agent's decision nodes, a utility node that imposes a large penalty if the actions taken at these nodes are the same.

4.3 Mandatory communication MAID

Representing mandatory communication requires making the following modifications to the no-communication MAID.

- At each stage, we need an indicator for whether an agent sent a message (left a trace) when it did its part of an interaction. This information is already contained in the transition and reward indicators.
- For each agent, each stage, and each of the $k+m$ interactions, we add a helper node called RCV that indicates whether a message regarding the corresponding interaction was received. A RCV is True if it was True at the previous level, or if the agent did its part of an interaction that was also done by the other agent. The parents of a RCV at level t are the RCV from level $t-1$, the other agent's indicator for the interaction from level $t-1$ and this agent's decision node at level t, for a size of $4A$.
- We change each decision node at each level t so that in addition to all previous decision and outcome nodes, its parents also include RCV nodes from level $t-1$, allowing an agent to base decision on what messages it received. The size of a decision node at level t is $A^t O^t 2^{k+m}$, resulting total size of decision nodes being $\mathcal{O}(T 2^{k+m} A^T O^T)$.

Mandatory communication exacerbates MAID's main problems because 1) receiving messages depends on actions done by the other agent arbitrarily long ago and 2) decision nodes now have even more information feeding into them.

4.4 Optional communication MAID

To represent optional communication, we need to make the following changes to the mandatory communication MAID:

- We add a *SND* node after each decision node. *SND* is a Boolean decision node representing the choice of whether to leave a trace or not. To represent perfect recall, a *SND* has all previous decision, *SND* and RCV nodes as parents. The total size of *SND* CPDs is therefore $\mathcal{O}((4AO(k+m))^{T+1})$.
- Decision nodes have the same set of parents as *SND* nodes, so their size is comparable.
- A RCV node at level t now depends on the RCV at level $t-1$, the decision node at level t, and the other agent's *SND* node at level $t-1$.

Experimentally comparing MAID to EFG and EDI-CR was not possible because of the simplifying assumptions we made (actions have unit durations and can repeat) in order to get a reasonable MAID representation, assumptions that place MAIDs and other representations on unequal footings. Without these assumptions, we would get even bigger MAIDs, but even with them, the MAIDs were too large to solve.

5. RELATED WORK: STRUCTURED GAME REPRESENTATIONS

MAID: As introduced earlier, Multi-Agent Influence Diagrams [11, 3] (MAID) is a representation for sequential games that is suitable for capturing independence among variables, rather than among actions of different agents. Initial work on MAIDs used this representation to decompose a game into interacting fragments, and provided an algorithm that finds equilibria for these smaller games in a way that is guaranteed to produce a global equilibrium for the entire game [11]. Later work addresses the issue that most realistic games are not decomposable in this way. Blum et. al. address this by exploiting finer-grained structure in MAIDs to improve the efficiency of a certain family of algorithms called continuation algorithm [3]. In Section 6, we discuss the possibility of exploiting the structure in loosely coupled games to improve continuation algorithm.

TAGG: Temporal action graph games (TAGG) [8] is a graphical representation of imperfect-information extensive form games that can be much more compact than MAIDs; a TAGG can be exponentially more compact than a *naiive* MAID representation. However, a carefully constructed MAID is only polynomial in the size of the TAGG. TAGGs represent games with anonymity (a player's payoffs depend on how many players took a certain action, rather than exactly who they are) and context-specific utility independence. TAGGs are an extension of AGGs to represent games taking place over multiple stages. Because TAGGs are specifically geared towards games with anonymity, we cannot use them to represent our games.

Succinct EFG: For some games, the game trees expressed in extensive form are too large to be stored in memory explicitly. To overcome this, Dudik and Gordon propose an implicit representation called *succinct EFG* [5]. A representation is succinct if it has enough information to support certain queries that make it possible to simulate play in a game through sampling. As such, MAIDs are themselves examples of succinct EFGs. However, MAIDs cannot represent context-specific independence (e.g. allowing different decision nodes to have different available actions), a drawback addressed by succinct EFG. For loosely coupled games, however, succinct EFG does not capture the large degree of independence that agents have, and still represents their interaction in a single game tree.

I-DIDs [4] model multi-agent interactions extending over time. An agent maintains and updates models of other agents as part of its belief update. We believe this explicit modeling and the maintenance of the models can get expensive. For loosely coupled games, agents may not need to construct and maintain such accurate models of each other.

Other representations Unlike the dearth of representations for sequential games, a number of representations have been proposed for 1-stage games with special structure. For example, graphical games [9], Game nets (G-nets) [12] and action-graph games [2] address games whose special structure is the locality of interactions where an agent only interacts with a subset of other agents whose size is small relative to the total number of agents.

The work on poker (e.g. [6]) tries to exploit structure in sequential games to scale to larger games and provides automatic abstractions that produce much smaller games whose solutions can be converted to solutions of the original games. The problem is that with the assumptions they make, it is not clear that these techniques are of general use.

6. CHALLENGES OF HAVING MORE THAN TWO AGENTS

Even though we pointed out some weaknesses in using MAIDs to represent loosely coupled games, MAIDs still have one important advantage over our bilinear program formulation: using MAIDs and algorithms developed for MAIDs, we can represent and solve games with more than two agents. In this section, we briefly overview the state-of-the art algorithm for solving MAIDs [3], investigate whether we can exploit the structure in loosely coupled games to make this algorithm more efficient, and highlight the challenges involved in doing so.

6.1 Continuation method for MAIDs

Continuation methods work by perturbing a problem into a simpler problem that can be easily solved. The solution is then traced to that of the original problem by decreasing the magnitude of the perturbation. When the perturbation is zero, we have a solution to the original problem.

This approach was used by Blum et. al to solve MAIDs [3]. A large pertubation is applied to the rewards in the form of a bonus vector that rewards an agent for its actions regardless of anything else that happens in the game. If large enough, these bonuses dominate the original game rewards and simply determine what the optimal strategies are.

When applied to MAIDs, tracing the solution of the perturbed problem to that of the original problem requires finding the Jacobian (the first order derivative) of the vector function $V^G(\sigma)$. Each entry $V_a^G(\sigma)$ in this function maps the profile σ to the payoffs obtained by the agent playing a for deviating from σ and playing a all the time. Using the sequence form representation, if there is a total of n sequences for all agents, then σ is a profile of length n, $V^G(\sigma)$ is a vector of length n and $\nabla V^G(\sigma)$ is an $n \times n$ matrix.

In an unstructured game, we would need to fill an entry in the Jacobian for each pair of sequences. In a MAID, however, Blum et. al decompose this task into computing a joint marginal distribution for every pair of agents i and j, and every utility node U_i of agent i over Pa_{U_i}, D_i and D_j, where Pa_{U_i} is the set of parents of U_i and D_x is the set of decision nodes of agent x. For node U_i, the calculation is

$$\sum_{Pa_{U_i}, D_i, D_j} \frac{Util(U_i) * P(Pa_{U_i}, D_i, D_j)}{\sigma_i(D_i)\sigma_j(D_j)}$$

where $\sigma_x(D_x)$ is the realization probability of decisions in D_x as dictated by x's part of the profile σ, and $Util_i(U_i)$ is i's utility from U_i under a given assignment of the variables in Pa_{U_i}, D_i and D_j. Note that the above expression is an expectation $E(\frac{Util(U_i)}{\sigma_i(D_i)\sigma_j(D_j)})$ taken over all values of the variables in $Pa_{U_i} \cup D_i \cup D_j$.

Instead of doing naiive inference on the induced Bayesian Network of the MAID[4] , Blum et. al use the clique tree algorithm to compute and cache factors over pairs and triplets of cliques which are later used to calculate the desired marginals. So the joint probability $P(Pa_{U_i}, D_i, D_j)$ would be obtained from a triple factor over the union of variables in the 3 cliques containing these 3 sets of variables. In what follows, we denote the clique containing a set of variables V by $Q(V)$.

6.2 Computational challenges

We tried to use Blum et. al's implementation of their continuation method for MAIDs [3] to solve 2-agent instances of Mars rovers[5]. We faced the following computational challenges. The first is typical of most implementations of numerical algorithms, while the second is a more inherent concern.

Sensitivity to the initial random seed: The continuation method starts with certain random parameters, which, because tracing the path is not 100% exact, can affect whether a run will find an equilibrium. For our examples, we found it difficult to hit upon a random seed that results in a solution.

Large size of Jacobian: In our instances, a profile can easily have 500 elements (with $A = 4$ and $T = 3$, σ has 512 elements), so the sheer size of the matrix is very large. Manipulating the matrix, and even constructing it, quickly becomes infeasible.

We next discuss how the second problem can be addressed.

6.3 Exploiting loose coupling

The construction of the Jacbobian as described in [3] does not use the fact that not all of an agent's variables affect another agent's reward. Our idea for making the calculation of the Jacobian more efficient is to exploit the structure in loosely coupled games to come up with reduced versions

[4]The induced BN of a MAID under a profile σ is obtained by replacing decision nodes in the MAID with random variables whose CPDs are dictated by the decision rules in σ

[5]We are very grateful to Prof. Christian Shelton of UC Riverside for the code and related discussions.

of the marginal utilities that abstract away details of one agent that are irrelevant to another agent's reward, leading to smaller factors and a speed up in calculation.

In our loosely coupled game, consider calculating the expectation of i's individual utility node at time 2, which only depends on i's decision at time 2 (d_i^2) and its probabilistic outcome (ch_i^2), so $Pa_{U_i} = \{d_i^2, ch_i^2\}$. $Q(Pa_{U_i})$, however, can potentially include many more variables; decisions of i and their outcomes, as well as reward and transition indicators of j. Similarly, a set D_x contains decision variables of agent x, but the clique $Q(D_x)$ contains these variables in addition to all but the last outcome variable of x. The number of parents of a utility node is therefore much smaller than the number of variables in the union of the 3 concerned cliques.

If we just wanted to calculate $E(Util(U_i))$, we could get rid of all variables except $\{d_i^2, ch_i^2\}$. But because of the terms $\sigma_i(D_i)\sigma_j(D_j)$, we can only get rid of some of these variables. To see how this can be done, we expand the expectation to

$$\sum_{D_i}\sum_{D_j}\sum_{CH_i}\sum_{CH_j}\sum_{T_j}\frac{Util(U_i) * P(D_i, D_j, CH_i, CH_j, T_j)}{\sigma_i(D_i)\sigma_j(D_j)}$$

where CH_x are x's chance outcome variables and T_x are its transition indicators. By pushing terms outward as far as the summations allow, we get

$$\sum_{d_i^2}\sum_{ch_i^2} Util(U_i) \sum_{D_i \setminus d_i^2} \frac{1}{\sigma_i(D_i)} \sum_{D_j} \frac{1}{\sigma_j(D_j)}$$
$$\sum_{CH_i \setminus ch_i^2}\sum_{CH_j}\sum_{T_j} P(D_i, D_j, CH_i, CH_j, T_j)$$

Clearly, the last 3 summations can be eliminated to give

$$\sum_{d_i^2}\sum_{ch_i^2} Util(U_i) \sum_{D_i \setminus d_i^2} \frac{1}{\sigma_i(D_i)} \sum_{D_j} \frac{1}{\sigma_j(D_j)} P(D_i, D_j, ch_i^2)$$

For a utility node representing i's reward from a shared task, the node's parents will be the reward indicator variables at the last level, which indicate whether each agent has done its part of the shared task. Again, our 3 cliques will contain variables that are irrelevant to the expectation, so we marginalize out all variables CH_i, CH_j and all reward indicators except the parents of the utility node.

The question we have not resolved yet is how to calculate these smaller joint distributions. Blum et. al calculate joint distributions by manipulating potentials computed in the calibration step of the clique tree algorithm. But since we want distributions over parts of cliques, we cannot do this.

7. CONCLUSION

In this paper, we addressed a special kind of stochastic games where the agents are largely independent except for a relatively small number of interactions among them. We characterized this kind of games and proposed a representation that separates the agents' decision processes and enumerates their interactions. Using this representation, we can formulate the problem of finding a Nash equilibrium as a bilinear program. We also discussed the suitability of two existing representations (extensive form games and multi-agent influence diagrams) for loosely coupled games. We introduce different kinds of communication as a way of varying the degree of coupling among agents. We investigate how changing this degree affects the compactness of the three representations we study, both analytically and experimentally. Finally, we looked at the potential of exploiting the special structure in our games to make algorithms for multi-agent influence diagrams more efficient, which would open the way to solving games with more than two agents.

One important future direction of our work is investigating the effect of having communication on the quality, in terms of social welfare, of the Nash equilibria we find. In order to do this, we need to be able to find (a bounded approximation of) the socially optimal equilibrium; the one with the highest total reward. We are currently trying to make the objective function of the bilinear program reflect this requirement.

8. REFERENCES

[1] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition independent decentralized markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004.

[2] N. A. R. Bhat and K. Leyton-Brown. Computing nash equilibria of action-graph games. In *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence*. University of Toronto, July 2004.

[3] B. Blum, C. R. Shelton, and D. Koller. A continuation method for nash equilibria in structured games. *Journal of Artificial Intelligence Research*, 2006.

[4] P. Doshi, Y. Zeng, and Q. Chen. Graphical models for online solutions to interactive pomdps. In *AAMAS 2007*, 2007.

[5] M. Dudík and G. Gordon. A sampling-based approach to computing equilibria in succinct extensive-form games. In *Proc. 25th Conference on Uncertainty in Artificial Intelligence*, Montreal, Canada, 2009.

[6] A. Gilpin and T. Sandholm. Finding equilibria in large sequential games of imperfect information. In *ACM Conference On Electronic Commerce*, pages 160–169, 2005.

[7] R. A. Howard and J. E. Matheson. Influence diagrams. *Readings on the Principles and Applications of Decision Analysis*, pages 721–762, 1984.

[8] A. Jiang, K. Leyton-Brown, and A. Pfeffer. Temporal action-graph games: A new representation for dynamic games. In *Proc. 25th Conference on Uncertainty in Artificial Intelligence*, pages 268–276, Montreal, Canada, 2009.

[9] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence*, CA, USA, 2001.

[10] D. Koller, N. Megiddo, and B. von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14, 1996.

[11] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1027–1034, 2001.

[12] P. La Mura. Game networks. In *Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence*, pages 335–343, San Francisco, CA, 2000.

[13] R. D. McKelvey, A. M. McLennan, and T. Turocy. Gambit: Software tools for game theory, 2007.

[14] H. Mostafa and V. Lesser. Offline planning for communication by exploiting structured interactions in decentralized MDPs. In *2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 193–200, Italy, 2009.

[15] M. Petrik and S. Zilberstein. A bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research*, 35:235–274, 2009.

Multi-robot planning under uncertainty with communication: a case study

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

Matthijs T.J. Spaan
Institute for Systems and Robotics
Instituto Superior Técnico
Lisbon, Portugal
mtjspaan@isr.ist.utl.pt

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

ABSTRACT

Although Dec-POMDP techniques can be useful to modeling a wide range of problems, their practical application is limited by the inherent computational complexity of the algorithms currently available to solve such models. The application of these techniques is typically restricted to theoretical examples. This work studies the application of a particular type of Dec-POMDP (a multiagent POMDP) model to solve a simple task in a realistically simulated robotic soccer environment. The multiagent POMDP exploits the availability of a communication channel, as is often the case in multi-robot systems. The necessary constraints on the problem are identified, and the steps taken to accomplish efficient cooperative behavior within real robot middleware are explained. Finally, results are presented for the proposed task that highlight further possible improvements.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Performance, Experimentation

Keywords

Multiagent Decision Processes. Planning Under Uncertainty. Cooperative Robotics.

1. INTRODUCTION

Planning and decision-making is one of the central topics of robotics and operations research. In most realistic environments, it is necessary to take into account uncertainty in the agent's actions and/or observations. A Markov Decision Process (MDP) is a widely known and well-studied mathematical framework to model problems where the outcome of an agent's actions is probabilistic, but knowledge of the agent's state is assumed [6]. For this type of problems, several algorithms exist that provide optimal and approximate solutions to MDPs in reasonable time.

When the agent's knowledge is insufficient to directly determine its state, for example a mobile robot with noisy sensors, then the uncertainty of its observations must also be considered. Such problems are where the Partially Observable Markov Decision Process (POMDP) framework finds its domain of application. Planning under uncertainty for a single agent using POMDPs has been the object of active research for the last decade [7, 8]. While solving a POMDP optimally in its general case is an difficult problem, various algorithms exist to compute approximate solutions to moderately-sized POMDPs [2, 8, 11] that may find some application in real-world scenarios.

In certain applications, however, a single agent is not enough to model the full scale of the problem. Such is the case, for example, in cooperative robotics, where multiple agents must work together to achieve a common goal. The Decentralized POMDP framework is a natural extension of the POMDP paradigm for multiple agents. In this type of model, not only do the agents need to consider the uncertainty of their own actions and observations, but also that of their partners, which may (or may not) be aided by the use of communication [12, 13]. Naturally, being more general, this type of models is also harder to solve than their POMDP and MDP counterparts. In fact, optimally solving a general Dec-POMDP is provably intractable [5, 1]. However, approximate solutions can be found [4], even if the computational complexity of the current existing algorithms restricts their application to small-scale problems. This same problem limits their usability in real world scenarios, and so far their application has been restricted to theorical experiments and small simulated examples [1]. Special classes of this model include Multiagent MDPs and POMDPs (MMDPs) and decentralized MDPs (Dec-MDPs) [1].

This work is means as an example of the application of POMDP-related techniques in a small-scale realistic scenario. By doing so, the necessary constraints and possible simplifications to the general model can be identified, and the underlying implementation problems become apparent. Specifically, the problem under study is identified as a multiagent POMDP, a particular form of Dec-POMDPs. The case-study for this work is robotic soccer, a widely known environment for cooperative robotics. Before applying these techniques directly to a real team of soccer robots, which would be impractical, a reasonable approach is to first test the behavior of such robots in realistic simulated environments. This work describes the steps taken from the design of a cooperative robotic task to its implementation in a real robotic middleware using techniques within the Dec-POMDP framework. We establish that, with some reason-

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

able abstractions and simplifications, (Dec-)POMDP techniques can be applied in real problems, and that, within the context of this particular case-study, the obtained results are comparable, and in some aspects advantageous, to other established decision-making frameworks.

2. THE DEC-POMDP MODEL

A Dec-POMDP is a tuple $< n, S, A, \Omega, T, O, R, h >$ where:

- n is the number of agents;
- S is the discrete set of states for the system. The initial state, $s_0 \in S$, is assumed to be unknown to the agents;
- A is the discrete set of joint actions. $A = \times_i A_i$, with A_i the set of actions available to agent i. At each step a joint action $a =< a_1, ..a_n >\in A$ is selected, where $a_i \in A_i$ is the action selected by agent i;
- Ω is the set of joint observations. As with the joint actions, $\Omega = \times_i \Omega_i$, and at each step a joint observation $< o_1, ..., o_n >\in \Omega$ is taken;
- T is the transition function, i.e. for states $s, s' \in S$, and joint action $a \in A$, $T(s', s, a) = P(s'|s, a)$, the probability of making a transition from state s to s' by applying action a;
- O is the observation function. $O(o, a, s') = P(o|a, s')$ specifies the probability of observing o after applying action a and ending up in state s';
- R is the reward structure. $R : S \times A \rightarrow \Re$ specifies the immediate reward granted to the agents as a whole, for taking joint action a in state s;
- h is the horizon of the problem, which represents the number of steps (decisions) that the agents can take.

The framework also allows models for communication between the agents [9]. In this particular context, communication is assumed free (although not instantaneous, refer to section 5). For a more thorough description of Dec-POMDPs and their mathematical properties, see [5].

In the most general case, in order to choose a specific action at each time step, each agent needs to take into account all the possible actions taken by every other agent up until that time step, and all the observations that they might have received. However, in scenarios where the agents are able to communicate freely with each other, the problem becomes simpler, since it is then possible to compute a probability distribution over the set of joint states (a *joint belief*) given only the latest information gathered by the agents. This Markovian signal eliminates the need to consider the complete history of the agents' histories. In this sense, the problem then reduces to a Multiagent POMDP problem. In each step, a joint action for all agents is selected based on this joint belief, which is then updated given the joint observation gathered by the agents.

3. MODELING A ROBOTIC SOCCER TASK

The task proposed in this work is based on a simple situation where two robotic soccer players must cooperate in order to take the ball towards their opponent's goal. During the course of their task, the robots may encounter obstacles that they should be able to avoid, although the position of these obstacles is not known beforehand. One of these players should carry the ball forward, and the other should position itself so that it may receive a pass from its partner, if necessary. The robots may choose to pass the ball in order to avoid imminent obstacles, since it is difficult to avoid obstacles while carrying the ball. The robot that carries the ball at any given time will be referred to in this case study as the "Attacker", and its partner the "Supporter". Whenever a pass occurs, the roles of the robots should switch, meaning that an Attacker becomes a Supporter and vice-versa. The Attacker should then kick the ball to the goal as soon as it detects an opportunity to score. The initial position of the robots and of the ball in their field of play is unknown, and so is their role. They should then determine which robot should carry the ball. The robots possess sensors to detect their own location, the position of the ball, and any surrounding obstacles.

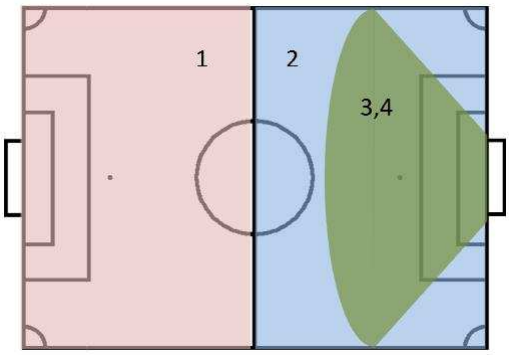


Figure 1: The different sections into which the field of play is divided according to an agent's localization information: 1-Its own half; 2-The opponent's half; 3-Near the goal; 4-In a shooting opportunity. Note that 3 and 4 are coincident with respect to position but vary in their orientation requirements.

3.1 Identifying states, observations, and actions

The first step in the modeling process of this task as a Multiagent POMDP is identifying the states of the overall system. It is assumed that the robot's field of play only contains the agents themselves, the ball, and an unknown number of opponents, and except for these obstacles, their navigation is free inside the field. The state of the robots can then be encoded through their localization information, the position of the ball, and the presence of obstacles. Regarding localization, the field of play is discretized into four different sections, as represented in Figure 1. The agent may be located in its own half-field, in its opponent's half, near the opponent's goal, or in a shooting opportunity, which requires not only for the robot to be near the goal while carrying the ball, but also turned towards it. The robot may also use localization information to sense if its ready to receive a pass from its partner. The information regarding obstacles can be encoded in a binary form, in the sense that the robot is either blocked by obstacles or free to move in its current direction. Finally, the robot is also able to detect whether or not it is in possession of the ball. Note that the robots share their localization information —they require this information in order to be able to follow each other and to be able

to sample their own observations (see Section 5). This is accomplished through explicit communication, as described in the following sections. None of the remaining state variables (information about the ball and obstacles) is shared, since they are not required by each robot's partner. This means that, given this information, the robot is able to estimate (with uncertainty) its own state, but not his partner's.

According to this description, each agent may then be in one of the 13 local states $s_i \in S_i$ described in the diagram in Figure 2. However, these local states are not independent, i.e., the problem specification mandates that one of the robots is an Attacker and the other a Supporter. Therefore, the total number of states is 60, which results from the admissible combinations of local states. It is important to note that this does not affect the partial observability of the agents. They may still receive conflicting observations (for example, that they are both Attackers or Supporters).

Since the complexity of most POMDP algorithms is heavily dependent on the number of observations, it follows that this set should be as reduced as possible. Since the robots must be able to switch roles, this set is necessarily the same for each agent. The set of possible observations is described, for each agent, as follows:

- Own Half —Having the ball in the half-field belonging to the agent's team;
- Opponent's Half —Having the ball in the opposing team's half-field;
- Near Goal —Having the ball near the goal;
- Ready —for an Attacker, this signals that the robot is in a shooting opportunity. For a Supporter, this implies that the robot is able to receive a pass;
- Not Ready —signal received by an Attacker without the ball, or a Supporter which isn't ready to receive a pass.
- Blocked —if the agent is blocked by obstacles.

The preceding signals encapsulate all necessary information about the environment, and result in 36 possible joint observations. The construction of the associated observation model, $O = \langle o^i, o^j \rangle, i = 1, ..., 6, j = 1, ..., 6$, may be further simplified by exploiting observation independence between agents (Section 3.2).

Note that the observation set for each agent does not depend on its specific role as either an Attacker or a Supporter. In some instances (the Ready and Not Ready signals), it is possible to use the same representations implicitly for both cases, since each agent will take its own observation into context through the observation function.

These observations are, in themselves, quite abstract, and each of them depends on possibly more than one source of information. To achieve this level of abstraction in real robots, it is necessary to implement high-level "virtual sensors" that classify the information collected by the robots' physical sensors (and, in the case of localization, information shared by the partner robot) into one of the observations defined above. The observation function can then be constructed in practice by collecting the output of these classifiers while setting the robot in a specific, *a priori* known state. The observation function is then estimated through the collected data.

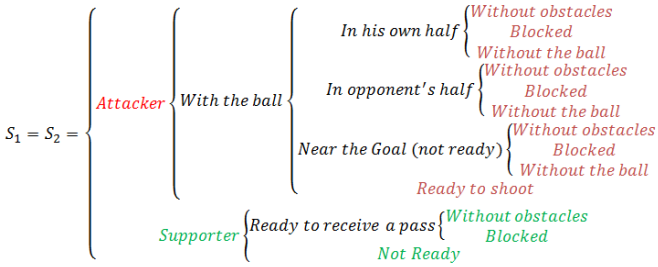


Figure 2: Diagram showing the different possible local states the agent may be in. Note that they are not independent, since for one agent to be the Attacker the other must be the Supporter and vice-versa.

The remaining component of this model that must be described is the set of joint actions, A. As with the observation set, this set is identical for both agents. For this particular task, the following actions are sufficient:

- Dribbling the ball towards the goal;
- Shooting (or kicking) the ball;
- Passing the ball towards the other robot;
- Recovering the ball if it becomes lost by the attacker;
- Following the attacker;
- Finding a position where a pass can be received (finding clearance for the pass).

Logically, the first four actions described in this manner should be performed by the Attacker robot, while the remaining actions should be taken by the Supporter. Therefore, $A = \langle a^i, a^j \rangle, i = 1, ..., 6, j = 1, ..., 6$.

Notice that these actions are defined as high-level behaviors that each robot can assume. Each of these high-level actions is then interpreted by the robot's middleware, and triggers a series of more basic behaviors, that may possess their own local decision-making loops. When the robot decides to dribble towards the goal, for example, these lower-level behaviors ensure that the robot is always turned towards the goal, and supplies the robot with the necessary controls so that it may drive the ball and try to avoid any imminent obstacles. The specific mechanisms through which this is accomplished lie outside of the scope of this work. However, it is important to note that the actions taken at this level impact the transition function of the Multiagent POMDP model. To define such functions rigorously, it is necessary to collect experimental data to such an extent that the transition probabilities estimated through this data approximate the correct values of the transition function (although this may be aided by the use of simulators).

3.2 Exploiting Local Independence

Given the set of joint observations and the set of joint actions, it is then necessary to describe the uncertainty in each of these elements, which is to say that the transition function T and observation function O must be defined. However, in constructing these models, it is advantageous to reduce as much as possible the required information about

the environment. Ideally, if the local state of an agent was not influenced by the actions of the other agent, the model would be completely conditional independent, i.e.:

$$P(s'|s, a_1, a_2) = P(s'|s, a_1)P(s'|s, a_2) \quad (1)$$

This would mean that there is an independent transition function for each agent, T_i, such that

$$T(s', s, a) = T_1(s', s, a_1)T_2(s', s, a_2) \quad (2)$$

for every $s, s' \in S, a = \langle a_1, a_2 \rangle \in A$. From the above description of the action set, it is evident that for some of the actions (namely, passing and shooting), this assumption is not valid, since the application of one of these actions by an agent may induce its partner to switch its role. However, it is valid for all of the remaining actions. The problem may still be further simplified by noting its symmetry. Since there is no characteristic feature to distinguish one agent from the other, their transition functions are identical, $T_1 = T_2$. This means that its only necessary to consider the effects of the 4 possible independent actions for each agent (which is a considerable reduction from the 36 possible joint actions). Also, since only half of the states correspond to a specific agent being Attacker or Supporter, this means that, in matrix form, the transition function for each of the independent actions is block-diagonal (i.e. it is impossible to transition from being an Attacker to a Supporter by applying these actions). For the joint actions that are not conditional independent, their distribution over the possible 60 states must be obtained.

For the observation model, a similar rationale can be taken, but in this case the problem is further simplified by noting the full observation independence in this particular Multiagent POMDP model. At first sight, the passing and kicking actions could be understood to also influence the observations of the respective partner robot, but this is indeed not the case, since the observations have been defined independently for each state, and the actions taken by the partner robot do not influence the ability of each agent to perceive its respective information. The joint actions in this task can then be said to be non-informative, in that they may influence the state of the system, but not the sensors of the agents. In practice, this means that:

$$\begin{aligned} P(o|a, s') &= P(o_1|a, s')P(o_2|a, s') && (3) \\ &= O_1(o_1, a, s')O_2(o_2, a, s'), o_{1,2} \in \Omega_{1,2} && (4) \end{aligned}$$

It should also be noted that $O_1 = O_2$.

So far nothing has been said about the reward structure for this particular robotic task. The agents will choose a course of action (a policy) according to the expected reward obtained in future steps (see Section 4). Although the behavior of the agents can be indirectly influenced by manipulating the reward structure in such a way that the desired actions are promoted, this is undesirable since it would in fact encode the optimal policy in the model itself, and would remove any merit from its solution. The definition of the reward model is simply to assign a high reward for kicking the ball in a shooting opportunity, and to penalize (lightly) every other step taken.

3.3 Comments on the Functionality of the Multiagent POMDP model

Although the transition and observation models for this particular Multiagent POMDP have been found to be relatively simple to define in theory, in order to rigorously obtain these models in practice, it is necessary to estimate the respective probability distributions by collecting large amounts of experimental data. However, due to restrictions in time, this was overlooked in favor of using empirically estimated values. The effect of this decision on the optimality of the resulting joint policy will be noted. The manner in which the environment was discretized into states may also prove some difficulties with respect to the definition of these models. Since the states were defined in a loose topological manner, they are coarse relative to the dimension of the agents, and so the probability of transitioning to a neighbour state depends heavily on the particular configuration of the agent inside a given state. Although this effect can be modelled, to some extent, in a given action's transition distribution, the resulting distribution will be necessarily flat, and little information can be taken from it. A possible way to overcome this problem is to take advantage of the specific sensor information (for localization) of the agent, which contains much more information than what is used by the Multiagent POMDP.

4. OBTAINING AN APPROXIMATELY OPTIMAL POLICY

Although various algorithms allow for the approximate solution of Dec-POMDPs in its most general form, such as JESP [3], Bayesian Game based approaches [4], and Memory Bounded Dynamic Programming [14], these algorithms typically take into account the *history* of each of the agents, i.e. all the past actions taken, as well as the perceived observations. This greatly increases the computational complexity of the problem. For the simpler case of Multiagent POMDPs, more efficient algorithms exist. The Perseus algorithm [2] was chosen to solve the Multiagent POMDP in this work, due to its efficiency in handling moderately-sized POMDPs. Perseus belongs to the family of point-based POMDP solvers, but it is by no means the only one [11, 15]. While it is true that often the algorithm to solve a given POMDP model should be chosen according to the problem's structure, this does not create, in this case, a dependancy on any particular algorithm. In fact, if communications were assumed instantaneous, the model could simply be reduced to a single-agent POMDP and dealt with accordingly by any appropriate solver.

The main objective of any MDP-based model is to maximize the expected reward of the agent(s) obtained for a given number of steps, the problem's horizon. This is described by the optimal *value function* associated with a particular MDP. The optimal action to take at each step is then simply the action that maximizes the value function. In the POMDP case, this function has the useful property of being piecewise-linear convex, i.e. the value of a given belief is a linear combination of a set of vectors:

$$V^*(b) = \max_k \sum_i b_i \alpha_i^k, \quad (5)$$

where b represents the belief (in this case the joint belief), t is the number of remaining decisions to make, and α^k are the vectors that define the linear segments of the value function. Although the value function itself can be efficiently described in this manner, the number of vectors that define the value function k, increases, in the worst case, exponentially in

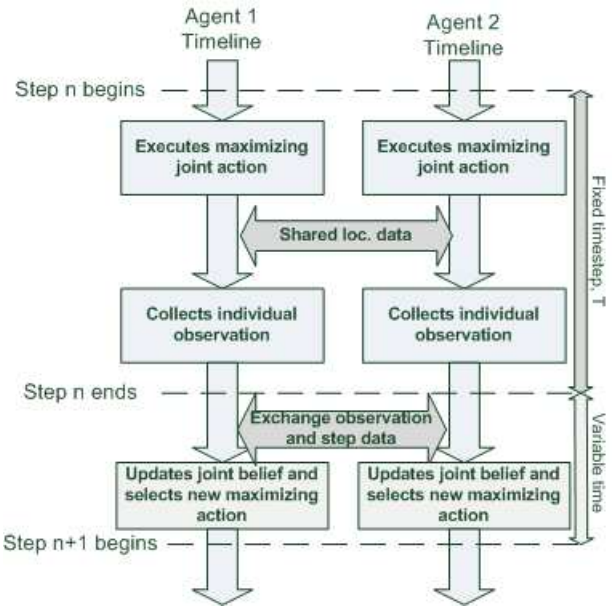


Figure 3: Synchronization timeline with instantaneous communication.

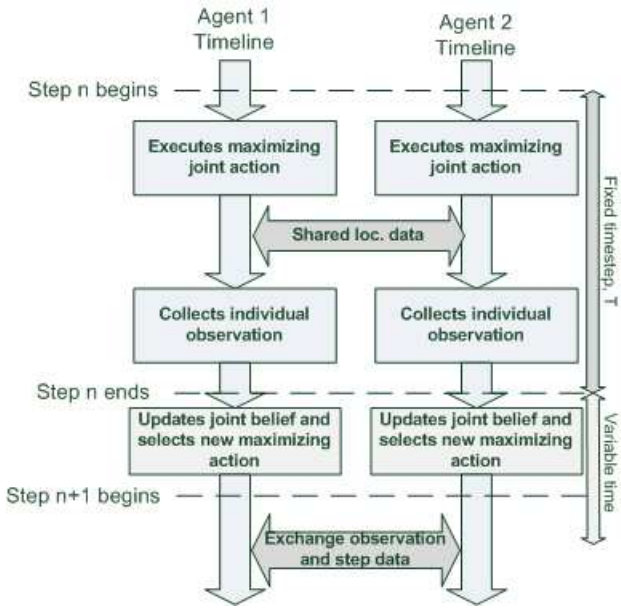


Figure 4: Synchronization timeline with delayed communication of observations.

the number of observations as the horizon of the problem increases, which is problematic when attempting to calculate infinite-horizon solutions. Instead of taking the whole belief space into account, the Perseus algorithm [2] considers only a set of reachable belief points, obtained through "simulated" exploration using the POMDP's transition and observation models. The algorithm then samples belief points from this reachable set at each step and calculates its corresponding α-vector (a reasonably inexpensive step) until the value for all points in the belief set has been improved. The algorithm then continues to iterate until a convergence criteria is met.

5. COMMUNICATION

In order for the problem to be modeled as a multiagent POMDP, it was assumed that the robots could communicate their observations freely to each other. However, the agents' optimal actions are dependent on the joint belief, which depends on each of their partners' observations (their actions are already known since the joint policy is common knowledge). For this information to be coherent, the robots must keep synchronized when carrying out their policies, i.e. they both must execute an action at (approximately) the same time. This way, the agents will calculate the same joint belief at each step.

The synchronization process and necessary explicit communication is performed according to the diagram in Figure 3. Each agent is assumed to perform its own component of the maximizing joint action for a fixed time step T. The agents' observations are only available after the outcome of this action is known. Therefore, after T has passed, the agents sample their own observations and exchange it with their partners'. If one of the agents is delayed, then its partner will wait for this information before proceeding. This step is where synchronization is enforced between both robots. This information is then used to locally calculate the joint belief. After the joint belief is obtained, each agent computes a new maximizing joint action and proceeds to the next step.

In a realistic scenario, it might not be feasible to assume instantaneous communication between the agents. This is often the case in robotic soccer, since the high number of agents quickly saturate the communication medium. In such a setting, it can be advantageous not to wait for synchronization between all involved agents. In this sense, the agent would select an optimal action based on its own local observation, and receive its partners' data throughout the decision step (Figure 4). It is then necessary to investigate the effect of receiving delayed information. Spaan et al. determined that the solution of a Dec-POMDP in such a setting, with up to one step of delay in the communication process, can be acheived through a series of Bayesian games [10]. Using a slightly modified version of the Perseus algorithm, results were obtained for this case.

6. RESULTS

Since the planning and execution phases must be carried out separately in the proposed task, here too they should be made distinct. With respect to planning, the Perseus algorithm performed favorably, and converged in as few as 100 iterations, as can be seen in Figure 5. Such a value function is a good approximation of a stationary solution, i.e., a solution that assumes an infinite horizon. Such a result demonstrates the tractability of the proposed Multiagent POMDP model.

The execution of the task itself was tested in the Webots simulation environment, which allows for realistic physics and control of the robotic agents. Furthermore, the real soccer robots upon which these agents were modeled may be controlled by directly using the same code as in the simulator, increasing its overall realism. The two agents were placed in arbitrary initial positions in the field of play, and the ball was initially placed in the center of the field (which is

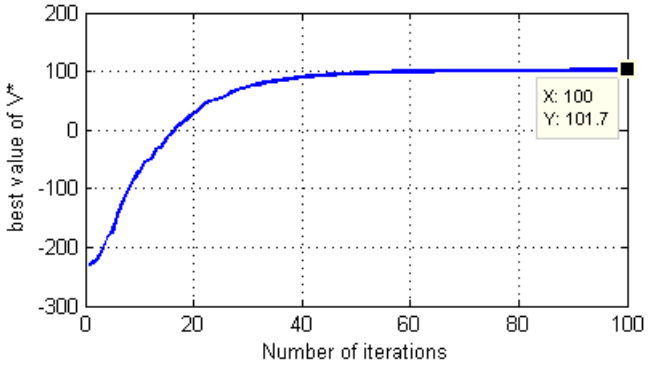


Figure 5: Convergence of the Perseus algorithm for the proposed Multiagent POMDP.

the origin of the world frame for the robots). The time step with which these agents should select an action was set to 3 seconds. The robots possessed an estimated 0.4 probability of performing a transition to a neighbour state when dribbling the ball (again, this was done empirically). The observation model for these robots considered both the possibility of having false positive and false negative detections of obstacles (0.1 probability of failing to detect an obstacle and 0.05 probability of detecting an inexistent obstacle). The localization uncertainty was deliberatily made small (around 0.05) since the information it provides is coarse and may heavily affect the agents' joint belief in the case of an erroneous observation.

Generally, the robots succeeded in their task in an efficient manner. In the instantaneous communication case, the expected reward was approximately ~ 360, with 150 the immediate reward for scoring a goal, -1 for all other actions and a discount factor of 0.95. The discrepancy between the expected reward through simulation and the final expected value as obtained by Perseus is caused by the automatic termination of each episode by the solver after scoring a goal, whereas in the simulator the ball was automaticaly reset to the center of the field and rewards continued to be accumulated.

When introducing a one-step delay in the communications, the robots performed less favorably with $\sim$ 30 expected reward. The comparative results shown in in Figure 6 were obtained by simulating both policies for 500 runs. The fact that this reward is positive, however, demonstrates that even in such a case, the robots are still able to cooperate in order to score goals, albeit notably less efficiently.

Although it is difficult to demonstrate the behaviors carried out by the robots in practice, two different situations are here presented that highlight the correct performance of the desired robotic task. These refer to the policy obtained assuming instantaneous communication.

In Figure 7, the positions of the robots and of the ball that were recorded from the simulator in a typical situation are shown. The supporter robot (here shown initially through a dashed line) maintains a fixed distance to the attacker, until at t_2 the attacker scores a goal. The ball is reset by the simulator back to the center of the field. Since the initial supporter is now the closest robot to the ball, it assumes the role of attacker, and at t_3 their roles have been exchanged

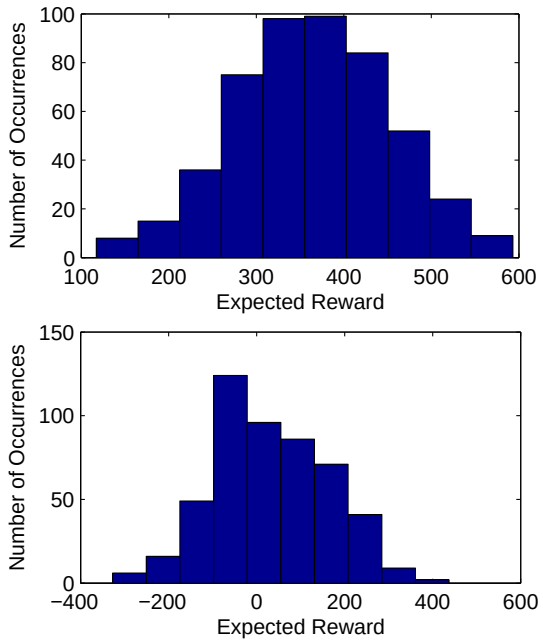


Figure 6: Expected reward collected from simulating the proposed task with instantaneous communication (top) and one-step delay (bottom).

from their initial configurations. The process then repeats itself. Note that in this case, there are no obstacles in the field other than the agents themselves.

A second situation occurs in the presence of obstacles, and is depicted in Figure 8. A barrier of obstacles is placed in front of the initial attacker agent (shown in a dashed line) and at t_2 it selects a pass action since its partner has better clearance to the goal. Their roles then switch, and the agent shown by a filled line then carries the ball until it scores a goal at t_3. Note that the initial attacker still decided to carry the ball for a short amount of time before passing. This is due to the fact that the agent is commited to performing the latest selected joint action until the predefined time-step expires. This presents a problem in dynamic environments, since the robot may not have enough time to select the optimal action when presented with a sudden change in its state. However, the time-step for these decisions cannot be reduced too much, since otherwise the robots do not have enough time to experience the effects of their own actions, i.e., they would most likely remain in the same state when using such coarse topological state definitions as the one proposed in this robotic task.

It is apparent that the resulting policy in both of these cases provides results that are comparable to those obtained with decision-making frameworks that are more common in this context (for example, manually defined policies through finite-state automata), despite the simplifications that were made while modeling the problem. These policies were obtained naturally as the solution of the associated Multiagent POMDP, and, once the observation and transition functions are obtained, only the reward model needs to be adjusted if a different task must be performed (provided that the possible

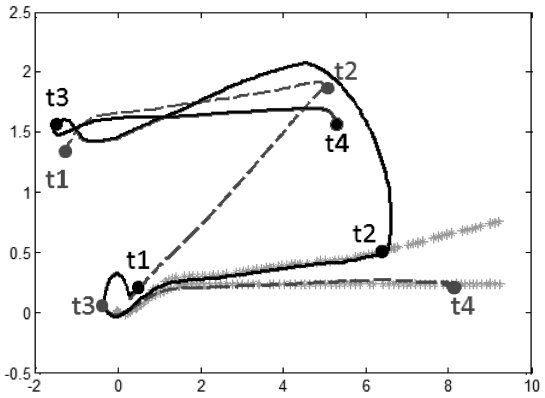


Figure 7: Behavior of the robots after scoring a goal. The positions of the initially attacking and supporting robots are shown by a filled and dashed line, respectively. The position of the ball is marked in gray. The goal is located at (9,0) and is 2m wide.

actions and observations remain the same). This may prove advantageous in environments where the possible tasks are repetitive and predictable.

7. CONCLUSIONS AND FUTURE WORK

This work described the implementation of a robotic task based on a Multiagent POMDP model in a realistic simulated scenario. The steps taken to model, solve and implement the task itself were presented, and results were taken from various simulations. These simulations show that the robots are able to efficiently complete the proposed task with the given model, even if their transition and observation functions are not rigorously defined. In this manner, it was established that it is possible to perform effective decision-making in realistic scenarios by using the Dec-POMDP framework.

The advantage of this approach over most current policy-definition methods in the robotic soccer environment is that it deals with uncertainty in the local information of the robots (regarding the position of obstacles or the ball, for example) in a natural way, without having to manually describe the desired policy for each of the agents. The model itself proved to be solvable in reasonable time. The effect of losing immediate synchronization between the agents was studied. It was shown that the control quality of the agents suffers, although the resulting policy still permits the agents to complete its task.

One problem with this type of approach is related to the time that must elapse between two successive decision steps. If this time value is too large, then the robot loses its ability to react to sudden changes in its state, as when it encounters obstacles along its course, for example. Even if the robot is able to receive an observation consistent with these changes, it may not be enough to alter the belief of the agent sufficiently for it to perform the desired action (the observation model has to be narrow in this sense). If the elapsed time between iterations is too small, then there is not enough time for the robots to transition to another state, and so the transition function would become flat and uninformative.

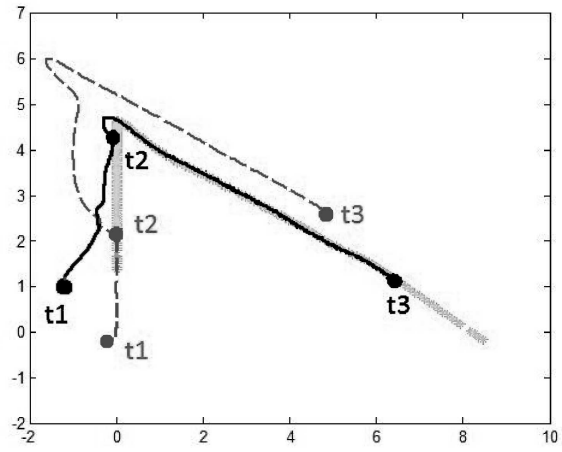


Figure 8: Behavior of the robots when passing to avoid obstacles. The initial attacker, in this case, is shown by the dashed line. At t_2 a pass occurs.

As future work, it would be advantageous to develop a mechanism to condition the observation model on each agent's local information, which is typically much more accurate than the information contained in the coarse topological definition of the system's state.

8. REFERENCES

[1] Claudia V. Goldman, Shlomo Zilberstein, *Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis* , Journal of Articial Intelligence Research, 22: 143-174, 2004.

[2] Matthijs T. J. Spaan and Nikos Vlassis, *Perseus: Randomized Point-based Value Iteration for POMDPs.* Journal of Articial Intelligence Research, 24: 195-220, 2005.

[3] Ranjit Nair, David Pynadath, Makoto Yokoo, Milind Tambe, and Stacy Marsella, *Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings.* In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI), pages 705-711, 2003.

[4] Frans A. Oliehoek, Matthijs T.J. Spaan, Nikos Vlassis, *Optimal and Approximate Q-value Functions for Decentralized POMDPs.* Journal of Artificial Intelligence Research, 32: 289-353, 2008.

[5] D. Bernstein, R. Givan, N. Immerman, S. Zilberstein, *The complexity of decentralized control of Markov decision processes.* Mathematics of Operations Research, 27(4): 819-840, 2002.

[6] R. Bellman, *Dynamic programming.* Princeton University Press, 1957.

[7] H. T. Cheng, *Algorithms for partially observable Markov decision processes.* Ph.D. thesis, University of British Columbia, 1988.

[8] M. L. Littman, A. R. Cassandra, L. P. Kaelbling, *Learning policies for partially observable environments: Scaling up.* In International Conference on Machine Learning, San Francisco, CA, 1995.

[9] P. Xuan, V. Lesser, S. Zilberstein, *Communication decisions in multi-agent cooperation: Model and experiments.* In Proc. of the International Conference on Autonomous Agents, 2001.

[10] Matthijs T. J. Spaan, Frans A. Oliehoek, and Nikos Vlassis, *Multiagent Planning under Uncertainty with Stochastic Communication Delays.* In Proc. of Int. Conf. on Automated Planning and Scheduling, pp. 338-345, 2008.

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

[12] M. Roth, R. Simmons and M. Veloso, *Reasoning about joint beliefs for execution-time communication decisions.* In Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 786-793, 2005.

[13] 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.

[14] S. Seuken and S. Zilberstein, *Memory-bounded dynamic programming for DECPOMDPs.* In Proc. of the International Joint Conference on Artificial Intelligence, pp. 2009-2015, 2007

[15] J. Pineau, G. Gordon and S. Thrun, *Point-based value iteration: An anytime algorithm for POMDPs* In Proc. Int. Joint Conf. on Artificial Intelligence, Acapulco, Mexico, 2003

GaTAC: A Scalable and Realistic Testbed for Multiagent Decision Making

Prashant Doshi and Ekhlas Sonu
Dept. of Computer Science
University of Georgia
Athens, GA,30602, USA
[pdoshi,sonu]@cs.uga.edu

ABSTRACT

Recent algorithmic advances in multiagent sequential decision making have opened up a need to move beyond the traditional toy problems such as the multiagent tiger problem. Further evolution of the algorithms will only make the gap more significant. In this paper we introduce the *Georgia testbed for autonomous control of vehicles (GaTAC)*, which facilitates scalable and realistic problem domains pertaining to autonomous control of unmanned agents such as uninhabited aerial vehicles (UAVs). GaTAC provides a low-cost, open-source and flexible environment for realistically simulating the problem domains and evaluating solutions produced by multiagent decision making algorithms. We describe GaTAC in detail and demonstrate example problem settings that we are using in GaTAC. We expect GaTAC to facilitate the development and evaluation of scalable decision making algorithms with results that have immediate practical implications.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Experimentation

Keywords

scalability, testbed, autonomous vehicles

1. INTRODUCTION

There is a critical need for problem domains for sequential decision making (planning) in multiagent settings, which are realistic and scalable to accommodate more elements such as environmental states and other agents in a natural way. These problem domains will allow research in multiagent decision making to move beyond the traditional toy problems such as the multiagent tiger problem [10], machine maintenance problem [5], and the box pushing problem [13], to name a few. While these simple problem domains aid in illustrating the challenges of multiagent decision making, and sometimes turn out to be surprisingly rich in structure, their solutions do not have immediate practical implications. This gap will become significant as algorithms for multiagent decision making mature sufficiently to enable application.

We think that desired problem domains should, (a) be scalable to naturally allow for greater numbers of physical states, actions, observations, and agents while maintaining the plausibility of the problem; (b) produce solutions that are rich in structure and which have practical implications; and (c) be realistic and have popular appeal. In this paper, we introduce a problem domain that meets these criteria.

Unmanned agents such as uninhabited aerial vehicles (UAVs) are used in fighting forest fires [4], law enforcement [9] , and wartime reconnaissance. They operate in environments characterized by multiple parameters that affect their decisions, including other agents with common or antagonistic preference. The task is further complicated as the vehicles may possess noisy sensors and unreliable actuators. In such complex and unreliable settings, an autonomous UAV must choose navigational and surveillance actions that are expected to optimize its objective of say, timely reconnaissance of target while avoiding detection. UAV operation theaters may be populated by a single reconnaissance target or a host of other agents including UAVs working together as a team, or other hostile UAVs who could be reasoning about the subject UAV's actions [11].

Related research in this field focuses on automatically formulating the flight trajectories of the UAVs that optimize the coverage of region of surveillance interest while avoiding hazardous areas [1, 2, 8]. While relevant to static or high-altitude operation theaters, the research has little applicability to UAVs in low-altitude urban theaters (e.g. Raven RQ-11), because the interesting or hazardous objects could themselves be mobile; their actions must be predicted to enable optimal reconnaissance. This motivates the application of more expressive decision-theoretic models such as interactive POMDPs [7] and decentralized POMDPs [3].

In order to facilitate application of multiagent decision making to the problem domain of UAV reconnaissance and its evaluation, we have developed the *Georgia testbed for autonomous control of vehicles (GaTAC)*. GaTAC is a computer simulation framework for evaluating autonomous control of aerial robotic vehicles such as UAVs. It provides a low-cost and open-source alternative to highly complex and expensive simulation architecture. GaTAC uses a free, open-source and multi-platform flight simulator software called *FlightGear*. GaTAC deploys multiple instances of the flight simulator utilizing realistic 3D terrain data on a networked cluster of computing platforms using a scalable architecture. It is flexible allowing the interchange of instances of manually controlled vehicles with autonomous ones. It can be extended to include complex scenarios such as multiple UAVs, possibly hostile, and reconnaissance targets attempting to blend in with civilians.

In Section 2 we describe GaTAC in detail focusing on its architecture and its components. We illustrate an example application and evaluation of multiagent decision making to a specific problem setting simulated in GaTAC, in Section 3. Finally, in Section 4, we discuss the utility of realistic testbeds such as GaTAC toward

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

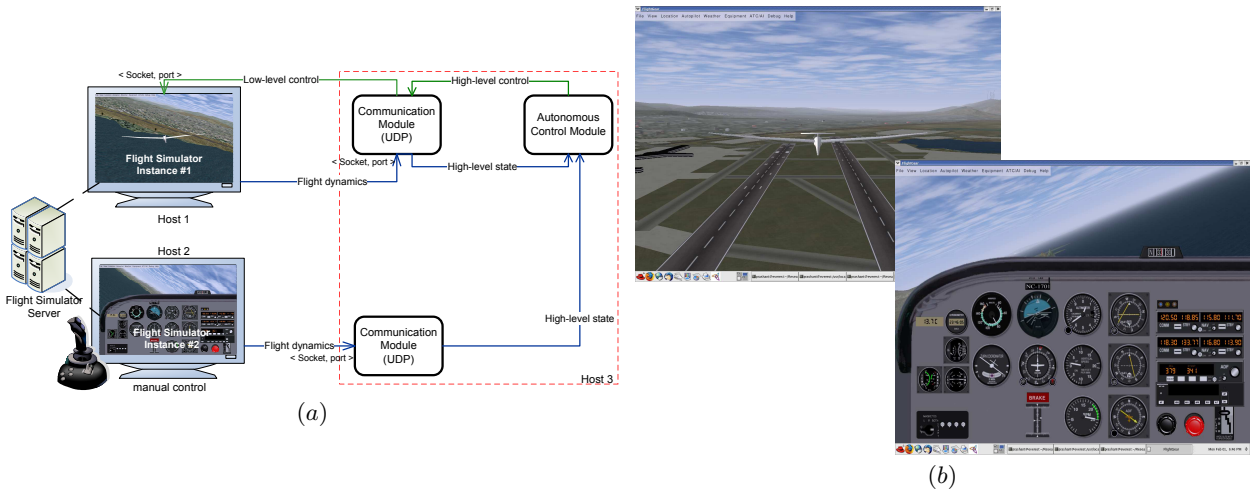


Figure 1: (a) **Design of GaTAC showing two networked instances of a flight simulator (FlightGear with 3D scenery from TerraGear), one autonomously and other manually controlled. GaTAC is extensible and more instances may be added.** (b) **Snapshots of a UAV flying within FlightGear. Notice the two different viewpoints – one of which is an external view while the other is the cockpit view.**

furthering the research on multiagent decision making.

2. TESTBED FOR AUTONOMOUS CONTROL

As we mentioned previously, the objective behind the development of GaTAC is to provide a realistic and scalable testbed for algorithms on multiagent decision making. GaTAC facilitates this by providing an intuitive and easy to deploy architecture that makes use of powerful, open-source software components. Successful demonstrations of algorithms in GaTAC would not only represent tangible gains but have the potential for practical applications toward designing autonomous UAVs. We think that multiagent decision making could make significant contributions in this area.

2.1 Architecture

We show a simplified design of the GaTAC architecture in Fig. 1, where a manually controlled UAV is interacting with an autonomous one. Briefly, GaTAC employs multiple instances of an open-source flight simulator possibly on different networked platforms that communicate with each other via external servers, and an autonomous control module that interacts with the simulator instances. GaTAC can be deployed on most platforms including Linux and Windows with moderate hardware requirements, and the entire source code is available. Gatac is implemented using C++ as the programming language.

We describe the individual components next.

2.1.1 Flight Simulator

We utilize *FlightGear* [12] as the flight simulator in GaTAC. FlightGear flight simulator project is an open-source, multi-platform, hyperrealistic flight simulator with a goal to develop a sophisticated flight simulator for use in academic and research environments. The entire source code of FlightGear written in C++ is available under GNU General Public License, allowing full extensibility. This dictates our choice of programming language. It provides a flexible platform with options to choose from multiple aircrafts, including UAVs (e.g., Predator), which could be operated manually or guided automatically by external programs. FlightGear uses a generic, six degrees-of-freedom flight dynamics model for simulating the motion of aerial vehicles. It simulates the effect of airflow on different part of the aircraft making it possible to perform the simulation based on geometry and mass information combined with more commonly available performance numbers for an aircraft. FlightGear utilizes realistic 3-dimensional scenery available from TerraGear, which virtually maps many parts of the world including models of the sky.

Of particular importance is that FlightGear provides multiple views of the flying aircraft, including external views from different viewpoints and an internal cockpit view. The cockpit view allows for a realistic flying experience, and is somewhat similar to the screens shown to the actual UAV operators. Finally, multiple instances of FlightGear may be run on different hosts and are linked together through external servers located in different countries. This multi-player mode allows for multiple aircrafts to fly simultaneously and see each other if the aircrafts are in visual range. This is a crucial functionality for its use in multiagent systems research.

Besides providing a hyperrealistic view of the field of operation of the UAV, the flight simulator allows realistic testing of decision-making problem formulations and the assumptions implicit in the formulation, which often go unnoticed and could be unreasonable. It may also explicitly bring out the advantages of decision-theoretic agents over other conventional methods used currently.

2.1.2 Communications Module

FlightGear allows remote control of the aircrafts through UDP socket based communication channels. The communication module in GaTAC (see Fig. 1(a)) establishes the UDP sockets, and sends or receives data from the instance of FlightGear. Control data at a low level is sent to FlightGear in order to remotely pilot the UAV. This data includes values for more than 30 flight parameters including the throttle, rudder, elevator and aileron settings. The communications module receives the aircraft's flight dynamics in real time from FlightGear. This includes data about the current latitude and longitude location of the aircraft, the values of the different flight surfaces, and current fuel level. During flight, the communication module continuously sends and receives data from the FlightGear instance at a pre-specified baud rate. GaTAC associates a communication module with every instance of FlightGear regardless of whether the corresponding aircraft is autonomously

or manually controlled. If the aircraft is manually controlled, the communication module simply receives the flight dynamics of the aircraft in order to remain informed about the state of that aircraft.

The communication module also provides a way to make UAVs aware of the positions of the other agents in the theater. This could be used to formulate the observations of the UAVs as needed.

2.1.3 Autonomous Control Module

In order to allow algorithmic control of the aircraft, GaTAC implements an autonomous control module (see Fig. $1(a)$). This module implements control actions such as *takeoff*, *fly straight*, *change heading* as well as aircraft *turns* using low-level actions that change the settings of the various flight surfaces to achieve the corresponding high-level actions.

We utilize these actions to provide a set of *high-level actions* which guide the UAV on a user-defined grid. These actions include moving the UAV in one of the four cardinal directions (*north*, *south*, *west*, and *east*), flying the UAV from one location to another defined using latitude and longitude coordinates, and making the UAV *hover* over a particular area. Additionally, GaTAC allows users to define their own grids of any size by specifying the latitude and longitude coordinates of the starting location of the grid, size of each grid cell, and the number of cells in two directions.

Consequently, algorithms for piloting the UAV now have a library of high-level actions made available to them. While we have made efforts to implement generic actions that could be composed to produce complex aircraft behavior, this library is extensible to include additional actions. The low-level implementations of the selected control actions are sent to the communications module for delivery to FlightGear. In response, the communication module sends back the flight dynamics, which are interpreted to obtain information about the state.

Because we intend to utilize GaTAC with multiagent decision making frameworks, it implements methods that read *policy tree* files in different formats including the output format of the popular *POMDP solver* [1]. We expect these policy trees to encode intelligent ways of guiding the UAVs. We intend to support additional formats including those output by the Multiagent Decision Process Toolbox [14].

We have made effort to make GaTAC independent of any particular type of decision-theoretic framework. It may be easily integrated with existing implementations by simply providing it with the behavioral policies generated by the various algorithms for decision making.

3. EXAMPLE PROBLEM SETTINGS

In order to demonstrate the use of GaTAC in multiagent decision making research, we provide example problem settings that could be simulated in GaTAC in straightforward ways. Beginning with relatively simple settings consisting of another hostile UAV and a ground reconnaissance target (Fig. $2(a)$), we may easily extend simulations in GaTAC to include complex scenarios comprising of a team of UAVs, multiple hostile UAVs and reconnaissance targets attempting to blend in with civilians. Notice that these settings have immediate pragmatic relevance.

A particular simple setting that we have been using to evaluate the performance of our decision-making algorithms is shown in Fig. $2(b)$. Analogous to the previous setting, UAV I performs reconnaissance of a potentially hostile theater populated by another UAV J, which is the target of reconnaissance. We have divided the theater into a 3×3 grid of equal-sized sectors – a common practice in actual combat theaters. For notational convenience, let's label UAV J as the *fugitive*. UAV I must track down the fugitive before it flees to the safe zone (indicated by the gray sector). The problem is made complex by assuming that the fugitive is unaware of its own location though it knows the location of the safe zone, and I may not be aware of the fugitive's location. The problem is further complicated if we realistically assume nondeterministic actions and observations.

Complexity of the above problem formulation is limited due to the particular multiagent decision making technique that we use. Its utility is that it demonstrates how decision-making problems could be formulated for GaTAC. The problem formulation may be extended to larger grids and more agents in a straightforward way.

3.1 Formulation

As a majority of the decision making frameworks require a formulation of the problem in terms of the physical states, actions, observations, and the corresponding dynamic functions, we provide examples of these below. Note that we assume partial observability of the agents' locations, and there are other ways to model this problem as well.

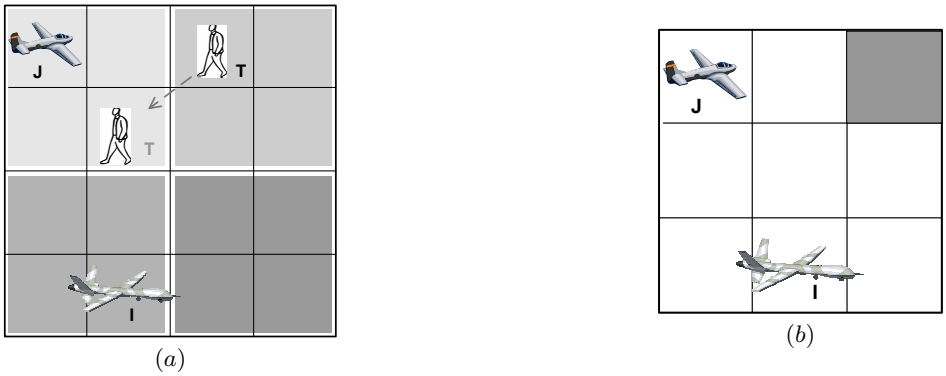
- We begin with the *physical states*. Because we must track the locations of both agents in the setting, a straightforward formulation entails 81 states for I and 9 states for a fugitive who is ignorant of I or 81 states otherwise. However, these numbers of physical states are unwieldy for many of the current multiagent decision-making algorithms. Hence, we reduce the state space for I by using the possible relative positions of the fugitive as states. Hence, possible states would be *same*, *north*, *south*, *east*, *west*, *north-west*, and so on. Based on degrees of coarseness, we may have different numbers of states. Our representation consists of 25 physical states for the UAV I. We initially assume that the fugitive is unaware of I resulting in 9 physical states for it.

- Both UAV I and the fugitive may move in one of the four cardinal directions or hover at their current position and listen to get informative observations. Thus the *actions* for both I and the fugitive are {*move_north*, *move_south*, *move_west*, *move_east*, *listen*}. We may synchronize the actions for the two agents in GaTAC by allocating equal time duration to the performance of each action.

- Typically, UAVs have infrared and camera sensors whose range is limited. Accordingly, we assume that both the UAV I and the fugitive can sense whether their respective target is north of them (*sense_north*), south of them (*sense_south*), west or east of them in the same row (*sense _level*) or in the same location as them (*sense _found*). For I the target is the fugitive, while the fugitive's target is the safe zone.

- Because we assume that the fugitive is unaware of the UAV I, its *transition function* is straightforward and simply reflects the possible nondeterministic change in grid location of the fugitive as it moves or listens. However, transitions in physical state of I are contingent on the joint actions of both agents as is usually the case in multiagent settings. Furthermore, the probability distribution over the next states is not only due to the nondeterminism of the actions, but is also influenced by the current physical state. For example, if the current physical state is *north* and the agents perform (move_south, listen), the next state could either remain *north* or become *north-north* based on whether the UAV I was in the bottom row or not. Thus, the formulation of the transition function is

[1]The POMDP solver is available at *http://www.cassandra.org/pomdp/code/index.shtml*.



(a) (b)

Figure 2: (a) **Example theater of UAV** I**, which performs low-altitude reconnaissance of a potentially hostile theater populated by another UAV** J **with conflicting objectives and a ground reconnaissance target,** T**.** I **may receive a noisy communication which informs it about the quadrants that likely contain** T**. UAV** J **may inform** T **that it is likely to be spotted, in which case** T **may move. The problem may be flexibly scaled by adding more targets and sectors.** (b) **A simpler setting that we use consisting of one other hostile UAV** J **whose task is to flee to the safe zone (gray sector) while avoiding UAV** I**.**

impacted by our decision to use a coarse state space.

• To provide an opportunity for the UAV I to catch the fugitive, we assume that the fugitive can sense the safe zone when it is within a distance of 1 sector (horizontally or vertically) from it. Because the fugitive knows the location of the safe zone, these observations would help it learn its own location(s). On the other hand, UAV I's observations of the fugitive are not limited by this constraint. Thus, if the fugitive is in any location that is north of I (including north-west or north-east), I receives an observation of *sense_north*. To simulate noise in the sensors, we assume that the likelihood of the correct observation is 0.8 while all others are equi-probable.

• The reward function is straightforward with the fugitive receiving a reward if its location is identical to that of the safe zone, and small costs for performing actions to discourage excessive action taking. Analogously, UAV I recieves a reward on performing an action and receiving an observation of sense_found, and incurs small costs for actions that lead to other observations.

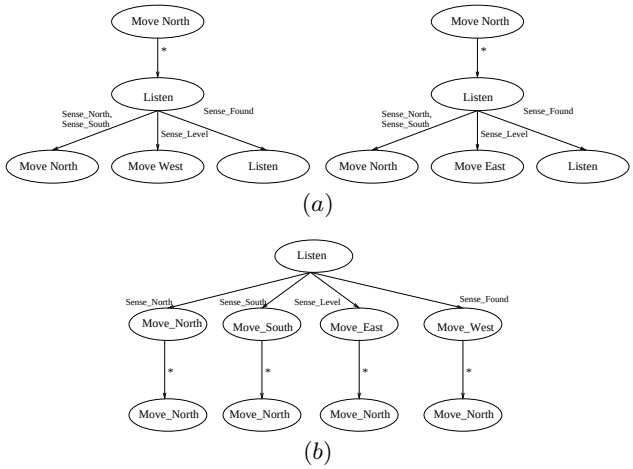
• Finally, the optimality criterion could be either finite horizon or infinite horizon with discounting. We believe that a small horizon of about 10 is sufficient to compute an optimal policy for the fugitive while larger horizons are needed for the UAV I.

3.2 Solution

We modeled the problem formulation described in Section 3.1 within the finitely-nested *interactive POMDP* framework [7]. We modeled the UAV at a single level of nesting, which models the fugitive at level 0. Thus, UAV I models the fugitive intentionally but ignorant of other agents in its environment. We assumed that the UAV and the fugitive perform their actions simultaneously and in step.

The size of the problem precludes exact solutions and approximate solution methods are needed. We picked the *interactive point-based value iteration (I-PBVI) technique* [6] in order to approximately solve the problem and generate policies for guiding the UAV. We utilized 50 level 1 belief points and ran the I-PBVI over 3 time horizons. After running for about a day on a 2.8GHz Xeon dual core with 4GB of RAM and Linux, we obtained satisfactory solutions.

In Fig. 3, we show the approximately optimal policy trees for



(a) (b)

Figure 3: (a) **Approximately optimal policy trees for the fugitive who is unaware of its location, obtained by solving its POMDP.** (b) **UAV** I**'s policy tree obtained by solving a level 1 I-POMDP using I-PBVI.**

the fugitive modeled as a POMDP and the level 1 UAV. Because the fugitive knows the location of the safe zone (but not its own), it starts by moving north followed by listening. Based on the informative observations it receives, the fugitive then moves accordingly. Given I's belief that it knows that the fugitive is unaware of its own location and it is itself unaware of the physical state, the UAV begins by listening. Based on its observations about the relative location of the fugitive, the UAV then moves accordingly followed by moving north in the last step. This is because I knows that the fugitive is going to be moving north since it knows the location of the safe zone.

While these policies appear satisfactory, we may evaluate them conclusively by deploying them in GaTAC. In particular, we could note the number of times the UAV is able to track down the fugitive over multiple runs. We deployed two instances of FlightGear in GaTAC for each of the two agents. The fugitive instance was controlled by randomizing between the policy trees in Fig. 3(a), while the UAV instance was controlled by the policy tree in Fig. 3(b). The autonomous control modules associated with both instances

parsed the policy trees. As we mentioned previously, the exact position of another instance is communicated to the subject UAV at all times using the communication module. The stochastic observations were implemented by adding noise while reporting the position of the fugitive to the UAV.

We simulated several runs of the UAV and the fugitive in GaTAC with the UAV and the fugitive's location randomly sampled. We noticed that the fugitive was caught about 60% of the times while it reached the safe house 20% of the times. No result was obtained the remaining times.

3.3 Scalability

Though the example problem we presented in this section is limited, which is due to the constraints imposed by the algorithm we used, GaTAC, being independent of the decision-making algorithm, can be used to simulate policies for more complex problem settings involving larger grids, more complex actions and observations and multiple agents which can be controlled either autonomously or manually. GaTAC is flexible and may be integrated easily with other problem formulations using the provided communication and autonomous control modules.

4. DISCUSSION

Toy problem domains abound in the multiagent decision making literature. These problem domains help highlight the challenges for multiagent decision making while allowing the application of these complex algorithms. However, solutions of these problems have minimal practical significance, and often, they fail to explicitly test the scalability of the algorithms.

We think that GaTAC fills this critical gap. In addition to being low cost and open source, it provides a satisfactory simulatory experience of a problem domain that has popular appeal, and is extensible. GaTAC represents a realistic testbed for multiagent decision making research, and a first step in our knowledge toward enabling decision-making algorithms to cross over to domains of practical import. Problem domains of different sizes in GaTAC could also function as common benchmarks for comparing the various algorithms within individual decision-making frameworks.

Finally, the problem of autonomous UAV reconnaissance requires important contributions from the multiagent decision making community. Successful evaluation of the performance of potentially useful algorithms in a simulation environment like GaTAC paves the way for prototype deployments in actual UAVs.

Acknowledgement

This research is supported in part by AFOSR through grant no. #FA9550-08-1-0429 and in part by a NSF CAREER award with grant no. #IIS-0845036. Views and findings expressed in this article are those of the authors alone and do not reflect in any way upon the funding agencies. The authors also wish to thank the anonymous reviewers for their valuable comments.

5. REFERENCES

[1] R. Beard and T. McLain. Multiple uav cooperative search under collision avoidance and limited range communication constraints. In *IEEE Conference on Decision and Control*, 2003.

[2] R. Beard, T. McLain, M. Goodrich, and E. Anderson. Coordinated target assignment and intercept for unmanned air vehicles. In *IEEE Conference on Robotics and Automation*, 2002.

[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] D. Casbeer, R. Beard, T. McLain, L. Sai-Ming, and R. Mehra. Forest fire monitoring with multiple small uavs. In *American Control Conference*, pages 3530–3535, 2005.

[5] P. Doshi and P. Gmytrasiewicz. Monte carlo sampling methods for approximating interactive pomdps. *Journal of Artificial Intelligence Research*, 34:297–337, 2009.

[6] P. Doshi and D. Perez. Generalized point based value iteration for interactive pomdps. In *Twenty Third Conference on Artificial Intelligence (AAAI)*, pages 63–68, 2008.

[7] P. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

[8] S.-M. Li, J. D. Boskovic, S. Seereeram, R. Prasanth, J. Amin, R. Mehra, R. Beard, and T. McLain. Autonomous hierarchical control of multiple unmanned combat air vehicles. In *American Control Conference*, 2002.

[9] D. Murphy and J. Cycon. Applications for mini vtol uav for law enforcement. In *SPIE 3577:Sensors, C3I, Information, and Training Technologies for Law Enforcement*, 1998.

[10] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized pomdps : Towards efficient policy computation for multiagent settings. In *International Joint Conference on AI*, pages 705–711, 2003.

[11] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus. Security in multiagent systems by policy randomization. In *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 273–280, 2006.

[12] A. R. Perry. The flightgear flight simulator. In *UseLinux*, 2004.

[13] S. Seuken and S. Zilberstein. Improved memory bounded dynamic programming for decentralized pomdps. In *Uncertainty in Artificial Intelligence (UAI)*, pages 2009–2015, 2007.

[14] M. Spaan and F. Olihoek. The multiagent decision process toolbox: Software for decision-theoretic planning in multiagent systems. In *Workshop on Multiagent Sequential Decision Making in Uncertain Domains, AAMAS*, pages 107–121, 2008.

A new approach for solving large instances of DEC-POMDPs: Vector-Valued DEC-POMDPs.*

Arnaud Canu
GREYC (UMR 6072), Université de Caen Basse-Normandie
Campus Côte de Nacre, boulevard du Maréchal Juin
BP 5186 - 14032 Caen CEDEX, FRANCE
arnaud.canu@info.unicaen.fr

Abdel-Illah Mouaddib
GREYC (UMR 6072), Université de Caen Basse-Normandie
Campus Côte de Nacre, boulevard du Maréchal Juin
BP 5186 - 14032 Caen CEDEX, FRANCE
mouaddib@info.unicaen.fr

ABSTRACT

Multi-agent planning under uncertainty, when execution is decentralized and there is no communication, is one of the hardest problem studied by the planification community. In such a problem, the classical approaches consist in computing a joint behaviour policy for all of the agents. DEC-POMDP is the most robust model to describe such a problem and to compute a joint policy. However, the high complexity of this model limits its applications in real world domains. To our knowledge, there is no solver able to deal with a general DEC-POMDP considering more than two agents. Indeed, the complexity grows in a combinatorial explosion with the number of agents. In order to overcome this limitation, some new models have been introduced, like ND-POMDPs which are able to solve problems with a lot of agents. Unfortunalty, those approaches need some strong assumptions such as independent transitions or observations between agents. Such assumptions drastically reduce the applicability to real-world domains. In this paper, we will show a new method for solving problems with more than two agents which does not need any assumption on the problem structure. We will describe how we can solve a DEC-POMDP with several agents by computing an independent policy for each agent and compute a coordination step afterwards. Experimental results show that our approach remains competitive in comparison with the most popular algorithms with a degradation of the overall solution quality but deals with more than two agents. We also discuss about how we could increase the solution quality.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coherence and coordination, Multiagent systems*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Plan execution, formation, and generation*

General Terms

Algorithms, Theory, Experimentation

*This work is supported by the DGA (Direction Générale de l'Armement).

AAMAS 2010 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2010, Toronto, Canada.

Keywords

Agent Reasoning :: Planning (single and multi-agent), **Agreement Technologies ::** Collective decision making, **Agent Cooperation ::** Teamwork, coalition formation, coordination, **Agent Cooperation ::** Distributed problem solving

1. INTRODUCTION

Planning under uncertainty is a classical problem which can usualy be solved with a Markov Decision Process (an MDP [9]). This formalism has been introduced in order to formulate problems where the outcomes of agents's actions are uncertain. DECentralized MDPs (DEC-MDPs) have been introduced to extend MDPs to multi-agent settings. Finally, DECentralized Partially Observable MDPs (DEC-POMDPs [3]) have been introduced in order to extend DEC-MDPs to partially observable settings.

DEC-POMDP can express every kind of multi-agent problem, which makes it a robust model, but has a high complexity for computing a solution. The number of possible solutions is so huge that it is not realistic to sweep all the search space to find the best one. A lot of work has been done to reduce the complexity and great improvements have been accomplished, but one point is still as hard as it was at the begining: dealing with a large number of agents. When we add agents, the complexity of the problem grows exponentially and, to our knowledge, none of the existing solvers can compute a solution for more than two agents.

Some new formalisms have been introduced in order to overcome this limitation. Network Distributed POMDPs (ND-POMDPs [8],[6]) are one of the most performant ones and can solve problems with much more than two agents. Those methods still have a big drawback: they need strong assumptions. ND-POMDPs for example can only solve problems where agents have independent transition and observation functions. Finally, we can not solve a DEC-POMDP with such a method, but only a problem which can be described in a decomposed fashion. Then, such an assumption stronlgy limits the number of problems that can be solved.

Other approaches have been studied which consist in computing an independent policy for each agent (without taking the other agents into account). In those approaches, the coordination aspect is not considered during the planning task as it is in standard methods. In OC-DEC-MDPs [4], the coordination is made with constraints on tasks executions. In 2V-DEC-MDPs [7], it is made during a second comput-

ing step. Such an approach is really interesting because it gives the possibility to solve a problem with more than two agents, while keeping a DEC-MDP formalism. However, those approaches are limited to DEC-MDPs (which are fully observable problems) and still need strong assumptions on the problem structure.

We introduce in this paper a new approach to solve DEC-POMDPs with more than two agents. We called this approach 2V-DEC-POMDP (Vector-Valued DEC-POMDP). After a few words about the existing methods, we will explain how our formalism works and how it can compute a policy for a given agent. We will show in section 4 how to solve a classical DEC-POMDP with our method and compute a joint-policy. In section 5, we describe experiments we developed and results we obtained. We will show that we can solve a DEC-POMDP with up to 4 agents without making any assumption on the problem structure and that we remain competitive with the most popular algorithms with a degradation of the overall solution quality. Finally, we will talk in section 6 about how we could increase the quality of our policies, what are our method's limits and how we can overcome those limits.

2. BACKGROUND

Before introducing 2V-DEC-POMDPs, there are planning formalisms we need to be familiar with. First of all are (PO)MDPs, then DEC-(PO)MDPs. We will also present 2V-DEC-MDPs.

2.1 Planning with one agent

2.1.1 MDP

In planning theory, a classical problem is, for an agent, to compute a policy using stochastic actions (when the agent is not sure of the effect of its actions). Markov Decision Process (MDP) [9] framework has been introduced in order to solve such a problem and compute an optimal policy. An MDP is a tuple $\langle S, A, T, R\rangle$, with: (1) S a set of states, (2) A a set of actions, (3) $T : S \times A \times S \to [0; 1]$ the transition function and (3) $R : S \times A \times S \to \mathbb{R}$ the reward function which expresses both positive reward for goal states and negative reward for hazardous states.

In order to solve such a problem, we need an optimality criterion which will give us a value for a given policy. One of the most used is the expected reward on an infinite horizon. With $\gamma \leq 1$, the optimal value function V^* of a state s is defined by:

$$V^*(s) = \max_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot V^*(s')), \forall s \in S$$

Using such a criterion, we can compute a policy π. A policy is a mapping $\pi : S \to A$ and the optimal policy π^* is such that:

$$\pi^*(s) = \operatorname*{Argmax}_{a}(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot V^*(s')), \forall s \in S$$

2.1.2 POMDP

A POMDP (Partially Observable MDP) is an MDP in which an agent can not fully observe its environment. When it executes an action, it does not know in which state it will end up in but it receives an observation. The model is the same than the one used for an MDP and we add an observations set Ω and an observation function O:

$$O : S \times A \times S \times \Omega \to [0; 1]$$

In a PO problem, agents do not count on states while computing a plan, but on belief-states, which are a probability distribution over states.

2.2 Planning with two or more agents

2.2.1 DEC-MDP

The DEC-MDP formalism has been introduced in order to deal with problems where two or more agents are involved. In the same way MDPs have been extended to POMDPs, DEC-MDPs have been extended to DEC-POMDPs, in order to solve problems where two or more agents are involved and where each agent has a partial view of its environment.

2.2.2 DEC-POMDP

A DECentralized Partially Observable MDP [3] (DEC-POMDP) is a tuple $\langle n, S, b_0, A, T, \Omega, O, R, Z\rangle$ where:

- n is the number of agents involved in the problem,
- S is a finite set of states,
- $b_0 \in \Delta S$ is the initial belief state,
- A_i in $A = \{A_1, ..., A_n\}$ is a finite set of actions available to agent i (with $A_1 \times ... \times A_n$ the set of joint actions),
- T is the transition function (on joint actions),
- Ω_i in $\Omega = \{\Omega_1, ..., \Omega_n\}$ is a finite set of observations available to agent i,
- $O : A_1 \times ... \times A_n \times S \times \Omega_1 \times ... \times \Omega_n \to [0, 1]$ is the observation function,
- $R : S \times A_1 \times ... \times A_n \times S \to \mathbb{R}$ is the reward function,
- Z is the number of steps of the problem solution.

In the following, we will write X_i if i refers to an agent, X^i otherwise. In such a problem, a policy is represented as a tree where nodes are actions and edges are observations. A solution for the problem will be a joint policy tree, one per agent.

2.3 Using local interactions

We will describe two different approaches for solving DEC-MDPs. Those two approaches are based on the same idea: solving the problem of each agent independently of the other agents but keeping the coordination aspect inside the individual problems.

2.3.1 OC-DEC-MDP

In [4], the OC-DEC-MDP formalism has been introduced in order to solve large decision problems with temporal, precedence and resource constraints. In this approach, each agent has a set of tasks to execute and know what are the task sets of the other agents. It can then choose the order in which it will execute its tasks, while considering the consequences of its actions on the other agents.

In such an approach, agents solve their problems independently of each others but do not completely ignore the other agents. However, the agents take each others into account while choosing the order in which they will execute their tasks and when.

2.3.2 2V-DEC-MDP

In [7], the Vector-Valued Decentralized Markov Decision Process (2V-DEC-MDP) framework has been proposed to coordinate locally the actions of a group of agents. Assuming without loss of generality that all agents are identical, a 2V-DEC-MDP is a set of 2V-MDPs, one per agent. A 2V-MDP is composed of an off-line part (to represent the behaviour of an agent ignoring its neighbours) and an on-line part (to adapt its actions with the other agents in the neighborhood).

The off-line part is an MDP such as the one we described in sec.2.1.1. The on-line part of a 2V-MDP is built with the computation of a local social impact, according to local observations. The functions for computing the value of the social impact are ER for the individual reward (using the MDP), JER for the group interest and JEP for the negative impact on the group.

The on-line part is a sequence of small DEC-MDPs with a special transition model. During this part, an agent builds a new DEC-MDP after each decision it takes (only considering agents in its neighborhood and planning on a short horizon, less than 3). In this DEC-MDP, the probability for an agent i to go from a joint state s to a joint state s' only depends on a_i.

Deriving a policy for this DEC-MDP consists in solving a multi-criteria Bellman equation based on an Augmented Reward $AR = (ER, JER, JEP)$. To solve this equation, a regret based value iteration using $LexDiff$ operator [7] has been designed.

For each possible policy π_i, $LexDiff$ builds a vector $v = (ER(\pi_i), JER(\pi_i), JEP(\pi_i))$ and normalize each value vector $v_i = (v_i^1, v_i^2, v_i^3)$ to a utilitie vector $v_u = (v_u^1, v_u^2, v_u^3)$. $LexDiff$ then uses a leximin operator to find the best vector: it permutes those utilities vectors so that each vector (v^1, v^2, v^3) be such that $v^1 \geq v^2 \geq v^3$. The best vector (and so the best policy) is then derived from a lexicographic order: for two vectors $v_a = (v_a^1, v_a^2, v_a^3)$ and $v_b = (v_b^1, v_b^2, v_b^3)$, we choose v_a if $v_a^1 > v_b^1$ and v_b if $v_a^1 < v_b^1$. If $v_a^1 = v_b^1$, we compare v_a^2 and v_b^2, and so on.

2.3.3 Scope of those approaches

OC-DEC-MDPs and 2V-DEC-MDPs are powerfull tools for solving DEC-MDPs with a lot of agents, but some issues remain. In OC-DEC-MDPs, we can only solve DEC-MDPs with a very peculiar structure, where each agent has a set of tasks to accomplish. In 2V-DEC-MDPs, we solve standard DEC-MDPs, but we only consider very basic interactions (help or block). Moreover, none of those approaches work on DEC-POMDPs.

3. OUR FORMALISM: 2V-DEC-POMDPS

We will now introduce our formalism. First, we will explain its concept. Second, we will show how to formalize a problem using a 2V-DEC-POMDP. Finally, we will give a first approach to solve such a 2V-DEC-POMDP.

3.1 Concept

We will describe how a 2V-DEC-POMDP works, in addition to several examples. We will propose an answer to the question: **How to get rid of the combinatorial explosion due to the number of agents in a DEC-POMDP, without loosing expressiveness?**

3.1.1 Statement

DEC-POMDPs are really hard to solve, so researchers tried to find ways to reduce this complexity in order to solve problems with larger horizons, number of states or observations. Some of them tried to reduce the number of policies we keep in memory [10], to work only with "probably intersting policies" [5], to use memory-bounded controllers [1] or to solve goal-oriented problems [2]. If we look deeper into a DEC-POMDP mechanism, we can see that the "decentralized" aspect, which implies taking into account every possible combination of joint policies, involves a combinatorial explosion. It is one of the biggest reasons of DEC-POMDPs' complexity (without this point, a DEC-POMDP would "only" be as complex as a POMDP) but almost nobody tried to work on this aspect of a DEC-POMDP.

ND-POMDPs [10] have been introduced addressing this specific problem. In an ND-POMDP, each agent has its own transition and observation functions (they are completly independent) and rewards are computed on local sub-sets of agents. Such a formalism can address much bigger instances in terms of number of agents, but the expressiveness is very limited.

In [7], Boussard showed how an approach which considers the individual and the coordination aspects as two different problems can have good results with DEC-MDPs (or DEC-POMDPs with full local observation). So, why not trying such an approach with general DEC-POMDPs?

3.1.2 Consequence

After making this statement, we tried a way to get rid of this combinatorial explosion and introduced the 2V-DEC-POMDP formalism. In the same way as in 2V-DEC-MDPs, an agent tries to solve its own problem independently of other agents and coordinates itself with the other agents during its life cycle. We will separate our problem in two smaller ones. One of them will be the problem of a solo agent, making the assumption it is alone (this problem will answer to the agent how to meet its objectives) and the other one will be a coordination problem (which will answer to the agent how to react if it meets with other agents).

Contrary to 2V-DEC-MDPs, we wanted a real coordination problem, capable of expressing any type of interaction. This coordination problem, which contains the combinatorial complexity, can be seen as a collection of small problems, one per possible type of interaction. By splitting this problem into several independent sub problems, we will drastically decrease the combinatorial complexity. Such an approach **do not limit the number of interactions**.

Briefly, when we consider a problem, we split it into two parts (individual and coordination), solve each part and merge those two solutions in order to compute a global solution (for the original problem). Like in 2V-DEC-MDPs, we will not compute a joint policy for the global problem. Instead of that, each agent will solve its own problem in two steps: the first will be offline (computing a solution for each part), while the second will be online (after each action, updating its belief state and choosing its best next action according to its value functions).

3.1.3 Examples

Those two examples show how a 2V-DEC-POMDP can be used to describe a problem:

- Example 1: docker robots. We can imagine a problem

where several agents are in a room, with several items on the ground. Agents have to catch those items and to put them on shelves, but some items are too heavy to be carried by a single agent. In a standard DEC-POMDP, agents would compute an (optimal) joint policy in which they will store all the items. In a 2V-DEC-POMDP, every agent will compute its own policy in two steps:

  - computing an individual policy, to store a maximum number of items, independently of the other agents,
  - computing coordination policies, exactly one per each possible type of interaction (how to carry a heavy item with another agent, how to store its items without blocking its neighborhood, etc...).

  So, during the execution, the robot will follow its individual policy. But, if it meets another robot, it will take this opportunity and apply one of its coordination policies.

- Example 2: robots platooning. In this problem, several robots have to move in a coordinate fashion, and to reach a destination while not breaking the platoon. Such a problem can be solved by a traditional DEC-POMDP but is really hard. With a 2V-DEC-POMDP, we solve:

  - the individual problem, which consists in reaching the destination,
  - the coordination problems: staying in the platoon, waiting for slow agents, avoiding collisions with other agents, etc...

  Like for the dockers, robots in the platooning problem will mainly follow their individual policy, but will use their coordination policies to solve local problems in order to keep the platoon unbroken. Splitting individual and coordination policies will make this problem much easier.

Those are just examples, but 2V-DEC-POMDP can express every type of problem that a DEC-POMDP can express.

3.2 How to define a problem using a 2V-DEC-POMDP

A 2V-DEC-POMDP "PB" is the union of two small problems. We write $PB = \langle n, Pb^{ind}, Pb^{coo} \rangle$, with n the number of agents, Pb^{ind} the individual problem and Pb^{coo} the coordination problem. We will use $S = \{s^1, \ldots, s^{|S|}\}$ the set of every possible joint state for our problem with $s^i = (s_1^i, \ldots, s_n^i)$ and s_k^i the part of s^i relative to the agent k (and we write s_i the part of s relative to the agent i).

3.2.1 The individual problem: Pb^{ind}

Pb^{ind} will be a POMDP. Once the problem of one agent is clearly defined, writing the associated POMDP is easy (we will see in the next section how to extract this POMDP from a DEC-POMDP). For an agent i, the states for this POMDP will be $\{s_i^1, s_i^2, \ldots, s_i^{|S|}\}$ and the actions will be A. Solving this part can be made with any POMDP solver. Actually, we use the TOP solver, because of its speed. All we need is a set of Alpha Vectors for the output, so we can have a value function $V^{ind}(b, a)$ with b a belief-state and a an action.

3.2.2 The coordination problem: Pb^{coo}

This part is more complex than the first one. We will not compute two functions like we do in a 2V-DEC-MDP (JEP and JER), but a single one. $Pb^{coo} = \{IC^0, \ldots, IC^j\}$ will be a set of coordination problems IC^i, one for each possible interaction. Each IC^i is associated to an Interaction class.

Definition 1. Interaction class. In a given state, we use a criterion to decide if two agents are in interaction or not. Two agents i and j in respectively states s_i and s_j are in interaction when their joint reward JR (the reward of the group) at these states is not the sum of their individual rewards R at these states. More formally speaking : $\exists a \in A^2, JR(s_i, s_j, a) \neq R(s_i, a_i) + R(s_j, a_j))$. An interaction class IC is a set of joint states. We distribute S among j sets according to which agents are in interaction (with $0 < j \leq |S|$). After that, we reduce each set to the agents involved in it (example: we build a set S^i from S with states where the 5 first agents are in interaction and we replace every state $s^i = (s_1^i, \ldots, s_n^i)$ in S^i by a state $s'^i = (s_1^i, \ldots, s_5^i)$). Each reduced set is then an interaction class.

We made the choice to solve Pb^{coo} with a meta-MMDP (a set of MMDPs) but any other multi-agent formalism could work (see sec. 6 for more discussions about this point). So, for each interaction class IC^i with k agents, we define a MMDP $Pb^i = \langle S', A', T, R \rangle$ with:

- $S' = IC^i \cup \epsilon$, with ϵ an abstract state representing the transition to another interaction class,
- $A' = A^k$ the set of possible joint actions,
- $T : S' \times A' \times S' \rightarrow [0; 1]$ the transition model between each state where:
  - $\forall a^i \in A', T(\epsilon, a^i, s^j) = p$ and $p = 1$ if $s^j = \epsilon$, $p = 0$ otherwise,
  - $T(s^j, a^i, \epsilon) = p$, and $p = 0$ if applying a^i in the state s^j can not lead to another interaction class (else, p is computed such as $\sum_{s'} T(s^j, a^i, s') = 1$).
- $R : S' \times A' \rightarrow \mathbb{R}$ the reward function.

Solving such an MMDP is easy (it can be solved like an MDP). All we need for the output is a value function $V^{Pb^i} : S' \times A' \rightarrow \mathbb{R}$.

4. USING A 2V-DEC-POMDP TO SOLVE A DEC-POMDP

We will show how to solve a DEC-POMDP using a 2V-DEC-POMDP. We have to convert the DEC-POMDP into a 2V-DEC-POMDP, using a method to compute the interaction classes. Once this is done, we will solve this new problem and convert the solution into a DEC-POMDP's solution.

4.1 Computing the interaction classes

If we want to convert a DEC-POMDP into a 2V-DEC-POMDP without any human intervention, we need a way to automatically compute the ICs. We can imagine several criteria to distribute states among ICs, which respect this rule: each state is in one IC and only one.

We chose a criterion based on the reward. We consider an agent ag_1 is in interaction with an agent ag_2 in a state s if and only if:

$$\exists (a_3, \ldots, a_n) \in A_3 \times \cdots \times A_n, \exists a_2, a_2' \in A_2 \ tq \ :$$

$$\underset{a_1 \in A_1}{\text{Argmax}}\, R(a_1, a_2, ..., a_n, s) \neq \underset{a_1' \in A_1}{\text{Argmax}}\, R(a_1', a_2', a_3, ..., a_n, s)$$

Intuitively, it means ag_1 interacts with ag_2 if the choice of the best action a_1 for ag_1 (the action which maximize the immediate reward) depends on the action ag_2 chosen by ag_2.

So, using this criterion to define an interaction, we will have an IC for states where ag_1 works alone, another for states where ag_1 is in interaction with ag_2, etc...

4.2 How to convert a DEC-POMDP into a 2V-DEC-POMDP

We take, for the input, a DEC-POMDP $\langle S, A, T, R, \Omega, O \rangle$ with n agents where each state s is a tuple $s = (s_1, \ldots, s_n)$ of agents' states (example: a state s which describes positions of every agent in a problem can be split into n states s_i, one per agent i, describing the position of i). We will split the DEC-POMDP into two new problems: Pb^{ind} and Pb^{coo}.

4.2.1 Generating the individual problem: Pb^{ind}

Computing a POMDP $\langle S_i, A, T_i, R_i, \Omega_i, O_i \rangle$ associated to the DEC-POMDP is simple: we average the DEC-POMDP. For an agent i, we build:

- $S_i = \{s_i^1, s_i^2, \ldots, s_i^{|S|}\}$,
- A is directly cloned from the DEC-POMDP,
- we consider:

  $$S_{-i} = S_1 \times \cdots \times S_{i-1} \times S_{i+1} \times \cdots \times S_n$$

  We build A_{-i} and O_{-i} in the same way.
- $T_i : S_i \times A \times S_i \rightarrow [0; 1]$ with:

  $$T_i(s_i, a_i, s_i') = \underset{\substack{s \in S_{-i}, \\ a \in A_{-i}}}{\text{avg}} \sum_{s' \in S_{-i}} T((s_i, s), (a_i, a), (s_i', s'))$$

  ,
- $R_i : S_i \times A \times S_i \rightarrow \mathbb{R}$ with:

  $$R_i(s_i, a_i, s_i') = \underset{\substack{s \in S_{-i}, \\ a \in A_{-i}}}{\text{avg}} \underset{s' \in S'_{-i}}{\text{avg}} R((s_i, s), (a_i, a), (s_i', s'))$$

- Ω_i is directly cloned from the DEC-POMDP,
- $O_i : S_i \times A \times S_i \times \Omega_i \rightarrow [0; 1]$ with $O_i(s_i, a_i, s_i', o_i) =$

  $$\underset{\substack{s \in S_{-i}, \\ a \in A_{-i}, \\ s' \in S'_{-i}}}{\text{avg}} \sum_{o \in O_{-i}} O((s_i, s), (a_i, a), (s_i', s'), (o_i, o))$$

4.2.2 Generating the coordination problem: Pb^{coo}

For the coordination problem, we will generate $Pb^{coo} = \{Pb^1, \ldots, Pb^k\}$ the k sub problems describing each possible interaction. First of all, we need to split the states of S into a set of k interaction classes (sec. 3.2.2): the next section of this paper details this aspect of the problem. Once ICs are generated, we build an MMDP $Pb^j = \langle S^j, A, T^j, R^j \rangle$ per IC j with m agents:

- $S^j = IC^j \cup \epsilon$ (the special state ϵ is described previously in sec. 3.2.2),
- A is directly cloned from the DEC-POMDP,
- we write S_{-S_j} the set of tuples $(s_{m+1}, ..., s_n)$ associated to the agents $(m+1), ..., n$ which are not in IC^j. We build A_{-A_j} in the same way.
- $T^j : S^j \times A^m \times S^j \rightarrow [0 : 1]$ with:
  - $T^j(s, a, s') =$

    $$\underset{s_c \in S_{-S_j}}{\text{avg}} \underset{a_c \in A_{-A_j}}{\text{avg}} \sum_{s_c' \in S_{-S_j}} T((s, s_c), (a, a_c), (s', s_c'))$$

  - $T^j(s, a, \epsilon) =$

    $$\underset{s_c \in S_{-S_j}}{\text{avg}} \underset{a_c \in A_{-A_j}}{\text{avg}} \sum_{s' | s' \in S \wedge s' \notin S^j} T((s, s_c), (a, a_c), s')$$

- $R^j : S^j \times A^m \rightarrow \mathbb{R}$ with:

  $$R^j(s, a) =$$

  $$\underset{s_c \in S_{-S_j}}{\text{avg}} \underset{a_c \in A_{-A_j}}{\text{avg}} \sum_{s' \in S} R((s, s_c), (a, a_c), s')$$

Using an "average" operator for computing T and R is a first idea but not necessarily the most efficient one. Finding a smart operator is a subject left to futur works (see sec.6 for details).

4.2.3 usage of sum and averaging operator

During the convertion process, we use a lot the averaging operator. We also use the sum operator. Such an approach can decrease the precision of the extracted sub-problems. We will see in sec. 5 that we are a bit far from optimal solutions, compared to other algorithms. Those averaging operators are probably responsible of this lack of optimality.

In sec.6.3, we analyze this problems and we propose in sec.7 some ways to overcome those difficulties.

4.3 How to solve the problem and generate a solution for the DEC-POMDP

First of all, we need to solve Pb^{ind} and Pb^{coo} independently. We mentioned that we can solve Pb^{ind} with a classical POMDP solver. Solving Pb^{coo} is a little more complicated:

- we solve every MMDP,
- we write a value function $V(s, a)$ for PB^{coo}: for each state $s \in S$ and each joint action $a \in A^n$, we look for the IC associated and reduce s and a to the concerned agents. We then write, with $s^{reduced} \in IC^i$:

  $$V(s, a) = V^{Pb^i}(s^{reduced}, a^{reduced})$$

- finally, we write a function $V^{coo}(b, a)$ with b a joint belief-state with: $V^{coo}(b, a) = \sum_{s \in b} b(s) \dot{V}(s, a)$.

Now, each agent has two functions V^{coo} and V^{ind} and an initial belief-state b. In a traditional DEC-POMDP, we generate a (joint) policy but, here, the behavior will be computed during the on-line step. During its life, an agent i acts as follow:

- It chooses its next action a:
  - First, it computes b_i:
$$\forall s_j \in S_j, b_i(s_j) = \sum_{s \in S_{-j}} b((s, s_j))$$
  - Second, it chooses the best action a:
$$a = \underset{a \in A^n}{\text{Argmax}}\, LexDiff(V^{ind}(b_i, a), V^{coo}(b, a))$$
  - Third, it applies a_i extracted from a.

  LexDiff needs to put coefficients c^{ind} on Pb^{ind} and c^{coo} on Pb^{coo}. Choosing c^{ind} and c^{coo} is not trivial and can have a big impact on the final behaviour. Experiments showed we can not definitely choose them: for some problems, we need c^{ind} to be greater than c^{coo} but, in other problems, we need the opposite. Actually, we manually choose those coefficients, but we need a way to compute them (in order to make the conversion fully human independent). This problem will be the subject of a future work.

- It updates its belief-state. With b_o^a the belief-state in which we end after a joint action a is applied in a belief-state b and the agent received an observation o, we write:
$$b_o^a(s) = \frac{O(o|s,a) \sum_{s' \in S} P(s|s',a) b(s')}{\sum_{s' \in S} \sum_{s'' \in S} O(o|s'',a) P(s''|s',a) b(s')}$$
  In this equation, we compute $O(o|s,a)$ with a sum of all the joint observations containing o. We don't have a, because we don't know what actions the other agents did, but we have an optimal joint action we computed previously, so we use this joint action for updating b.

5. EXPERIMENTAL RESULTS

We made some experiments to test our method. We will present first the context in which we made those experiments and, second, the results of those tests.

5.1 Context

Because our formalism does not produce a joint policy tree but a set of value functions, it is not trivial to compare it with other existing methods. We made two kinds of comparison:

- We computed an ADR (Average Discounted Reward): this test is made in two steps. First, we compute a policy (a tree, a value function, or anything else). Second, we simulate the execution of this policy (in a simulator, each agent follows its own policy and the world evolves according to the transition function. After each action, an agent receives an observation according to the observation function). During this execution, we cumulate discounted rewards (total reward R after n steps is $R = r_0 + \gamma^1 r_1 + \cdots + \gamma^n r_n$, $\gamma \leq 1$). If we run enough simulations and average the cumulated reward of each execution, we obtain a representative value of the policies' quality.
- We artificially built a joint policy tree: using the value functions, we can build a joint policy-tree. According to the initial belief-state, we can choose an initial action for each agent. Then, we can add an edge for each possible observation, update the belief state of each agent according to this observation and choose the next action, and so on. Once we have those policy-trees, we can compute the cumulated reward.

Each of those methods have positive and negative aspects. With an ADR, the good point is we can work on an "infinite" horizon (we stop the simulation at step i when γ^i becomes unsignificant) and we have a result near to what would happen in "real world". The problem is the possibility to "ignore" some parts of the joint policy, because their probability is too small, so the global discounted reward is misestimated. This problem does not exist when we compute the value of a policy tree, because we are sure to explore the totality of the tree, but we then have to stop the policy at an arbitrary horizon (we can not compute an infinite tree), which is not representative of the real behaviour of an agent following this policy. Moreover, computing the value of all the tree is not necessary representative of what would happen in a real execution (some parts can have a high value but nearly never be used).

We made the choice to compare our results with the PBIP solver. According to the rate quality/time, PBIP is actually the best solver. In [5], PBIP is better than MBDP (and its extensions). Moreover, we used the solver written by the PBIP author for our tests, so we are sure it works and it is representative of this algorithm quality.

We wrote a DEC-POMDP solver using our formalism, a simulator to compute the ADR and a policy-tree extractor to compute its value. We used TOP to solve the individual part and MMDPs to solve the coordination part. We used the same file format for the input problem as in the PBIP solver, so we are sure that the two algorithms solve the same problem.

We give results with the following problems, which are standard DEC-POMDP benchmarks: Dec-Tiger and Meeting in a 3x3 Grid. For the ADR tests, we made 10000 simulations per problem. We used the following coefficients to solve those problems: 0.8 for π^{ind} and 0.2 for π^{coo}.

5.2 Results

5.2.1 Using an ADR

The table 1 shows results we had while making tests with an ADR. We compare PBIP with 2V (Vector-Valued, our method). We chose a horizon of 10 for PBIP, which is enough for computing an ADR in those problems. Moreover, PBIP needs a second parameter which is the number of trajectories explored: we set a value of 7 to this parameter (which is, according to the author of PBIP, the best parameter to use in those problems).

Here, 2V is faster than PBIP for every problem. We note that our method brings a better ADR than PBIP on the meeting in a Grid problem. This problem fits very well to our method, because interactions are very local, in opposition to dec-tiger where agents have to act in a fully coordinate fashion. We can see here how the problem structure is important and influence on the performances of our method.

Those results are good, but are just a simulation. Perhaps the simulator ignored some very rare situations (but

	PBIP		Vector-Valued	
	ADR	Duration	ADR	Duration
Dec-Tiger	1.168	0.54s	-3.471	0.22s
Grid 3x3	1.238	0.99s	2.152	0.41s

Figure 1: Computation time and ADR of a policy.

	PBIP		Vector-Valued	
	Reward	Duration	Reward	Duration
Dec-Tiger	9.31	0.54s	-4.47	0.22s
Grid 3x3	5.21	0.99s	3.68	0.41s

Figure 2: Computation time and cumulated reward of a policy.

not impossible) in which PBIP would be better than 2V. Because of this uncertainty, we chose to artificially build a policy-tree and to compute its value.

5.2.2 Building a policy-tree

Like in the previous part, we have a table showing our results. After solving the problem, we built a policy tree according to our value functions. After building the tree of every agent, we computed a cumulated reward associated to the execution of this joint-policy with an horizon of 10.

In table 2, we can see computation times are the same (we didn't change the resolution process) but values are different. Now, our method is lower than PBIP in terms of cumulated reward even for 3x3 Grid. In those conditions, how much is good or bad our method compared to PBIP? It depends on how we look at it.

Those results are not a big surprise: we are really fast, but not so good in terms of quality (not so bad too). Our objective wasn't to reach a high quality but to break the combinatorial complexity. Now, we can work on ways to improve the quality of our policies (by using little DEC-POMDPs instead of MMDPs for example, see sec.6). So, what about problems with more than 2 agents?

5.2.3 Solving problems with a lot of agents

Because existing DEC-POMDPs solvers can't solve problems with a lot of agents (we usually don't go further than 2 agents), there are no existing benchmarks for those type of problems. So, we took problems usually solved by ND-POMDP and convert them into DEC-POMDPs (with joint transition/observation functions, global reward functions, etc...). Once this translation is done, we have real DEC-POMDP benchmarks.

We solve the DEC-POMDP equivalence of a Fire Fighting problem with 2 to 4 agents and the equivalence of a Sensor Network problem. We wrote those problems in order to have global transitions, observations and reward functions defined on the state of the system (contrary to an ND-POMDP where we have n functions, each one defined on the state of one agent).

The Fire Fighting problem was solved in approximaly 20 seconds. We will not try to compare those results with ND-POMDP ones, because it is not what is important here and the problem description is not the same anymore. The thing to see is the following: there are lots of problems which can not be solved by an ND-POMDP. With our formalism, we can solve them and we saw we remain competitive in terms of quality, with good hopes for improving it. As long as we know, this is the first time that a real DEC-POMDP with so much agents is solved.

5.2.4 Deeper analysis

We can solve problems with a lot of agents without too much difficulties. This point is a real success according to our first objective. Now, we have to increase the quality of the policies we compute. Anyway, there are still some limitations: going from 3 to 4 agents increase the computation time from 1.6s to 20s.

First of all, we searched which part of the solving process takes the most amount of time. Our results are as following:

- Computation times for solving the coordination part almost didn't increase: each sub-problem being very simple, computation times are realy fast.
- Computation times for solving the coordination part increased a little: in the Fire Fighting problem, the number of states depends on the number of agents (there are one house per agent). If only the number of agents had increased, computation times for this part would have stayed the same.
- Computation times for reading the input problem and building the sub-problems increased a lot. This is the hard part of our method. A transition matrix grows exponentially with the number of agents, so does the time for manipulating this matrix.

So, if we want to completely get rid of the combinatorial explosion, we need to work on this specific problem (manipulating the data). First of all, it is important to see our solver is absolutly not optimized. It is wrote in python (for easing the modification), we manipulate strings, etc... The first thing to do will be a rewriting of this solver, in an optimized fashion. This will help us to manipulate larger instances but will not probably be enough. We will discuss in sec.6 several ways to manipulate larger instances.

6. DISCUSSION

Before concluding, we will try to have a global view of our work and analyze its key aspects and its limits.

6.1 Advantages of 2V-DEC-POMDPs

Before everything else, this work is a proof that it is possible to solve bigger instances of DEC-POMDPs than we usually do, by initially solving the individual problems and coordinating the agents in a later step.

Using our formalism offers a fast way to solve big DEC-POMDPs, while keeping the possibility to solve problems where interactions between agents evolve during the execution of the problem (in opposite to ND-POMDPs where the interaction model is fixed). Moreover, even if it is not an optimal solution, tests prove we can have a "good" solution, with good results on classical benchmarks. As long as we know, this work is the first one to solve independently the individual aspect and the coordination aspect while solving all kind of DEC-POMDPs. It permitted us to solve instances that no other solver can solve.

6.2 Key aspects

There are some "key" points in our work which have a big influence on the results. First of all, the choice of the POMDP solver will change a lot of things. Actually, we use the TOP solver, because it is very fast. The problem is it can sometimes compute bad policies which will bring us bad results. The choice of a solver must be made according to what is the most important: quality, time, etc...?

Another key point is the choice of a coefficient for Pb^{ind} and Pb^{coo}. Actually, we manually chose those coefficients and the impact on the final result is very important (sec. 5). We must find a way to compute those coefficients, in order to make the solving process completely automatic.

A last key point is the choice of the operator used for extracting Pb^{coo}'s transition and reward functions from the orginial DEC-POMDP's functions. Actually, we use an average operator but it is probably too naive: with such an operator, functions are too approximative, which can explain the degradation of the overall solution quality. We will probably have to find better operators which will compute functions fitting better the situation.

6.3 Model limits

Even if, while solving the problem, we are not limited by the number of agents anymore, it is still a limit in terms of memory space. A transition (or observation) matrix can become really big when we have a lot of agents. Representing it in memory, and manipulating it for computing the individual and the coordination problems is not trivial. In the tests we made, manipulating data takes up to 2/3 of the total time, and we did not present results with bigger instances. During our tests, we couldn't go as far as we wanted in terms of number of agents, because of this difficulty.

One solution could be to compactly represent the problem, using some "tricks". Such a solution could work for instances "a little bigger", but not for all instances. One idea we had is to directly use our formalism to represent the problem in a distributed way. ND-POMDPs [8] are a solution for representing problems in a distributed way (each agent has its own transition function, observation function, etc... and the reward function is joint on local sets of agents). The problem of ND-POMDPs is their limited representativity: we can not express problems where agents' transitions or observations are linked. With our formalism, we could use the individual part to express the transition and observation functions of one agent, and use the coordination part to link those functions beetween agents in interaction.

Moreover, we use sum and averaging operators to extract individual and coordination problems from the main one. This lead to the loss of informations. Using a model in which the individual and the coordination parts would be explicitly described could then improve our results.

7. FUTURE WORK AND CONCLUSION

There are still points on which we have to work on but 2V-DEC-POMDP is already a promising model.

7.1 Future work

We could work on automatically computing coefficients for Pb^{ind} and Pb^{coo}. We could work too on different criterion to compute ICs in order to find the best one. We also have some ideas to make the interaction problem more optimal and then to increase the quality of the global solution.

The most important work to come (which we are actually working on) is writing an adapted model, in which we could explicitly define the individual and the coordination parts. Such a model could lead us to solve optimally problems with more thant two agents without any restrictions on the number of interactions. Moreover, once we will be able to directly express Pb^{ind} and Pb^{coo}, we should be able to manipulate bigger instances, with a lot of agents.

7.2 Conclusion

2V-DEC-POMDP is a promising model, which gave us good results in the tests we made. We actually have good behaviors, with a small complexity (relative to the complexity of a DEC-POMDP). Obviously, there are still a lot of work, like making the conversion from a DEC-POMDP fully automatic, increasing the quality of the computed policies or expressing really big instances. But, such an approach gives us a new way to solve DEC-POMDPs and opens a possibility to work on real size instances.

8. REFERENCES

[1] C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing memory-bounded controllers for Decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligences (UAI-07)*, Vancouver, BC, Canada, 2007.

[2] C. Amato and S. Zilberstein. Achieving goals in Decentralized POMDPs. In *the Eighth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-09)*, Budapest, Hungary, May 2009.

[3] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov Decision Processes. In *Proceedings of the 16th UAI*, pages 32–37, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[4] A. Beynier and A. Mouaddib. An iterative algorithm for solving constrained decentralized Markov Decision Processes. In *The Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[5] J. S. Dibangoye, A. Mouaddib, and B. Chaib-draa. Point Based Incremental Pruning heuristic for solving finite-horizon DEC-POMDPs. In *8th International AAMAS Conference*, Budapest, Hungary, may 2009.

[6] A. Kumar and S. Zilberstein. Constraint-based dynamic programming for Decentralized POMDPs with structured interactions. In *AAMAS*, pages 561–568, Budapest, Hungary, May 2009.

[7] A. Mouaddib, M. Boussard, and M. Bouzid. Towards a framework for Multi-Objective Multi-Agent Planning. In *AAMAS*, 2007.

[8] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed pomdps: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the Twentieth National Conference on Artificial Intelligence*, 2005.

[9] M. L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. In *John Wiley and Sons, New York, NY*, 1994.

[10] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *IJCAI*, 2007.