Proceedings of the
Seventh Annual Workshop
on
Multiagent Sequential Decision Making Under Uncertainty
(MSDM-2012)

Held in Valencia, Spain
in conjunction with AAMAS

June 5, 2012
Organizing Committee

Prashant Doshi University of Georgia
Akshat Kumar University of Massachusetts Amherst
Jun-young Kwak University of Southern California
Frans Oliehoek Maastricht University
Stefan Witwicki INESC-ID, Instituto Superior Técnico

Program Committee

Christopher Amato MIT
Raphen Becker Google
Daniel Bernstein University of Massachusetts Amherst
Aurélie Beynier University Pierre and Marie Curie (Paris 6)
Alan Carlin University of Massachusetts Amherst
Brahim Chaib-Draa Laval University
Georgios Chalkiadakis Technical University of Crete
François Charpillet INRIA
Ed Durfee University of Michigan
Alessandro Farinelli University of Verona
Alberto Finzi Universita di Napoli
Claudia Goldman GM Advanced Technical Center Israel
Michail Lagoudakis Technical University of Crete
Francisco S. Melo INESC-ID Lisboa
Hala Mostafa BBN Technologies
Abdel-Illah Mouaddib Universit de Caen
Enrique Munoz De Cote INAOE, Mexico
Brenda Ng Lawrence Livermore National Laboratory
Praveen Paruchuri Carnegie Mellon University
David Pynadath University of Southern California
Xia Qu University of Georgia
Zinovi Rabinovich Bar-Ilan University
Anita Raja University of North Carolina at Charlotte
Paul Scerri Carnegie Mellon University
Jiaying Shen SRI International, Inc.
Matthijs Spaan Delft University of Technology
Katia Sycara Carnegie Mellon University
Karl Tuyls Maastricht University
Pradeep Varakantham Singapore Management University
Jianhui Wu Amazon
Makoto Yokoo Kyushu University
Chongjie Zhang University of Massachusetts Amherst
Shlomo Zilberstein University of Massachusetts Amherst
# Table of Contents

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>POMDPs in OpenMarkov and ProbModelXML</td>
<td>1</td>
</tr>
<tr>
<td>Manuel Arias, Francisco Javier Díez, Miguel Ángel Palacios-Alonso, Mar Yebra, and Jorge Fernández</td>
<td></td>
</tr>
<tr>
<td>Solving Finite Horizon Decentralized POMDPs by Distributed Reinforcement Learning</td>
<td>9</td>
</tr>
<tr>
<td>Bikramjit Banerjee, Jeremy Lyle, Landon Kraemer and Rajesh Yellamraju</td>
<td></td>
</tr>
<tr>
<td>Planning Delayed-Response Queries and Transient Policies under Reward Uncertainty</td>
<td>17</td>
</tr>
<tr>
<td>Robert Cohn, Edmund Durfee and Satinder Singh</td>
<td></td>
</tr>
<tr>
<td>Improved Solution of Decentralized MDPs through Heuristic Search</td>
<td>24</td>
</tr>
<tr>
<td>Jilles Dibangoye, Christopher Amato and Arnaud Doniec</td>
<td></td>
</tr>
<tr>
<td>Automated Equilibrium Analysis of Repeated Games with Private Monitoring: A POMDP Approach</td>
<td>32</td>
</tr>
<tr>
<td>Yongjoon Joe, Atsushi Iwasaki, Michihiro Kandori, Ichiro Obara and Makoto Yokoo</td>
<td></td>
</tr>
<tr>
<td>Exploiting Sparse Interactions for Optimizing Communication in Dec-MDPs</td>
<td>40</td>
</tr>
<tr>
<td>Francisco S. Melo, Matthijs Spaan, and Stefan Witwicki</td>
<td></td>
</tr>
<tr>
<td>Tree-based Pruning for Multiagent POMDPs with Delayed Communication</td>
<td>48</td>
</tr>
<tr>
<td>Frans Oliehoek and Matthijs Spaan</td>
<td></td>
</tr>
<tr>
<td>Strategic Behaviour Under Constrained Autonomy</td>
<td>56</td>
</tr>
<tr>
<td>Zinovi Rabinovich</td>
<td></td>
</tr>
<tr>
<td>Prioritized Shaping of Models for Solving DEC-POMDPs</td>
<td>64</td>
</tr>
<tr>
<td>Pradeep Varakantham, William Yeoh, Prasanna Velagapudi, Katia Sycara, and Paul Scerri</td>
<td></td>
</tr>
<tr>
<td>Coordinated Multi-Agent Learning for Decentralized POMDPs</td>
<td>72</td>
</tr>
<tr>
<td>Chongjie Zhang and Victor Lesser</td>
<td></td>
</tr>
</tbody>
</table>
POMDPs in OpenMarkov and ProbModelXML

M. Arias1 F. J. Díez1 M. A. Palacios-Alonso2 M. Yebrα1 J. Fernández1
1Dept. Artificial Intelligence. UNED. Juan del Rosal, 16. 28040 Madrid. Spain
2Computer Science Department. National Institute of Astrophysics, Optics and Electronics.
   Luis Enrique Erro 1. 72840 Tonantzintla. Mexico
3HC Energía. Plaza de la Gesta, 2. 33007 Oviedo. Spain

ABSTRACT

OpenMarkov is an open-source tool for editing and evaluating probabilistic graphical models, such as Bayesian networks, influence diagrams, MDPs, POMDPs, Dec-POMDPs, etc. ProbModelXML is a format for encoding probabilistic graphical models. In this paper we explain how to edit MDPs and POMDPs using OpenMarkov’s graphical user interface, and how these models can be stored in ProbModelXML.

1. INTRODUCTION

Markov decision processes (MDPs) were developed around the mid-20th century as a tool for planning, more specifically, for solving multistage decision problems in which the outcomes are partly random and partly under the control of a decision maker [4]. The main limitation of those models was the assumption that the state of the system is always known with certainty, which is unrealistic in most cases. The relaxation of this assumption led to the emergence of partially observable Markov decision processes (POMDPs) [3], in which the state of the system is not directly observable, but there is a variable that correlates probabilistically with it.

A probabilistic graphical model (PGM) consists of a probability distribution $P$ defined on a set of variables $V$ and a graph $G$ such that each node in the graph represents one of the variables of $V$ and the structure of the graph represents the probabilistic relations in $P$. Roughly speaking, we can say that in general each link in $G$ represents a dependency in $P$ and each link absent in $G$ represents a relation of conditional independency in $P$. The exact relation between $G$ and $P$ depends on the type of PGM, mainly on whether the links in $G$ are directed or undirected. The first types of PGMs were influence diagrams [16] and Bayesian networks [22].

Some types of PGMs have a dynamic version (in this context, “dynamic” is a synonym for “periodic”), in which each node represents a variable (a real-world property) in a particular instant of time, as shown in Figure 1. These dynamic PGMs generalize Markov models developed several decades earlier. Thus, dynamic Bayesian networks [8] generalize Markov chains and hidden Markov models [20]; factored MDPs [5, 6] and factored POMDPs [7] generalize flat (i.e., non-factored) MDPs and POMDPs, respectively.

Factored (PO)MDPs can model efficiently many problems that in practice could not be represented with flat models, and new algorithms developed in the last years are able to solve problems several orders of magnitude bigger than those affordable in the recent past [14, 23, 25].

In practice, the use of PGMs and (PO)MDPs requires two steps: building a model and doing inference on that model. There are many tools for building Bayesian networks and influence diagrams using a graphical user interface (GUI)—see Section 2.2—but to our knowledge only one of them, Netica, claims that it can be used to build (PO)MDPs; however it uses a non-standard representation, and is not open-source (see again Sec. 2.2). Therefore, the usual way of building a (PO)MDP is to generate an ASCII file using a text editor—or an XML editor, in its case—, which makes the process difficult, time-consuming, and prone to errors. In the field of robotics, where the practitioners of (PO)MDPs are usually computer scientists or engineers, this task is a burden but not an obstacle for the use of this type of models. On the contrary, it is extremely rare to find a health professional with the expertise and the patience necessary to build a (PO)MDP using a text editor. For this reason, it might be very helpful to have a software tool with a friendly GUI for building (PO)MDPs, especially in some domains, such as medicine [12]. That tool should be open-source in order to allow researchers to modify and extend it.

We have mentioned in passing the need of a format for storing the models. Several formats have been proposed for PGMs, but only of them is able to encode Bayesian networks, influence diagrams, and factored (PO)MDPs: DNET, which is the native format of Netica. However, to our knowledge this format has never been used to build a (PO)MDP for a real-world application; in Sections 2.2 and 2.3 we discuss the reasons that may explain it.

In this context, we decided to build OpenMarkov, an open-source tool for building and evaluating several types of PGMs, including Bayesian networks, influence diagrams, and several types of Markovian models. We are also developing ProbModelXML, an XML format for encoding those models, with a wide variety of potentials, including tables, trees, ADDs, and canonical models, such as the noisy OR, noisy MAX, noisy AND, etc. [10].

The purpose of presenting our work at the MSDM-2012 workshop is threefold. First, to offer our tools to the com-

1 In this paper we reserve the term “MDP” for fully-observable Markov decision processes, and will write “(PO)MDP” to refer to both MDPs and POMDPs.

2 www.openmarkov.org.
munity of (PO)MDPs practitioners. Second, to receive their feedback: if a significant number of potential users demand a new feature in OpenMarkov and/or in ProbModelXML, we will consider the possibility of adding it. And third, to find researchers interested in integrating in OpenMarkov the algorithms they have already implemented. This would not only be useful for evaluating (PO)MDPs inside OpenMarkov's GUI; it would permit to use OpenMarkov as a workbench for comparing the performance of different algorithms, provided that they use the same basic data structures, to make the comparison meaningful.

The rest of this paper is structured as follows: Section 2 presents background material about dynamic models and reviews some of the software tools and formats for PGMs and (PO)MDPs developed in the past. Section 3 offers an overview of OpenMarkov and Section 4 describes briefly the ProbModelXML format. Section 5 explains how to build (PO)MDPs using OpenMarkov's GUI and how multiagent models are stored in ProbModelXML. Finally, Section 7 summarizes the conclusions and discusses some lines for future research.

2. STATE OF THE ART

2.1 Dynamic models

As mentioned in the introduction, in a dynamic model, some variables are indexed by time: \( t \in \{0, \ldots, h\} \), where \( h \) is the horizon of the model, which can be infinite.\(^3\) If there is a variable \( X^t \) for a certain \( t \), then there is also a variable \( X^{t'} \) for each \( t' \in \{0, \ldots, h\} \)—see Figure 1, where the number between brackets denotes the index \( t \). The subgraph that contains all the nodes \( X^t \) having the same temporal index \( t \) and the links between them is known as the \( t \)-th time slice. A constraint of dynamic models is that they can not have links to the past, i.e., of the form \( X^t \rightarrow Y^{t'} \) with \( t > t' \). If there is a \( k \in \mathbb{N}^+ \) such that every link \( X^t \rightarrow Y^{t'} \) satisfies that \( t' - t \leq k \) (which means that all the parents of a node \( Y^{t'} \) are in the \( t' \)-th slice or in the previous \( k \) slices), and this property is not satisfied for any smaller value of \( k \), we say that the model is of \( k \)-th order. In practice, it is usual to work with first-order models (\( k = 1 \)).  

A node \( X \) is stationary after \( k \) if it has the same neighbors and the same potentials (probabilities and utilities) after the \( k \)-th slice. Put formally, for each \( t > k \),

- there is a link between \( X^t \) and \( Y^{t'} \) if and only if there is a link of the same type between \( X^k \) and \( Y^k \);
- if there is a conditional probability \( P(x^t | pa(X^t)) \), then \( P(x^t | pa(X^k)) = P(x^k | pa(X^k)) \);

\(^3\)In standard MDPs and POMDPs all the variables are indexed by time. However, we admit the possibility of atemporal variables, whose value does not change with time—see [2].
• if there is a utility function $U^k(pa(U^k))$, then 
$U^t(pa(U^t)) = U^k(pa(U^k))$;

and some of these properties are not fulfilled for any $k' < k$.

A dynamic model is stationary after $k$ if all its nodes are stationary after $k$ and there is no other $k' < k$ that makes the model stationary. Such a model admits a compact representation that contains only the first $k$ slices. Stationary first-order models, which constitute the most common case, can be represented by only the first two time slices. In particular, MDPs and POMDPs are almost always stationary and first-order, and usually have infinite horizons, and for these reasons their standard representation is a two-slice model, as in Figure 1.

2.2 Software tools for PGMs and (PO)MDPs

Several computer packages for PGMs have been developed in the last years; nearly all of them are contained in Kevin Murphy’s list. Some of them have GUIs for building PGMs, but all of them, except OpenMarkov, limit themselves to Bayesian networks and influence diagrams, i.e., they offer no support for (PO)MDPs.

Conversely, there exist several open-source tools for (PO)MDPs, such as Anthony Cassandra’s pomdp-solve; Jesse Hoey’s SPUDD [14]10, Pascal Poupart’s Symbolic Perseus [23]11, and the APPL tool developed at the National University of Singapore8, but none of them has a GUI for editing and evaluating this type of models.

There is only one tool for Bayesian networks and influence diagrams that claims to offer support for POMDPs: Netica9, but it is not open-source; it is a commercial program developed by a private company, Norsys. Additionally, the ability to build (PO)MDPs is claimed only once: in the introduction of the specification of Netica’s DNET format (see below). In any case, Netica does not represent POMDPs in the standard way, i.e., in the form of a two-slice model (see Fig. 1), but by means of labeled links, such that each variable is represented by a single node in the model (instead of having a node $X^t$ for each time slice $t$) and a link from $X$ to $Y$ with a label $n$ in the compact model represents a link from $X^n$ to $Y^{t+n}$ for every $t$ in the expanded model. Another limitation of Netica and DNET is that they cannot represent trees nor ADDs, which are almost indispensable when building real-world applications. Finally, the most important limitation of Netica with respect to (PO)MDPs is that apparently it offers no specific algorithm for evaluating this type of models; the only way of evaluating them consists of expanding the network up to a certain horizon and obtaining its policies as if it were an influence diagram, but in general this approach is unfeasible except for very small problems and very short horizons.

2.3 Formats for PGMs and (PO)MDPs

Several formats have been proposed for storing PGMs. In general, each format has been developed for a particular software tool: DNET was developed for Netica, DSC

10sites.poli.usp.br/pfabio.cozman/Research/InterchangeFormat.
13www.ia.uned.es/~elvira/.

and XBN for Microsoft’s MSBNx, XDSL for Smile and GE-NIE, etc. (The references for these formats and tools can be found in [2].) The only format intended to become a kind of standard is XML BIF12, proposed by Fabio Cozman with suggestions from a few other people. Unfortunately, this format is restricted to the representation of Bayesian networks with finite-state variables.

Similarly, several formats have been developed for (PO)-MDPs, each one designed for a particular software tool: there is format for Cassandra’s pomdp-solve, another one for Hoey’s SPUDD, the PomdpXML format for APPL... (Again, the references for these formats and tools can be found in [2].) SPUDD’s format is able to encode factored POMDPs, but Cassandra’s format and PomdpXML can only represent flat POMDPs. A drawback of SPUDD’s format is that it is not XML; it uses a Lisp-like syntax with some peculiarities that make it difficult to build a parser.

Two additional formats were proposed to encode the problems proposed at the Probabilistic Planning Track of the International Planning Competition (IPC). The Probabilistic Planning Domain Definition Language (PPDDL),11 used at the 4th and 5th IPC in 2004 and 2006 respectively, was able to encode factored MDPs with finite-state variables. The Relational Dynamic Influence Diagram Language (RDDL), used at the 7th IPC in 2011, was able to represent relational (PO)MDPs with both finite-state and continuous variables.12 These formats were not intended to encode non-temporal Bayesian networks and influence diagrams, but they have enough expressive power to represent them as well.

3. OPENMARKOV

This project started in 2003 at the Department of Artificial Intelligence of the Universidad Nacional de Educación a Distancia (UNED), in Madrid, Spain. Its original name was Carmen [1], but in 2010 it was renamed as OpenMarkov. We departed from our experience in the construction of Elvira13,13 an open-source tool began in 1997 as a join project of several Spanish universities, but everything in the new program was redesigned and the code of OpenMarkov was built from scratch.

Programming language

The development language for OpenMarkov is Java, mainly in order to allow it to run on different platforms. Since the beginning we used version 1.5, which introduced new syntactical features, such as typed collections (for instance, ArrayList<CertainClass>) and enhanced loops, which significantly facilitate the iteration on lists.

Data structures

A probabilistic network is represented in OpenMarkov as a generic data structure consisting mainly of a graph, a set of variables, and a set of potentials. Each type of network is defined by a set of constraints, as shown in Table 2. It is possible to implement new types of networks easily by combining the existing constraints and, if necessary, by adding new ones.
OpenMarkov accepts three types of variables: finite-states, numerical, and discretized, and two types of links: directed and undirected. It also has several types of potentials: uniform (mainly used to assign a default potential to each node in networks that only contain directed links), table (the most frequently used potential), delta (i.e., Kronecker delta for finite-state variables and Dirac delta for numeric variables), tree/ADD, several canonical models (OR, AND, MAX, MIN, etc.), sum, product, linear combination, conditional Gaussian, exponential, mixture of exponentials, and logistic regression. There are also three potentials for dynamic models: same as previous, cycle length shift, and Weibull distribution. In the future, we will add other potentials, such as mixtures of polynomials. (See [2] for a detailed description of the types of potentials and the network types, with bibliographical references.)

OpenMarkov is able to represent several types of networks, such as Bayesian networks, Markov networks, influence diagrams, LIMIDs, and decision analysis networks (DANs), as well as several types of temporal models: dynamic Bayesian networks, simple Markov models, MDPs, POMDPs, Dec-POMDPs, and dynamic LIMIDs. Currently OpenMarkov can only evaluate Bayesian networks, influence diagrams, and simple Markov models, but it can be used to build several types of models, such as (PO)MDPs, that might be read by any other tool able to parse the ProbModelXML format.

### Algorithms

We have implemented in OpenMarkov the most usual algorithms for Bayesian networks and influence diagrams, such as variable elimination [27, 9] and clique tree propagation [19, 17]. We also programmed some algorithms for evaluating MDPs, such as value iteration [4], policy iteration [15], and modified policy iteration [24], but they are not integrated in the last version of OpenMarkov, which underwent a major refactoring at the end of 2011. In OpenMarkov it is possible to learn Bayesian networks interactively from databases by applying the two most popular methods: search-and-score, with several metrics, and the PC algorithm [21].

### Graphical user interface

The graphical user interface (GUI) is very similar to those of other software tools for PGMs, especially to that of Elvira. It has two main working modes: edition and inference. It has been designed for internationalization; currently messages can be displayed in English or Spanish, and other languages can be easily added using Java’s facilities.

### Testing

In order to guarantee as much as possible the robustness of the tool, we have built a test suite for each class in OpenMarkov with JUnit. After introducing a modification in OpenMarkov, we run the battery of tests in order to detect possible bugs in the program.

### Code hosting and version control system

As a version control tool, we initially chose Subversion, installed on a local server, but in October 2011 we reorganized the code into a set of Maven subprojects and migrated the code to Bitbucket, a code-hosting system similar to SourceForge, JavaSource, or Google Code. It offers two control version systems: Mercurial and Git. OpenMarkov uses Mercurial. It also offers wikis, issue trackers, and other facilities. We use Mercurial to store in Bitbucket a working copy of OpenMarkov’s Java source code. There is also a wiki and an issue tracker for each subproject.

Using Maven, OpenMarkov is deployed on a local Nexus repository, which contains three types of files: Java source code of snapshots and releases (in contrast with Bitbucket,

---

14Three of these models have been proposed originally by our group: DANs, simple Markov models, and dynamic LIMIDs.


16[nexus.openmarkov.org](http://nexus.openmarkov.org).
Table 2: Constraints used in OpenMarkov. The letter in each cell indicates whether a constraint is associated with a network type: Y = yes, N = no, O = optionally. A dash means that a constraint does not make sense because of the presence of another constraint—see [2]. Each constraint has a default behavior; a bold letter in this table means that the default behavior has been overridden for a particular type of network.

which stores the whole version history, including the most recent files not yet deployed), compiled Java binaries, and Javadoc HTML files. For further details, see OpenMarkov’s wiki.\textsuperscript{17}

4. PROBMODELXML

4.1 Motivation

As mentioned in the introduction, we decided to develop a new format for PGMs aimed at becoming a common language for several research groups and several software tools. The first requisite was that the format should accommodate different types of models (Bayesian networks, Markov networks, influence diagrams, POMDPs, etc.), three types of variables (finite-states, numerical, and discretized), a variety of potentials (table, tree/ADD, canonical models...), and a wide range of properties. However, given that it is impossible to foresee from the beginning all the needs that will emerge in the future, the format should be extensible, i.e., it should be able to represent new types of models and new properties without changing the specification of the format. Second, the syntax and the semantics of the format should be clearly documented, in order to avoid ambiguities and misinterpretations. Third, the syntax of the format should be based on the Extensible Markup Language (XML) specification produced by the World Wide Web Consortium (W3C),\textsuperscript{18} mainly because XML is much easier to parse than other types of syntaxes: there exist many tools for parsing XML from several programming languages (Java, C++, etc.), with the corresponding utilities for writing XML files from those languages, as well as other tools for specifying XML formats and for validating them (DTD, XML Schema, Relax NG, ISO DSDL...).

4.2 Specification of models

The skeleton of a ProbModelXML file is:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<ProbModelXML formatVersion = "string">
  <ProbNet type="enumNetworkType" />\textsuperscript{0.1}
  <InferenceOptions />\textsuperscript{0.1}
  <Policies />\textsuperscript{0.1}
  <Evidence />\textsuperscript{0.1}
</ProbModelXML>
```

and the extension we propose for those files is .pgmx. The skeleton of a probabilistic network is:

\textsuperscript{17}wiki.openmarkov.org.

\textsuperscript{18}www.w3.org/XML.
probnets type=enumNetworkType >
<AdditionalConstraints />0..1
<Comment />0..1
<DecisionCriteria />0..1
<Agents />0..1
<Language />0..1
<AdditionalProperties />0..1
<Variables />0..1
<Links />0..1
<br Potentials />0..1
</ProbNet>

The tag <AdditionalProperties> is used to extend the ProbModelXML format by representing other properties not explicit in this format. An additional property can appear in the context of a ProbNet, an Evidence Case, a Criterion (for multicriteria decision making), an Agent (in multi-agent models), a Variable, a State of a variable, a Potential and a Policy. In all the cases, the skeleton for encoding additional properties is as follows:

<AdditionalProperties>
<Property name="string" value="string" />1..n
</AdditionalProperties>

A complete specification of the ProbModelXML format, with detailed explanations and many examples can be found in [2].

5. (PO)MDPS IN OPENMARKOV AND PROBMODELXML

5.1 Editing (PO)MDPs with OpenMarkov’s GUI

The code of OpenMarkov’s GUI does not depend directly on the type of network, but on the constraints. Therefore it is very easy to extend the GUI to deal with new types of networks: it suffices to modify the aspects that depend on the new constraints, if any. Given that the constraints that define an influence diagram are almost the same as those for a POMDPs or a Dec-POMDP (cf. Table 2), the edition of these three types of models is very similar. Figure 1 shows a factored POMDP that models a robot designed for serving coffee to a single user. This example, proposed by Jesse Hoey and encoded in SPUDD format [14], is included with the source code of Pascal Poupart’s software Symbolic Perseus.19

As that figure shows, in OpenMarkov, rectangles represent decisions (actions), rounded rectangles represent chance variables, and hexagons represent utility nodes, i.e., costs and rewards. In this figure, the number in square brackets indicates the time slice to which a node belongs. A link from a node in a time slice to a node in the same slice is said to be intratemporal; a link from a time slice to the next one is said to be intertemporal. Links from the second time slice to the first one are not allowed, because it would contradict our notion of causality. A link $X \rightarrow D$, where $D$ is a decision, is called an informational link and means that the value that variable $X$ has taken is observed when making decision $D$.

In this example, there are two observable variables, Charge observed, that gives information about the state of the battery, and Wet pavement, which provides indirect evidence on whether it is raining or not.20

As in the case of Bayesian networks and influence diagrams, each chance node in the first time slice has an associated probability potential, which in this example is a prior probability, because no chance node in that slice has parents, but in other examples there may be intratemporal links in the first slice. If a node has parents, either in the same time slice or in the previous one, its associated probability is conditioned on that parent. For instance, the probability of Wet pavement in this example is conditioned on the current value of Raining outside and on the Action chosen in the previous time slice; when Action[0] was different from measurewet, the probabilities of the two values of this variable, namely wet and dry, are the same, and therefore this “observation” does not affect the probability of Raining outside.

As in the case of influence diagrams, each utility node has an associated utility potential that depends on the parent of that node in the graph. In this example there is one reward that takes a positive value when the chance variable User has coffee takes the value yes, and four costs, that depend on the Action chosen. In other examples, a utility (node) may depend on both chance variables and actions.

The edition of a potential in OpenMarkov depends on the type of potential. If the potential is a table, its edition is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a table, its edition is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a table, its edition is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a table, its edition is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a table, its edition is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tree or an ADD, the window for editing it is very similar to that of other software tools for PGMs. If the potential is a tre
Multiagent models can be edited in OpenMarkov by first defining a list of agents and then associating each decision to an agent by means of a push-down widget.

6. RELATED WORK

ProbModelXML is similar to the RDDL format mentioned in Section 2.3, which can represent relational models and logic relations, while ProbModelXML cannot. On the other hand the latter can represent more types of networks and potentials, including Dec-POMDPs. Another advantage of ProbModelXML is its XML syntax. It would be interesting to extend ProbModelXML to cover all the features included in RDDL.

The RDDL simulator\(^\text{21}\) is a program that can show graphically the models written in this format, but it cannot edit them. In contrast, it can evaluate those models, while OpenMarkov cannot.

7. CONCLUSIONS AND FUTURE WORK

OpenMarkov is an open-source tool for editing and evaluating probabilistic graphical models. If permits to edit several types of probabilistic networks, including MDPs, POMDPs and Dec-POMDPs, and it is easy to extend it to deal with other types of networks. It also implements several types of potentials, including tables, trees, and ADDs. The native format for OpenMarkov is ProbModelXML. Given that the format and the tool have been developed in parallel, the types of variables, potentials, and networks are essentially the same in both of them, even though some of the features included in the specification of the format have not yet been implemented in OpenMarkov. This is one of the first tasks in the agenda of OpenMarkov's developers. Other tasks are to continue debugging OpenMarkov, mainly its GUI, which is the most difficult component to debug, and to extend its documentation. Later, we will consider the possibility of integrating in OpenMarkov some of the existing algorithms for evaluating POMDPs; we have received some nice offers from researchers of other groups, but currently we cannot address that issue given the limited amount of resources in our group.
Acknowledgments
We thank Jesse Hoey for providing us the coffee robot example and several useful comments. We thank one of the anonymous reviewers for making us aware of the existence of PPDDL and RDDL. We are grateful to all the people who have collaborated in the OpenMarkov project.

This work has been supported by grants TIN-2006-11152 and TIN2009-09158, of the Spanish Ministry of Science and Technology, and by FONCICYT grant 85195.

8. REFERENCES
ABSTRACT

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a powerful modeling technique for realistic multi-agent coordination problems under uncertainty. Prevalent solution techniques are centralized and assume prior knowledge of the model. We propose a distributed reinforcement learning approach, where agents take turns to learn best responses to each other’s policies. This promotes decentralization of the policy computation problem, and relaxes reliance on the full knowledge of the problem parameters. We derive the relation between the sample complexity of best response learning and error tolerance. Our key contribution is to show that even the “per-leaf” sample complexity could grow exponentially with the problem horizon. We show empirically that even if the sample requirement is set lower than what theory demands, our learning approach can produce (near) optimal policies in some benchmark Dec-POMDP problems. We also propose a slight modification that empirically appears to significantly reduce the learning time with relatively little impact on the quality of learned policies.

1. INTRODUCTION

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a powerful modeling technique for realistic multi-agent coordination and decision making problems under uncertainty. Because solving Dec-POMDPs is NEXP-complete [4], exact solution techniques for finite horizon problems require significant time and memory resources [20, 15, 18]. However, solution techniques for Dec-POMDPs (exact or approximate) suffer from less acknowledged limitations as well: that most of them are centralized and assume prior knowledge of the model. That is, a single program computes the optimal joint policy, with the full knowledge of the problem parameters. While these techniques have had success in benchmark problems with comprehensively defined domain parameters, such strict definitions may be difficult and tedious in many real-world problems. In these cases, the problem parameters may need to be first estimated from experience and then exact/approximate solvers may be applied. However, this problem of model estimation can be complex in POMDPs because states are unobservable [6], and additionally in Dec-POMDPs agents are incapable of observing each other’s actions and observations. Not surprisingly, model estimation is largely ignored in the Dec-POMDP literature. Furthermore, the task of computing optimal policies in Dec-POMDPs has seldom been decentralized in any meaningful way (e.g., see [8].)

In this paper, we propose a distributed reinforcement learning approach to solving finite horizon Dec-POMDPs. When agents learn their own policies, not only is the task of policy computation distributed, but also the problem parameters do not need to be known a priori. In lieu of the knowledge of problem parameters, access to a simulator, or simply the ability to draw samples from unknown distributions would be sufficient. Effectively, estimation of the problem parameters is built into the learning algorithm. Policy learning in finite horizon tasks is justified due to the same reasons as finite horizon reinforcement learning, viz., that agents can learn policies in offline simulations before applying them in the real domain. Furthermore, unlike many exact and approximate solution approaches for Dec-POMDPs, the memory usage of a learning approach is not much larger than the size of a single policy per agent at any time, which makes it relatively more memory efficient. Thus we posit distributed reinforcement learning as a more practical alternative to the traditional Dec-POMDP solvers.

Reinforcement learning has been applied in infinite horizon POMDPs in both model based [6, 12, 17] and model free [13] ways. Model-based methods learn a model (e.g., hidden Markov models or utile suffix memories for POMDPs) of the environment first and then compute a policy based on the learned model, while model-free methods learn a policy directly. Model learning can be more complex in Dec-POMDPs because the actions and the observations of the other agents are unobservable. We use a semi-model based approach, where we do not attempt to estimate the Dec-POMDP parameters owing to their hidden parts, but instead learn intermediate functions that capture the visi-
ble parts of the dynamics (see equations 4, 5 given later) via Monte Carlo estimation, and compute a policy based on these functions.

Zhang and Lesser [21] recently applied reinforcement learning to a variant of the finite horizon Dec-POMDP problem, where agents are organized in a network, and agents’ influence on each other are limited to cliques. This factored structure of the domain is exploited to solve such problems more scalably than regular Dec-POMDPs. Zhang and Lesser also exploited the known communication structure to coordinate the sub-teams via distributed contract optimization, and produced a more efficient learning-based alternative to the regular solvers. While our goal is similar and we also consider finite horizon problems, we focus on less structured and unfactored Dec-POMDPs that are inherently less scalable. As in our work, [21] does not guarantee optimality unless the agent coordination graph is acyclic.

Our initial experiments with concurrent independent reinforcement learning [7] in benchmark Dec-POMDP problems have yielded unsatisfactory results, some of which are included in the experiments section. In this paper we propose a non-concurrent independent learning approach, where agents take turn in learning best response policies to each other via a semi-model based Monte Carlo algorithm, but no agent explicitly attempts to model the other agent. We show theoretically that Monte Carlo reinforcement learning for best response learning can converge to local optima. Finally, we propose a slight modification of our algorithm, that empirically appears to significantly reduce the learning time with relatively little impact on the quality of the learned policies. A shorter version of this paper appears in [3].

2. DECENTRALIZED POMDP

The Decentralized POMDP (Dec-POMDP) formalism is defined as a tuple \((n, S, A, R, \Omega, O)\), where:

- \(n\) is the number of agents playing the game.
- \(S\) is a finite set of (unobservable) environment states.
- \(A = \times_i A_i\) is a set of joint actions, where \(A_i\) is the set of individual actions that agent \(i\) can perform.
- \(P(s' | s, \bar{a})\) gives the probability of transitioning to state \(s' \in S\) when joint action \(\bar{a} \in A\) is taken in state \(s \in S\).
- \(R : S \times A \rightarrow \mathbb{R}\), where \(R(s, \bar{a})\) gives the immediate reward the agents receive upon executing action \(\bar{a} \in A\) in state \(s \in S\).
- \(\Omega = \times_i \Omega_i\) is the set of joint observations, where \(\Omega_i\) is the finite set of individual observations that agent \(i\) can receive from the environment.

3. REINFORCEMENT LEARNING

Reinforcement learning (RL) problems are modeled as Markov Decision Processes or MDPs [19]. An MDP is given by the tuple \((S, A, R, \Omega)\), where \(S\) is the set of environmental states that an agent can be in at any given time, \(A\) is the set of actions it can choose from, \(R : S \times A \rightarrow \mathbb{R}\) is the reward function, i.e., \(R(s, a)\) specifies the reward from the environment that the agent gets for executing action \(a \in A\) in state \(s \in S\); \(\Omega : S \times A \times S \rightarrow [0, 1]\) is the transition probability function specifying the probability of the next state in the Markov chain consequential to the agent’s selection of an action in a state. In finite horizon problems, the agent’s goal is to learn a non-stationary policy \(\pi : S \times t \rightarrow A\) that maximizes the sum of current and future rewards from any state \(s\), given by,

\[
V^\pi(s, t) = \mathbb{E}_P \left[ \sum_{t=0}^{T-1} R(s^0, \pi(s^0), t) + R(s^1, \pi(s^1, t + 1)) + \ldots + R(s^{T-t}, \pi(s^{T-t}, T)) \right]
\]

where \(s^0, s^1, \ldots, s^{T-t}\) are successive samplings from the distribution \(P\) following the Markov chain with policy \(\pi\).

Reinforcement learning algorithms often evaluate an action-quality value function \(Q\) given by

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

This quality value stands for the sum of rewards obtained when the agent starts from state \(s\) at step \(t\), executes action \(a\), and follows the optimal policy thereafter. Action quality functions are preferred over value functions, since the optimal policy can be calculated more easily from the former. Learning algorithms can be model based or model free. Model based methods explicitly estimate \(R(s, a)\) and \(P(s, a, s')\) functions, and hence estimate \(Q(s, a, t)\). Model free methods directly learn \(Q(s, a, t)\), often by online dynamic programming, e.g., Q-learning. In this paper, we use (semi-) model based learning for Dec-POMDPs which makes for easier analysis of sample complexity, thus establishing a baseline for RL in Dec-POMDPs. Model based reinforcement learning algorithms have been analyzed in many domains before, but to the best of our knowledge such analysis have not been performed for decentralized POMDPs, where partial observability of the learner’s environment is compounded by the unobservability of the other agents’ observations and actions.

4. RL FOR DEC-POMDPS
Solution techniques for Dec-POMDPs have been mostly centralized [20, 15, 18], in that a single program computes the optimal joint policy, with the full knowledge of the problem parameters, viz., P, R, O. While these techniques have had success in benchmark problems with comprehensively defined P, R, O, such strict definitions may be difficult and tedious in real-world problems. In this paper we address this issue by applying reinforcement learning to the policy computation problem. The main distinguishing characteristics of our approach are

- Instead of a single program computing the optimal joint policy, each agent learns its own policy. In this paper agents learn distributedly, but not concurrently. That is, they share the task of policy learning, by only learning their own policies, but do not update policies concurrently. Concurrent learning will effectively parallelize Dec-POMDP solution, but it is also challenging due to potential oscillation. Our experiments show that concurrent learning is not as efficient as the proposed distributed learning approach. We leave the improvement of concurrent learning in Dec-POMDPs as a future avenue.

- Instead of using knowledge of P, R, O, agents learn on the basis of sampling these unknown functions. This allows our approach to be readily applicable in tasks where these parameters are unknown, or hard to compute. However, for evaluation purposes, we still consider well-defined benchmark problems in this paper.

- Most Dec-POMDP solvers maintain many policies in memory at any time, partly or wholly. Even memory bounded techniques [16] maintain multiple policies, although of a bounded total size. Instead, a learner only needs to effectively maintain sufficient information in memory to construct one policy. However, for finite horizon problems, this policy has a size exponential in T.

Although the agents are unaware of P, R, O, we assume that the agents know the size of the problem, i.e., |A|, |S|, |O|, the maximum magnitude over all rewards, Rmax, and that they are capable of signalling to each other so that no two agents are learning at the same time. With > 2 agents, the order of learning phases must also be fixed by prior agreement.

Since states are not visible, a reinforcement learning agent can use the policy representation of finite horizon Dec-POMDPs, and learn a mapping from histories of its past actions and observations to actions [21]. For simplicity of notation, we assume two agents only, and identical action and observation sets for both agents. Given the policy of the other agent, π, the quality of a learner’s action a at a given level-t history h_t is given by

\[
Q^*_t(h_t, a|\pi) = R^*_t(h_t, a|\pi) + \max_{\omega} \sum_{s, h_{t+1}} H^*_t(h_t, a, h_{t+1}|\pi) \cdot \omega
\]

where h_{t+1} is a level-t + 1 history produced by the concatenation of h_t and (a, \omega), i.e., h_{t+1} = (h_t, a, \omega). The best response policy of the learner, π_t, to the other agent’s policy π is given by

\[
\pi_t(h_t) = \arg \max_a Q^*_t(h_t, a|\pi).
\]

The functions R^*_t and H^*_t represent level-t reward and history transition functions for the learner, given by

\[
R^*_t(h_t, a|\pi) = \sum_{s, h_{t-1}} P(s|h_t, h_{t-1})P(h_{t-1}|h_t, \pi)R(s, \bar{a})
\]

\[
H^*_t(h_t, a, h_{t+1}|\pi) = \sum_{s, s', h_{t-1}} P(s|h_t, h_{t-1})P(h_{t-1}|h_t, \pi) \cdot P(s'|s, \bar{a})\sum_{\omega_{t-1}} O(\bar{\omega}|s', \bar{a})
\]

where h_{t-1} is the history of action-observations encountered by the other agent, \bar{\omega} = (\omega, \omega_{t-1} and \bar{a} = (a, \pi(h_{t-1})) are the joint observation and action respectively. A learner agent is unaware of every factor on the right hand sides of equations 4, 5, and must estimate R^* and H^* solely from its own experience of executing actions and receiving observations and rewards. Note that while the learners do use rewards as “shared observations” in order to estimate the quality values while learning, they do not rely on such observations when executing the policy (i.e., equation 3). Therefore, the crux of a Dec-POMDP is preserved, making our approach an effective alternative to classical Dec-POMDP solvers.

For brevity, we call the following expression β.

\[
\beta = |A| \left( \left(\frac{|A||\Omega|}{|A||\Omega| - 1} \right)^T - 1 \right).
\]

β gives the maximum number of (h, a) pairs of all lengths that a learner may encounter. We now give the definition of a key parameter that appears in the complexity expression of our algorithm.

**Definition 1.** The minimum reachability over all feasible states at level t for a fixed policy of the other agent, π, is given by

\[
\rho_{t, \pi} = \min_{s, h_t, h_{t-1}|\pi} P(s|h_t, h_{t-1})P(h_{t-1}|h_t, \pi)
\]

Feasibility excludes unreachable states, and therefore ensures that always ρ_{t, \pi} > 0.

When a learner takes action a at history h_t (with the other agent executing π(h_{t-1})) and the resulting joint observation is \bar{\omega} = (\omega, \omega_{t-1}), then reachability can be propagated as

\[
P(s|h_t, h_{t-1})P(h_{t-1}|h_t, \pi)P(s'|s, \bar{a})O(\bar{\omega}|s', \bar{a})
\]

where h_{t+1} = (h_t, a, \omega) and h'_{t+1} = (h_{t+1}, \pi(h_{t-1}), \omega_{t-1}). Clearly, the minimum reachability at level t + 1 is

\[
\rho_{t+1, \pi} \leq \rho_{t, \pi}
\]

forming a monotonically decreasing sequence with increasing t. Therefore, we refer to the minimum reachability over all steps, \rho_{T-1, \pi}, simply as ρ dropping both subscripts when π is clear from the context.

**4.1 The Algorithm: MCQ-ALT**

In this paper we present an approach where agents take turn to learn best response to each others’ policies, using an R-Max [5] like approach to learn the best response Q-values. The main algorithm for any learning agent – called Monte Carlo Q Alternating, or MCQ-ALT – is shown in Algorithm 1, along with its subroutines in Algorithms 2–5. The learner records immediate rewards and history transitions at every
Algorithm 1 MCQ-Alt(N)
1: repeat
2: \( h \leftarrow \emptyset \)
3: \( a \leftarrow \text{SelectAction}(h) \)
4: Execute \( a \) and receive \( r, \omega \)
5: for \( t = 1 \ldots T - 1 \) do
6: \( b \leftarrow \text{Step}(h, a, \omega, r) \)
7: \( h \leftarrow (h, a, \omega) \)
8: Execute \( b \) and receive \( r, \omega \)
9: \( a \leftarrow b \)
10: end for
11: \text{EndEpisode}(h, N)
12: until \( \text{Known}(\emptyset) = \text{True} \)

Algorithm 2 SelectAction(h)
1: if \( \text{Known}(h) = \text{True} \) then
2: \( a \leftarrow \arg \max_{a \in A} Q(h, b) \)
3: else
4: \( a \leftarrow \text{min}_b \text{frequency}(h, b) \)
5: end if
6: \( \text{frequency}(h, a) \leftarrow \text{frequency}(h, a) + 1 \)
7: Return \( a \)

history encountered, providing samples of \( R^* \) and \( H^* \) given in equations 4, 5 respectively. These samples are incorporated into running averages to maintain estimates \( \hat{R} \) and \( \hat{H} \) respectively. For histories of length \( T - 1 \) (i.e., full length), \( h_{T-1} \), if the pair \((h_{T-1}, a)\) has been encountered

\[
N = \max(|S|^2|\Omega|^{T-1} A/\rho, \left( \frac{4R_{\max}T|S|^2|\Omega|^{T+1}}{\alpha^2} \right) \ln(16|S|^2|\Omega|^T/\beta/\delta)).
\]

times, then the learner sets \( Q_{T-1}(h_{T-1}, a) \) to the average of the immediate rewards received (i.e., \( \hat{R}_{T-1}(h_{T-1}, a) \)), via Algorithms 4, 5. It then marks \( (h_{T-1}, a) \) as “Known”. If \((h_{T-1}, a)\) is “Known” for every \( a \), then \( h_{T-1} \) is marked “Known”. For an intermediate length history, \( h_t \), if every history, \( h_{t+1} \), produced by concatenating \( h_t \) with \((a, \omega)\) for all combinations of action-observations encountered is “Known”, then \( h_t \) is marked “Known”. For a “Known” intermediate history \( h_t \), \( Q_t(h_t, a) \) is updated for every \( a \) as

\[
Q_t(h_t, a) = \hat{R}_t(h_t, a) + \sum_{h'} H_t(h_t, a, h') \max_b Q_{t+1}(h', b).
\]

The learner’s exploration strategy is shown in Algorithm 2.
For a “Known” history, action selection is greedy, i.e., the Q-maximizing action is selected. For a history that is not yet marked “Known”, the least frequently taken action (ties broken randomly) is executed. The learner freezes its current policy when the empty history is marked “Known”, and signals to the other agent to start learning, while it executes its current policy without exploration. MCQ-Alt learns best response directly without modeling the other agents in the environment, and only modeling the visible parts of the environment’s dynamics.

The infimum reachability, \( \rho \) used in equation 7, may not be known in many problems. Since it decreases geometrically with increasing \( T \), it may even be hard to determine whether \( |S|^2|\Omega|^{T-1} \) dominates \( 4/\rho \). In such cases, it may be possible to ignore it, but this may be inadequate for higher \( T \). For

Algorithm 3 Step(h, a, \omega, r)
1: \( h' \leftarrow (h, a, \omega) \)
2: \( \hat{H}(h, a, h') \leftarrow \hat{H}(h, a, h') + 1 \)
3: \( \hat{R}(h, a) \leftarrow \hat{R}(h, a) + r \)
4: \( \text{Remaining}(h) \leftarrow \text{Remaining}(h) \cup \{(a, \omega)\} \)
5: Return \( \text{SelectAction}(h') \)

Algorithm 4 EndEpisode(h, N)
1: if \( \text{frequency}(h, a) > N, \forall a \in A \) then
2: \( \text{Known}(h) \leftarrow \text{True} \)
3: \( \text{Qupdate}(h) \)
4: end if
5: while \( h \neq \emptyset \) do
6: Let \( h = (h', a, \omega) \)
7: \( \text{Remaining}(h') \leftarrow \text{Remaining}(h') \setminus \{(a, \omega)\} \)
8: if \( \text{Remaining}(h') = \emptyset \) then
9: \( \text{Known}(h') \leftarrow \text{True} \)
10: \( \text{Qupdate}(h') \)
11: else
12: break
13: end if
14: \( h \leftarrow h' \)
15: end while

the domains and horizons used for experiments in this paper, \( \rho \) does not appear to be a dominating factor, so our analysis focuses on the dependence on \( T \) instead.

An important feature of MCQ-Alt is its well-defined stopping criterion, viz., when the empty history becomes “Known”, which is controlled by a single parameter, \( N \). In contrast, Q-learning is controlled by multiple parameters, and its stopping criterion can be affected by oscillation or non-convergence.

Note that in learning the best response, a learner attempts to cover every \((h_t, a)\) encountered equally well, to guarantee arbitrarily small errors in the value function. However, there are at least two reasons why the value function may not need to be accurate to an arbitrary degree: (1) policies usually converge long before value functions, which we verify in this paper experimentally, and (2) some less likely paths may have little impact on the value function and \( N \) could be lowered for these paths; we address this next.

4.2 Adjustment for Rare Histories

Figure 1 shows a learner’s entire possible experience tree.
Algorithm 5 $Qupdate(h)$

1: for $a \in A$ do
2: $Q(h, a) \leftarrow \hat{R}(h, a) / \text{frequency}(h, a)$
3: if $h$ is not full-length history then
4:   $Q(h, a) \leftarrow \frac{1}{N} \sum_{\omega|h=(h, a, \omega)} \hat{R}(h, a, h') \max_{b \in A} Q(h', b)$
5: end if
6: end for

in a 2-action, 2-observation, $T = 2$ scenario. MCQ-ALT would invest $N$ samples to every leaf node in this experience tree. However, in cases where some histories are rare, this becomes a significant liability, since it requires a vast series of episodes to collect sufficient (i.e., $N$) samples of such rare histories. Furthermore, such histories possibly contribute little to the value function. A valid policy is a (small) part of the experience tree, as shaded in Figure 1. Clearly, there are many histories that can be allotted fewer samples because they do not partake in the optimal policy.

In this paper, we test a simple modification of MCQ-ALT, called “MCQ-ALT Adjusted for Rare Histories”, or MCQ-ALT-ARH. This modification estimates the likelihood of a full length history, $h_{T-1}$, that has been actually encountered, as $f_{h_{T-1}} = \frac{\hat{R}(h_{T-2}, a, h_{T-1})}{\sum_{a, \omega} \hat{R}(h_{T-2}, a, \omega)}$ (where $h_{T-1} = (h_{T-2}, a, \omega)$ for a specific $a$ and $\omega$) and only requires $f_{h_{T-1}} \cdot N$ samples for $h_{T-1}$ instead of $N$. In the experiments section, we perform this modification only for histories whose $f_{h_{T-1}}$ falls below a threshold, $\epsilon$, given as an external parameter.

5. ANALYSIS

We focus on sample complexity analysis of MCQ-ALT. Although the fixed policy of the other agent effectively reduces the Dec-POMDP to a POMDP, the fact that the other agent’s actions and observations are unobservable to the learner makes the analysis more complex than a POMDP. In particular, the number of scenarios encountered by the other agent (referred to as $K$) becomes a key parameter in the sample complexity analysis, which would not appear in comparable POMDP analyses.

First we note that the number of episodes needed for the empty history to be “Known” is

$$N[\Omega]^{T-1} |A|^T$$

since the number of distinct $(h_{T-1}, a)$ tuples is $|\Omega|^{T-1} |A|^T$. Also, given the backup process of histories becoming “Known” in our algorithm, when all $(h_{T-1}, a)$ become “Known” the empty history must also become “Known”, and this takes $N$ visitations of each tuple. The actual number of episodes needed is, however, likely to be greater than $N[\Omega]^{T-1} |A|^T$, because only part of the exploration process is under the learner’s control, where it can select $a$ but not $\omega$. Thus it can be led to revisit paths that are already “Known”.

While it is fairly intuitive that the episode complexity given above should be exponential in $T$, it is not immediately clear that so could $N$. This is precisely where the complexity of Monte Carlo reinforcement learning differs between POMDPs [6, 12, 9] and Dec-POMDPs. In order to demonstrate this, we first present a generic sampling process, and use the resulting sample complexity expression to derive $N$.

5.1 The Basic Sampling Process

Consider the following sampling process, with $K$ classes of random variables, $\{X_j\}_{j=1}^K$, such that

$$Y = \frac{\sum_{i=1}^{N_1} X_{i1} + \sum_{i=1}^{N_2} X_{i2} + \ldots \sum_{i=1}^{N_K} X_{iK}}{N},$$

where $N = \sum_{j}^K N_j$. The process generates a sample of some $X_j$ at each of the $N$ iterations, where the probability that the sample belongs to class $j \in [1, K]$ is $P_j$. Therefore, all $X_j$ as well as all $N_j$ are random variables. Suppose that $E[X_{ij}] = M_j$, $\forall i$, and that the maximum magnitude of any $X_{ij}$ is $X_{\text{max}} > 0$. We wish to estimate the unknown value $\sum_{j}^K p_j M_j$, therefore we call $|Y - \sum_{j} p_j M_j|$ the estimation error. We claim the following sufficient condition for bounding the estimation error, but give the proof in [2] due to lack of space.

Theorem 1. If the total number of samples is set $N \geq \max(K \eta, 4\eta/\min_j p_j)$ where

$$\eta = \frac{4X^2\max K}{\epsilon^2} \ln(8K/\delta),$$

then the estimation error is bounded, i.e., $P(|Y - \sum p_j M_j| > \epsilon) < \delta$.

5.2 Derivation of $N$

In our analysis we shall use the max norm function, i.e., $\|f - \tilde{q}\|$ represents $\max_x |f(x) - \tilde{q}(x)|$. We first establish the dependence of the error in the $Q$ functions on the errors in our estimates $\hat{H}$ and $\hat{R}$.

Lemma 2. If $\|\hat{H}_t - H^\ast_t\| \leq \epsilon_1$ and $\|\hat{R}_t - R^\ast_t\| \leq \epsilon_2$ for all $t$, then at any step $t$

$$\|Q_t - Q^\ast_t\| \leq (T-t)\epsilon_2 + (T-t-1)|\Omega|R_{\text{max}} \epsilon_1$$

Proof: By induction. For basis, $t = T - 1$. Since $T$ is the last step in an episode, $\|Q_{T-1} - Q^\ast_{T-1}\| = \|\hat{R}_{T-1} - R^\ast_{T-1}\| \leq \epsilon_2$, hence true. For the inductive case, we see that $\|Q(t, h, a) - Q^\ast(t, h, a)\|

= |\hat{R}(t, h, a) + \sum_{b} H_t(h_t, a, h', b) \max_{b} Q_{t+1}(h', b) - R^\ast(t, h, a) + \sum_{b} H^\ast_t(h_t, a, h', b) \max_{b} Q^\ast_{t+1}(h', b)|

\leq |\hat{R}_t - R^\ast_t| + \sum (\hat{H}_t \max Q_{t+1} - H^\ast_t \max Q^\ast_{t+1}) + H^\ast_t \max Q^\ast_{t+1} - H_t \max Q_{t+1})|$

\leq \epsilon_2 + |\sum (\hat{H}_t \max Q_{t+1} - H_t \max Q^\ast_{t+1}) + (\hat{H}_t \max Q^\ast_{t+1} - H^\ast_t \max Q^\ast_{t+1})|

\leq \epsilon_2 + \max (Q_{t+1} - Q^\ast_{t+1}) + |\sum (\hat{H}_t - H^\ast_t) \max Q^\ast_{t+1})|

In the last expression, the second term is upper bounded by $(T-t-1)\epsilon_2 + (T-t-2)|\Omega|R_{\text{max}} \epsilon_1$, by the induction hypothesis. In the third term, $\max Q^\ast_{t+1} \leq R^\ast_{\text{max}}$, and the sum is taken over all observations. Therefore the third term is upper bounded by $|\Omega| R_{\text{max}} \epsilon_1$. Adding the bounds of the three terms we get the result.

Lemma 2 implies that the error bound increases for smaller histories, and therefore is maximum at the empty history. This is why the learner must continue until $Q_0$ is sufficiently accurate, i.e., the empty history becomes “Known”. In the
following analysis, we characterize “sufficiently accurate”, to
derive a bound on \( N \) used by the algorithm (equation 7).  

**Theorem 3.** To ensure that \( \|Q_0 - Q_0^\ast\| \leq \alpha \) w.p. \( \geq 1 - \delta \), it is sufficient to set  
\[
N \geq \frac{\max(|S|^4/\Omega^{T-1}, 4/\rho) (4R_{\text{max}}T|S|^2|\Omega|^{T+1}) \ln(16|S|^4/\Omega^{T+1} \beta/\delta) }{\alpha^2}.
\]

in our algorithm, where \( \beta \) is given in equation 6 and \( \rho \) results from Definition 1.  

**Proof:** By Lemma 2,  
\[
\|Q_0 - Q_0^\ast\| \leq T\epsilon_2 + (T-1)|\Omega|R_{\text{max}} \epsilon_1.
\]

To achieve the \( \alpha \) bound on the error, it is sufficient to set  
\[
\epsilon_1 \leq \alpha/2(T-1)|\Omega|R_{\text{max}}\epsilon_1 \quad \text{and} \quad \epsilon_2 \leq \alpha/2T.
\]

Now the number of \( \bar{H} \) and \( \bar{R} \) entries that need to be learned are \( |\Omega|\beta \) and \( \beta \) respectively, where \( \beta \) is given in equation 6. Therefore, it is sufficient to require both of the following for any \( t \):

\[
P(|\bar{H}_t - H_t^\ast| > \alpha/2T) < \delta/2\beta \tag{8}
\]

\[
P(|\bar{R}_t - R_t^\ast| > \alpha/2(T-1)|\Omega|R_{\text{max}}) < \delta/2\beta|\Omega|
\]

First consider \( ||\bar{R}_t - R_t^\ast|| \) and equation 4. The estimation of any \( \bar{R}_t \) in our algorithm matches the description of the sampling process, with \( E[X_j] = R(s, a) \) and \( p_j = P(s|h_t, h_{t-1}) \cdot P(h_{t-1}|h_t, \delta) \); the last quantity being the reachability, of which the infimum is \( \rho \) (Definition 1). Note that \( p_j \) cannot be set to \( \sum_h P(s|h_t, h_{t-1}) \) \( P(h_{t-1}|h_t, \delta) \), since each sample \( X_j \) received corresponds to a specific history \( h_{t-1} \) encountered by the other agent. Therefore in this case, the number of variable classes in the sampling process is \( K = |S^2|\Omega^T \leq |S|^2|\Omega|^{T-1} \). This is the maximum possible number of terms in the summation of equation 4 which corresponds to full length histories that can be encountered by the other agent, given its (fixed) policy. Making the substitutions for \( K, X_{max}\), and using \( \alpha/2T \) for \( \alpha \) and \( \delta/2\beta \) for \( \delta \) in Theorem 1, we see that to ensure equation 8, it is sufficient to set  
\[
N \geq \frac{\max(|S|^4/\Omega^{T-1}, 4/\rho) (4R_{\text{max}}T|S|^2|\Omega|^{T+1}) \ln(16|S|^4/\Omega^{T+1} \beta/\delta) }{\alpha^2}.
\]

Similarly, for \( ||\bar{H}_t - H_t^\ast|| \), the sampling process is characterized by \( E[X_{jt}]=1 \) and

\[
p_j = P(s|h_t, h_{t-1})P(h_{t-1}|h_t, \delta)P(s'|s, a)P(\omega|s', a).
\]

The last quantity is the propagated reachability, of which the infimum is also \( \rho \). Since \( t \leq T-2 \), this yields \( K = |S|^2|\Omega|^{T-1} \leq |S|^2|\Omega|^{T-1} \), and we have  
\[
N \geq \frac{\max(|S|^4/\Omega^{T-1}, 4/\rho) (4R_{\text{max}}(T-1)|S|^2|\Omega|^{T+1}) \ln(16|S|^4/\Omega^{T+1} \beta/\delta) }{\alpha^2}.
\]

Combining the two, we get the result. \( \square \)

It is interesting to note that \( N \) is polynomial in most problem parameters, except that it is logarithmic in \( |A| \), and exponential in \( T \). Although Theorem 3 suggests that \( N = O(T^3|\Omega|^{2T}) \), our experiments suggest that it does not require to grow in some domains due to simpler structure. 

Even in the domains where it does need to grow, the rate of growth could be lower.

### 6. Evaluation

#### 6.1 Initial Policy

If agents alternate in learning best responses, the agent that does not learn initially must play some previously specified fixed policy. Our experiments show that if this policy is random then the final outcome is unpredictably poor. Instead, we simply let the two agents perform concurrent reinforcement learning to learn initial policies, on a slightly simpler Dec-POMDP. This Dec-POMDP reduces the set of observations to one dummy observation. In other words, the agents simply ignore the observations, and learn a mapping from their own past action histories to actions \( (\pi_1, \pi_2, \ldots, \pi_n) = \pi \). This policy can be readily translated to the regular policy language, by setting \( \pi_1(a_1, \omega_1, a_2, \omega_2, \ldots, a_n, \omega_n) = a \) for all possible chains of observations \( (\omega_1, \omega_2, \ldots, \omega_n) \). This is the policy used by the initially non-learning agent. More details on the generation of this initial policy, and comparison with alternatives, can be found in [10, 11].

#### 6.2 Policy Computation and Evaluation

Although a learner computes its best response policy as given in equation 3, its executable policy can be given more compactly, since not all histories encountered during learning will be encountered when executing its best response policy. This executable policy can be constructed by only considering histories of the form  
\[ h_t = (h_{t-1}, \pi_t(h_{t-1}, \omega)) \]

for all possible observations \( \omega \), starting at \( h_0 = \emptyset \). If an observation was never encountered at some \( (h_{t-1}, \pi_t(h_{t-1})) \), then the learner can output a random action. In this case however, all histories that contains this history as a prefix will also be unseen, and will constitute an unlearned part of the policy tree.

In contrast with [21], we use the exact method (instead of simulation) to evaluate a joint policy, since we limit our evaluations to precisely defined benchmark problems. For more practical problems, the simulation approach to policy evaluation would be the only option. To evaluate a joint policy \( \pi \), we find  
\[ V^\pi(h_0) = \sum_{s \in S} b_0(s)V^\pi(h_0, s) \tag{9} \]

where \( b_0 \in \Delta(S) \) is the initial state distribution. \( V^\pi(h_t, s) \) for a given joint history \( h_t \) and state \( s \) is given by  
\[ V^\pi(h_t, s) = R(s, \pi(h_t)) + \sum_{s' \in S} P(s'|s, \pi(h_t)) \cdot \sum_{\omega \in \Omega} O(\omega|s', \pi(h_t))V^\pi((h_t, \pi(h_t), \omega), s'). \]

#### 6.3 Experimental Results

We present experimental results from two benchmark domains: DEC-Tiger [14] and RECYCLING-ROBOTS [1]. We used the initial policy learned by concurrent reinforcement learning (as described above) over 200000 episodes to perform alternating Monte-Carlo Q learning as described in this paper, for values of \( N \) ranging from 10 to 1000. There were 2 alternations, i.e., each agent learned best response to the other’s policy exactly once. In each experiment, the resulting joint policy after 2 alternations was evaluated to yield...
\[ |v_{jpol} - v_{opt}|/|v_{opt}|, \] i.e., the relative error based on known optimal values for horizons 3, 4 and 5. The plots in Figure 2 show these relative errors averaged over 50 runs. For comparison, we also show the result from concurrent Q-learning (referred to as “Q-conc”), with \( \alpha = 0.01, \epsilon = 0.005 \), which were found to produce best results in the selected settings. Table 1 also shows the average relative error rates with the initial policy (derived by concurrent learning), to verify that MCQ-ALT does indeed improve these policies.

Each setting of \( N \) and \( T \) makes MCQ-ALT finish in a certain number of episodes, say \( e_{N,T} \), i.e., until the empty history becomes “Known” in each alternation. The average of these numbers of episodes over all agents and all runs is used to determine the length of the Q-conc runs. The average relative error of Q-conc for a given \( N \) and \( T \) is reported at the end of (average) \( e_{N,T} \) episodes.

In Figure 2 (left) for Dec-Tiger, first we note that horizons 3 and 4 are solved accurately with \( N \geq 200 \) and 1000 respectively, by MCQ-alt. Q-conc solves horizon 3 accurately with a number of episodes corresponding to \( N = 1000 \), but is unable to solve horizon 4. Neither achieves 0 error for horizon 5. MCQ-ALT is also clearly more efficient than Q-conc. More importantly, we see that for a given \( N \), the relative error increases with increasing horizon. This is clear with MCQ-alt, but not so clear with Q-conc with even a hint of non-convergence (error increases for \( T = 4 \)). For MCQ-alt, this implies that \( N \) needs to increase to produce the same error on increasing horizons. This is direct evidence for the claim made earlier in this paper, although the rate at which \( N \) needs to increase falls short of the \( O(T^3|\Omega|^{2T}) \) rate established in this paper. This is explained by the fact that \( O(T^3|\Omega|^{2T}) \) is a sufficient rate for value convergence; it is not necessary, and policies can converge sooner.

MCQ-ALT and Q-conc. This is partly due to the fact that there are many unreachable histories in this domain. Table 1 shows the increasing proportion of unreachable histories in RECYCLING-ROBOTS with increasing \( T \), which suggests that \( K \) (and hence the error) grows much slower than Theorem 3 assumes. As it turns out, \( N \) does not need to grow with increasing horizons for small errors in this domain. Instead, the increasing values of \( e_{N,T} \) even for a fixed \( N \) are sufficient to actually reduce the average errors with increasing \( T \), as seen in Figure 2 (right). However, it must be noted that alternating best response learning also almost always converges to local optima in this domain, so 0 average error was never observed.

In Table 2, we show the results from MCQ-ALT-ARH in the two domains for \( T = 3,4 \) only. Here “Episode ratio” stands for the ratio of the average number of episodes (over all agents and all runs) needed by MCQ-ALT-ARH, to that needed by MCQ-ALT, both for \( N = 1000 \). \( \Delta \) (relative error) stands for the absolute difference between the average relative errors of the two algorithms for the same settings. As one would expect, the relative number of episodes needed to terminate MCQ-ALT-ARH falls with increasing \( \epsilon \), while the error in the policy value increases. Interestingly, the impact on the quality of policy is mostly small except in Dec-Tiger with \( T = 4 \). In the other settings, it appears that the adjustment for rare histories does indeed lead to significant savings in learning time with relatively little impact on the policy quality. More experiments need to be conducted to ascertain the beneficial impact of MCQ-ALT-ARH on a broad range of problems, and its relation to domain characteristics.

<table>
<thead>
<tr>
<th>( \epsilon )</th>
<th>Dec-Tiger</th>
<th>Recyling-Robots</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T = 3 )</td>
<td>( T = 4 )</td>
<td>( T = 3 )</td>
</tr>
<tr>
<td>Episode ratio</td>
<td>0.01</td>
<td>0.0975</td>
</tr>
<tr>
<td></td>
<td>0.02</td>
<td>0.864</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>0.189</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>0.055</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.025</td>
</tr>
<tr>
<td>( \Delta ) (relative error)</td>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>0.098</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.137</td>
</tr>
</tbody>
</table>

Table 1: Relative errors of initial policies, and the proportion of unreachable histories.

In Figure 2 (right) for RECYCLING-ROBOTS, we see something more interesting. Although MCQ-ALT is still more efficient that Q-conc, the relative errors now decrease with increasing horizon, for a given \( N \). This is true with both
7. CONCLUSION

We have presented a distributed Monte Carlo based reinforcement learning algorithm for solving decentralized POMDPs approximately. Agents alternate in learning best responses which, if accurate enough, is guaranteed to lead to a locally optimal Nash equilibrium. We have derived the sample complexity that guarantees arbitrarily accurate best response policies, and shown empirically that 2 alternations of best response learning can produce (near) optimal joint policies in some benchmark problems. A slight modification of the algorithm was also proposed to account for rare histories, and it appears to significantly reduce the learning time, with relatively little impact on policy quality in most settings. In the future, more judicious use of samples with a variable N will be explored, and a more elaborate investigation into the convergence behavior of concurrent learning will be undertaken. An important future goal is to apply our approach to real-world Dec-POMDP problems where models are unavailable, e.g., two inexpensive (limited sensing and no communication capability) robots carrying an object together. The main challenge in such an application is the fact that observations will be continuous valued, for which naive (discretization) as well as sophisticated (function approximation) techniques will be investigated.

8. ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for helpful feedback. This work was supported in part by the U.S. Army under grant #W911NF-11-1-0124.

9. REFERENCES

Planning Delayed-Response Queries and Transient Policies under Reward Uncertainty

Robert Cohn
Computer Science and Engineering
University of Michigan
rwcohn@umich.edu

Edmund Durfee
Computer Science and Engineering
University of Michigan
durfee@umich.edu

Satinder Singh
Computer Science and Engineering
University of Michigan
baveja@umich.edu

ABSTRACT

We address situations in which an agent with uncertainty in rewards can selectively query another agent/human to improve its knowledge of rewards and thus its policy. When there is a time delay between posing the query and receiving the response, the agent must determine how to behave in the transient phase while waiting for the response. Thus, in order to act optimally the agent must jointly optimize its transient policy along with its query. In this paper, we formalize the aforementioned joint optimization problem and provide a new algorithm called JQTP for optimizing the Joint Query and Transient Policy. In addition, we provide a clustering technique that can be used in JQTP to flexibly trade performance for reduced computation. We illustrate our algorithms on a machine configuration task.

Keywords

Human-robot/agent interaction, Reward structures for learning, Single agent Learning

1. INTRODUCTION

The work described in this paper addresses a problem that arises when one agent (which we will refer to as the robot) is acting on behalf of another agent (which we will refer to as the (human) operator). The robot should act so as to bring about states of the world that satisfy the preferences of the operator, but in complex environments the effort required of the operator to express preferences that account for every possible circumstance is prohibitive. In this paper, we assume that the operator’s preferences are expressed as a reward function. Thus, in all but very simple domains the robot is inherently uncertain about some aspects of the operator’s true reward function. Should the world the robot encounters veer into regions where its uncertainty over the operator’s true rewards is high, the robot’s only way to reduce its uncertainty is to ask the operator one or more queries.

Such a robot is semi-autonomous: it can behave autonomously for considerable intervals of time (in more routine circumstances where the operator’s preferences are unambiguously clear) but might periodically need human assistance. Because its human operator is only called on to help infrequently, the operator’s valuable cognitive capabilities will be shared with other tasks (such as operating other robots). Thus, we assume that the operator is not necessarily paying attention to the robot at all times. When the robot asks for assistance, the operator might require some (variable) time to complete or suspend her current cognitive activities, to understand what exactly the robot is asking, to introspect to discern an appropriate answer, and to articulate the response. Delays in communication channels can further extend the time lapse between when the robot asks a query and when it receives a response.

Because getting a response from the operator incurs cost (distracting the operator from other possibly valuable cognitive tasks), the robot must choose with care when to query and what to query about. Furthermore, because receiving a response might take some time, the robot needs to decide what to do while waiting. In fact, because the value of a query response often depends on what state the robot is in when that response arrives, the decisions about what to ask and what to do while waiting for the response are intertwined.

We focus in this paper on the formulation and solution of this joint sequential decision problem that the robot faces. We should note that in the treatment we provide here, the other agent in the system—the human operator—is simply assumed to provide a (truthful) response to the posed query after some (stochastic) delay. Thus, the sequential decision problem we consider here is multiagent only in a degenerate sense, in that the query response policy of the operator is assumed fixed. Nevertheless, solving the robot’s intertwined problems of deciding on a query and on a transient policy to follow while awaiting the response is itself challenging, and is influenced by the operator’s responsiveness.

In the remainder of this paper, we formally define this problem, and then identify aspects of the problem structure that can be exploited as part of a greedy myopic approach. Finally, we describe a technique for reducing computation in a flexible way by clustering together hypothetical future reward function beliefs arising from potential responses to queries.

2. PROBLEM DEFINITION

In this section, we define the two interacting processes that constitute the overall problem faced by the (robot) agent, the decision-making process that models the agent’s interaction with the environment and then the reward-knowledge process which models the agent’s interaction with the operator.

Decision-Making Process

First we define the two elements that form the decision-making process faced by the agent, namely controllable Markov processes and uncertainty over reward functions.

In a Controlled Markov Process (CMP), at time $t$ the agent oc-
couples state $s_t \in S$, executes an action $a_t \in A$, and stochastically transitions to state $s_{t+1}$ with probability governed by transition function $T(s_{t+1} | s_t, a_t)$. Given any reward function $R : S \times A \rightarrow \mathbb{R}$ that maps states and actions to reals, the value function of a policy $\pi \in \Pi$, where $\pi : S \rightarrow A$, is $V^\pi_R(s) \triangleq \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, \pi \right]$, the expected discounted sum of rewards achieved when the agent starts in state $s$ and behaves according to policy $\pi$. Thus, a reward function $R$ induces a partial ordering over policies for a CMP. The optimal policy $\pi^*_R$ satisfies $V^*_R(s) \geq V^R_{\pi^*_R} \forall \pi \in \Pi$ and is guaranteed to exist; its value function $V^*_R(s)$ is termed the optimal value function and denoted $V^*_R$. The optimal value function and corresponding optimal policy can be computed by algorithms such as value iteration and policy iteration.

CMPs are identical to MDPs except that, unlike MDPs, CMPs do not assume that the rewards are an observation from the environment. Instead, CMPs treat rewards as an expression of a (operator's) preference ordering over policies.

In this paper, we assume that an agent knows the CMP, but has uncertainty over which reward function is the operator's true reward function. The uncertainty is expressed as a distribution $\psi \in \Psi$ over a finite set of reward functions $\{R_1, R_2, \cdots, R_n\}$ (we will interchangeably use $\psi$ as a distribution and as a knowledge-state). The expected value function with respect to a reward function distribution $\psi$ that never changes is defined as follows:

$$V^\psi_R(s) \triangleq \mathbb{E}_{R \sim \psi} \left[ V^R_{\pi^*_R}(s) \right]$$

$$= \mathbb{E}_{R \sim \psi} \left[ \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, \pi \right] \right]$$

$$= \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, \pi \right],$$

(1)

where $R \sim \psi$ denotes reward function $R$ drawn with probability governed by $\psi$, and where $R^\psi(s, a) \triangleq \sum_{i=1}^{n} \psi(i) R_i(s, a)$ is the mean-reward for state-action pair $s, a$ under distribution $\psi$. $\psi(i)$ denotes the probability assigned to reward function $R_i$ by $\psi$. The Bayes-optimal policy $\pi^*_\psi \triangleq \arg \max_{\pi} V^\psi_R$ and the corresponding Bayes-optimal value function is denoted $V^*_\psi$. Note that solving for the Bayes-optimal policy with uncertainty only over rewards as above is no harder than solving a traditional MDP [8], so long as the mean-reward function can be calculated efficiently.

In this paper we consider situations in which an agent's decision-making problem is a CMP with uncertain rewards, but where the uncertainty over reward functions is not fixed for all time because the agent has the capability of actively querying its operator to acquire information about the distribution over rewards.

Reward-Knowledge Process

Here we consider CMPs with reward uncertainty in which the agent has the ability to query its operator about the reward function, where the operator in turn responds to these queries with some stochastic temporal delay. The response to a query is informative in some way about the true reward function, and thus potentially leads to an update to the distribution over rewards, but otherwise has no effect on the CMP. We will assume that the agent can only have one outstanding query at a time, and that each query has an associated cost. We now formalize the reward-knowledge process as a controlled semi-Markov process (CSM).

The state of the reward-knowledge process is the distribution $\psi$ over rewards $\{R_1, R_2, \cdots, R_n\}$ and the number of time steps since the last querying action was taken. Let the initial distribu-

3. JOINT QUERY AND TRANSIENT-POLICY PLANNING

The general problem of planning jointly optimal query and action policies in RQCMs presents a challenging optimization problem, which in this paper we will approximate by repeated myopic planning. Specifically, we consider what query an agent should select and how it should act whilst waiting for the query to return, ignoring the possible effects of future queries. It is repeated myopic planning because we repeat this myopic joint optimization whenever a query returns.
Query+Transient-Policy Value Function. Consider the expected discounted summed value of mean-rewards as a function of state \( s \), query \( q \), and transient non-stationary policy \( \pi \in \Pi^{\text{max}} \) (where \( \Pi^{\text{max}} \) is the set of non-stationary policies defined from the current time step to \( \tau_{\text{max}} \) additional time steps). We emphasize that \( \pi \) is transient because it terminates when the query returns, and non-stationary because in general the policy maps states and “time since the start of the policy” to actions. This query+transient-policy value function can be decomposed into three components by exploiting the factored structure of RQCMs. Formally, when the knowledge-state is \( \psi \) and there is no outstanding query (recall that only one outstanding query is allowed at a time),

\[
Q_{\psi}(s, q, \pi) = c(q) + r_{\psi}^0(s, \pi) + \sum_{s'} T^q(s'|s, \pi) \sum_{\psi' \in \Psi(q, \psi)} H_q(\psi'|\psi)V_{\psi'}(s')
\]  

(See Appendix for a derivation.) The first term, \( c(q) \), is the cost of asking the query. The second term is the expected value of the discounted sum of mean-reward obtained by the transient policy until the query returns; the expectation here is over the random state transitions in the decision-making process as well as the randomness over return times. More formally, the second term is defined as

\[
r_{\psi}^0(s, \pi) = \max_{\tau=0}^{\tau_{\text{max}}} F_q(\tau)E\left[ \sum_{k=0}^{\tau-1} \gamma^k P(s_{k+1}, q, \pi)|s_0 = s, \pi, \tau \right].
\]

The third term is the expected value of the state at the time the query returns; here the expectation is over the random state transitions of the decision-making process when the query returns. Note that the \( V_{\psi'} \) is the optimal posterior value that is computed assuming no further queries; hence the myopic nature of the query+transient-policy value function. More formally,

\[
T^q(s'|s, \pi) = \sum_{\tau=1}^{\tau_{\text{max}}} F_q(\tau) \sum_{k=0}^{\tau-1} \gamma^k P(s'|s, \pi, \tau)
\]

where \( P(s'|s, \pi, \tau) \) is the probability that the state in the decision making process \( \tau \) steps after the non-stationary transient policy \( \pi \) starts in state \( s \) is \( s' \). Furthermore note that \( T^q \) takes into account the effect of discounting because of the delay in query response. In summary, Equation 3 defines the expected discounted sum of mean-rewards obtained by an agent if it asks query \( q \) in state \( s \) and then behaves according to policy \( \pi \) until the query returns, after which the agent behaves optimally with respect to the posterior distribution over rewards (without asking any additional queries).

Given the definition of the query+transient-policy value function above, we can define three optimization problems (we will use the first and second in our empirical results below; the third is included here for completeness):

**Joint Query and Transient-Policy Optimization**

\[
(q^*, \pi^*)|s, \psi = \arg\max_{q \in Q, \pi \in \Pi^{\text{max}}} Q_{\psi}(s, q, \pi)
\]  

where \( \Pi^{\text{max}} \) is the set of non-stationary policies of length \( \tau_{\text{max}} \).

**Transient-Policy Optimization**

\[
\pi^*|s, q, \psi = \arg\max_{\pi \in \Pi^{\text{max}}} Q_{\psi}(s, q, \pi)
\]

**Query Optimization**

\[
q^*|s, \psi, \pi = \arg\max_{q \in Q} Q_{\psi}(s, q, \pi)
\]

Relationship to Options. The form of the query+transient-policy value function in Equation 3 resembles the option-value functions constructed in the options literature [13], where algorithms have been developed to solve for the option-value function. However, our form has two key differences. First, we have a joint optimization over the query space and transient-policy space, rather than an optimization over some option space. The transient-policy space is much larger than the usual case of a small number of options usually considered. Second, we have an additional inner sum over knowledge-states that takes into account possible changes caused by the query and its response.

**Algorithm Components**

We now present our Joint Query and Transient Policy (JQTP) algorithm, which solves the Joint Query and Transient-Policy Optimization problem (Equation 3) for a given \( s \) and \( \psi \). JQTP solves the transient-policy optimization problem for each possible query by solving a nested optimization problem as described below, and then picks the best query along with its paired optimal transient policy. We rewrite the transient-policy optimization problem below for reference (noting that \( c(q) \) does not depend on the transient policy):

\[
\pi^*|s, q, \psi = \arg\max_{\pi \in \Pi^{\text{max}}} \left\{ r_{\psi}^0(s, \pi) + \sum_{s'} T^q(s'|s, \pi) \sum_{\psi' \in \Psi(q, \psi)} H_q(\psi'|\psi)V_{\psi'}(s') \right\}.
\]

We can solve this transient-policy optimization problem by defining an induced terminal-reward problem in which the value for a particular non-stationary \( \pi \) depends on the summed discounted rewards \( r_{\psi}^0(s, \pi) \); it receives the query is returned, at which point it receives terminal reward \( \sum_{\psi' \in \Psi(q, \psi)} H_q(\psi'|\psi)V_{\psi'}(s') \) depending on which state \( s' \) it occupies. Note that both of the terms in the terminal reward, \( H_q(\psi'|\psi) \) and \( V_{\psi'}(s') \) are independent of \( \pi \) (the distribution over \( s' \) is a function of \( \pi \), which links the reward-knowledge and decision-making processes). Thus, this terminal-reward process can be represented as a non-stationary CMP, solvable by standard MDP solving techniques.

However, constructing this transient process requires the expensive computation of \( \sum_{\psi' \in \Psi(q, \psi)} H_q(\psi'|\psi)V_{\psi'}(s') \) for every \( s' \) in which the agent might receive the response to its query, each of which requires solving the mean-reward MDP associated with each possible \( \psi' \). Thus, as specified, JQTP’s asymptotic complexity depends linearly on both the number of states reachable within \( \tau_{\text{max}} \) time steps, and the number of possible posteriors considering all possible queries (the latter of which amounts to \( \sum_{q \in Q} |\Psi(q, \psi)| \) value calculations). The latter issue precludes JQTP as stated from being applied to problems with large query spaces without further approximations. In the next section, we present such an approximation in the form of a clustering technique.

**Posterior Belief Clustering**

In this section, we propose reducing the number of different optimal posterior value \( V_{\psi'}(s') \) calculations in Equation 3 by clustering posterior beliefs, and calculating the posterior value only once for each cluster (sharing that value among each member of the cluster). This idea is founded on the principle that similar posteriors should share similar values. We now show that this principle has theoretical justification by bounding the maximal difference between the optimal posterior value functions for posteriors which induce similar mean reward functions. The following Lemma, a result similar to those proved in [10] and [15], will be useful in proving our bound:
**Lemma 1.** If \( ||R_1 - R_2||_\infty \leq \epsilon \), then for all \( \pi \),
\[
||V^\pi_{R1} - V^\pi_{R2}||_\infty \leq \frac{\epsilon}{1 - \gamma},
\]
where \( ||x||_\infty = \max_{i} x_i \).

**Proof.** Using the fact that \( V^\pi_{R1} = (I - \gamma P^\pi)^{-1} R_1 \), and \( V^\pi_{R2} = (I - \gamma P^\pi)^{-1} R_2 \),
\[
||V^\pi_{R1} - V^\pi_{R2}||_\infty = ||(I - \gamma P^\pi)^{-1}(R_1 - R_2)||_\infty,
\]
where \( (I - \gamma P^\pi)^{-1}(R_1 - R_2) \) is the value function of policy \( \pi \) for the reward function \( R' = R_1 - R_2 \). By assumption \( ||R'||_\infty \leq \epsilon \) and thus with reward function \( R' \) the maximum value for any state would be \( \frac{\epsilon}{1 - \gamma} \) and the minimum value for any state would be \( -\frac{\epsilon}{1 - \gamma} \).

**Theorem 1.** Given two reward functions \( R_1 \) and \( R_2 \), if \( ||R_1 - R_2||_\infty \leq \epsilon \), then \( ||V^\pi_{R1} - V^\pi_{R2}||_\infty \leq \frac{\epsilon}{1 - \gamma} \).

**Proof.** Let \( \pi_1^* \) be optimal w.r.t. \( R_1 \) and \( \pi_2^* \) be optimal w.r.t. \( R_2 \).

Then, we know that \( \forall s \)
\[
V^\pi_{R1}(s) \leq V^\pi_{R2}(s) + \frac{\epsilon}{1 - \gamma}, \quad \text{(By Lemma 1)}
\]
\[
\leq V^\pi_{R2}(s) + \frac{\epsilon}{1 - \gamma}
\]
and
\[
V^\pi_{R1}(s) \geq V^\pi_{R2}(s) - \frac{\epsilon}{1 - \gamma}, \quad \text{(By Lemma 1)}
\]

Thus \( \forall s \)
\[
V^\pi_{R1}(s) - \frac{\epsilon}{1 - \gamma} \leq V^\pi_{R2}(s) \leq V^\pi_{R1}(s) + \frac{\epsilon}{1 - \gamma}.
\]

We now show that when the agent has uncertainty only in rewards, the maximal difference in optimal value between two posteriors can be bounded by the maximal difference between the mean reward functions induced by the two posteriors:

**Theorem 2.** Given distributions \( \psi_1 \) and \( \psi_2 \) over reward functions,
\[
||E_{R_1 \sim \psi_1}[V^\pi_{R1}] - E_{R_2 \sim \psi_2}[V^\pi_{R2}]||_\infty \leq \frac{1}{1 - \gamma} ||\bar{R}_{\psi_1} - \bar{R}_{\psi_2}||_\infty.
\]

**Proof.**
\[
||E_{R_1 \sim \psi_1}[V^\pi_{R1}] - E_{R_2 \sim \psi_2}[V^\pi_{R2}]||_\infty
= ||V^\pi_{R1} - V^\pi_{R2}||_\infty
\leq \frac{1}{1 - \gamma} ||\bar{R}_{\psi_1} - \bar{R}_{\psi_2}||_\infty.
\]

We now present our clustering algorithm, which given the agent’s current state \( s \) and belief \( \psi \), groups all possible posterior beliefs \( \psi' \in \bigcup_{\psi \in \Psi} \Psi(\psi, q) \).

**K-means Clustering of Posterior Beliefs.** Theorem 2 above shows that if the mean reward functions induced by two posteriors are close, the difference in expected value for a particular state given the two posteriors is small. We make use of this fact to save computation within JQTP in calculating \( \sum_{\psi' \in \Psi(\psi, q)} H_q(\psi'|\psi)V_{\psi'}(s') \) for each \( q \) and state \( s' \) reachable (from \( s \)) as follows.

Given the agent’s current state is \( s \), we cluster all posteriors \( \psi_1, \psi_2 \) in \( \bigcup_{\psi \in \Psi} \Psi(\psi, q) \) using K-means with the following distance function:
\[
D(\psi_1, \psi_2) = ||\bar{R}_{\psi_1} - \bar{R}_{\psi_2}||_1,
\]
where each posterior \( \psi' \) is assigned to cluster \( J(\psi') \). We can then use the clustering to compute the needed terms just for the cluster center and to use them as approximations to the terms for all the posteriors that fall into that cluster. Specifically, for each \( s' \) the agent could occupy when the query is returned and cluster \( j \), we compute the value given the posterior mean reward function \( \bar{V}_j \) represented by the center of \( j \):
\[
\bar{V}_j(s') = \frac{1}{\sum_{s} \sum_{\psi \in \Psi(\psi, q)} H_q(\psi'|\psi)V_{\psi'}(s')}. \]

Then, for each \( s' \) and \( q \), we estimate
\[
\sum_{\psi' \in \Psi(\psi, q)} H_q(\psi'|\psi)V_{\psi'}(s') \approx \sum_{\psi \in \Psi(\psi, q)} H_q(\psi'|\psi)\bar{V}_j(s').
\]

We term the JQTP algorithm that utilizes this clustering estimation technique JQTP Clustering. Note that the number of clusters is an input parameter to the K-means algorithm and hence to the JQTP Clustering algorithm; this will determine the amount of computational savings achieved by clustering as well as the quality of the approximation. In Section 4 we will show empirical results that demonstrate the significant tradeoffs in using JQTP Clustering compared to JQTP.

### 4. EXPERIMENT: MACHINE CONFIGURATION

In Section 3 we introduced JQTP, an algorithm that, at a particular decision point, finds an approximately optimal joint query and action policy as prescribed by Equation 3. The approximation is that JQTP is myopic, assuming that only one query can be asked and can only be asked at the current time step. In addition, we presented a method for clustering posterior beliefs that can be used to reduce the computation required by JQTP, and proved a worst-case bound on the difference in value given two posterior beliefs. In this section, we give a preliminary empirical evaluation of JQTP, comparing its performance and computational needs to alternative algorithms for computing a joint query and transient policy. In addition, we examine the degree to which clustering to reduce computation reduces the quality of the joint query and transient policy found.

**Domain Description.** We conduct our evaluation in the Machine Configuration domain, in which the agent’s task is to build a machine for its user consisting of \( N \) components \( \{c_1, c_2, ..., c_N\} \). There are \( M \) different types \( \{t_1, t_2, ..., t_M\} \) that can be assigned to each component \( c_i \). We refer to the type assignment for component \( c_i \) as \( l(c_i) \). The configuration agent has several actions at its disposal. It can select a next component to assign a type to. If it has already selected a component, it can assign a type to it. Hence, configuring a component is a two-step process (selecting the component and then assigning it a type). The agent also has an action of waiting, which leaves a partial configuration unchanged for the time step. However, at each time step until the configuration is completed and given to the user, the agent receives a penalty of \(-0.2\); hence, there
is a sense of urgency in completing the task, as waiting is costly.

The user has preferences over the assignment to each component that can be modeled in terms of reward function $R(t(c_i))$, and the reward for a complete configuration is the sum of the component reward functions $\sum R(t(c_i))$. The configuration agent, however, may not know the user’s component reward functions, and represents its uncertainty as a probabilistic distribution over the components’ possible reward functions. To reduce its uncertainty, the agent can pose to the user an action-query [4], which asks the user “What would you do?” given a partially specified machine, where the question may only be posed as asking which type to specify for a particular component. The user’s response, which for our experiments comes after a deterministic delay of 2 time units, allows the configuration agent to improve its probabilistic model of the user’s true component reward functions. Here we assume the user’s response is deterministic in that the user is sure of the optimal action; we refer to our earlier work [4] for handling noise in the user’s responses. Note that we have not included every possible state in the query space, as queries asking which part to configure next are useless since the user is indifferent as to in which order the parts are configured.

Hence, the configuration agent faces a decision about what query (if any) to pose, and what policy to follow while waiting for the response. Intuitively, the agent might ask about a component whose reward function it is least certain about. While awaiting a response, it might assign components about which it is reasonably certain. In the extreme case where it is sufficiently uncertain about any components, and the possible reward values are high enough, the agent might justifiably choose to wait for a response before making any assignment. (Note: We assume that once a component is assigned it cannot be changed.) Or, when query costs are high enough, the agent might make assignments as best as it can without asking any queries.

**Experiment Setup.** We compare five algorithms in the Machine Configuration domain. As a baseline, the first algorithm, Prior Policy, does no querying, and simply follows the optimal policy given the agent’s current model of the possible reward functions. The second algorithm, Optimal Query For Prior Policy, incorporates JQTP’s technique of iterating through the query space to find a (myopically) optimal query given a transient policy, but again assumes the agent follows the prior policy (corresponding to acting according to Equation 4 with $\pi = \pi^*_c$). That is, Optimal Query For Prior Policy follows a transient policy that has not been formulated to complement the query that is being asked, but has a chance to update its policy according to the response of its query before the machine has been completely configured (since a minimum of 6 actions must be taken in total to complete the task). The third algorithm, Optimal Query For Wait Policy, is at the other extreme: it follows the “always wait” transient policy and optimizes the query for it. The fourth algorithm, JQTP, is the full implementation of JQTP as described in Section 3, and finds a joint query and transient policy that together are guaranteed to maximize expected (myopic) value (acting according to Equation 3). Finally, the fifth algorithm JQTP Clustering is JQTP with posterior belief clustering, where we vary the $K$ parameter used in K-means to explore tradeoffs between computational cost and solution quality.

We use a small version of the Machine Configuration domain with query cost set to 0 for every query (varying query costs did not produce any interesting additional trends), discount set to 0.9, and $N$ and $M$ both set to 3. The algorithms were required to select a query at the start state (except for Prior Policy), and were not allowed any additional queries. We measured the expected performance in terms of value accrued by each algorithm, where the expectation was estimated over 180 trials: trials differed by the priors used by the agents (though identical across the agents for a given trial). On each trial, 10 candidate settings of $R(t(c_i))$ were sampled, where the reward for each possible component type was sampled uniformly from $[0, 3]$, and each agent’s prior was uniform over these 10 candidate settings. For each trial, values were measured as an expectation with respect to the trial’s prior. Even though the candidate reward functions were sampled uniformly, the small number of samples biased the agent’s priors towards expecting larger or smaller rewards for particular component types.

**Results.** Figure 1a shows our results for the above experiment. As expected, JQTP performs best since it is optimal in this setting where only one query may be asked and must be asked at the
beginning. Somewhat surprisingly, Optimal Query For Wait Policy performs worst, even worse than electing not to query at all. This is due to the fact that taking two steps longer to configure the machine penalizes the agent more than it gains by asking the best query which will be received before committing to any component types. Optimal Query For Prior Policy performs better than Prior Policy but worse than JQTP. This is because, as intuitively expected, JQTP queries to determine the user’s preferences about components it is less certain about, and while waiting assigns components that it is more certain about. In contrast, Optimal Query For Prior Policy assigns components in a rigid order (unaffected by querying) and so decides on a query around this order, meaning that sometimes it asks a query that is less useful because an answer to a more useful query would be received after the corresponding component would already be assigned. These results show that in this domain, optimal transient behavior (paired with its optimal query) lies somewhere in between the two extremes of always waiting for a response before taking irreversible actions and never waiting.

Lastly, JQTP Clustering displays improvement as the number of clusters used in K-means is increased (as it makes finer distinctions between queries’ results), approaching the performance of JQTP after $K = 20$. This means it is able to perform nearly as well as JQTP by evaluating only 20 posterior beliefs per reachable state, as opposed to JQTP’s evaluation of all ($768$) possible posterior beliefs per reachable state. The computational implications are shown in Figure 1b: in this domain, on average, JQTP Clustering can perform nearly as well as JQTP without clustering with only approximately $30\%$ of the computation (which includes that of performing the clustering step).

These positive results can be explained by noticing that the optimal number of clusters in this problem is 10: there are 10 distinct posteriors possible after asking a single query (3 distinct posteriors are possible for each query asking about one of the three components, and 1 posterior equaling the prior is possible if the user were to respond with “wait”). Clustering allows JQTP to avoid computing redundant values, and further shows a graceful degradation in performance as less similar posteriors are grouped. Overall, these results suggest that clustering posterior beliefs shows promise as an effective means for flexibly balancing performance and computation in domains with time-delayed query responses.

5. RELATED WORK

Our previous work ([5], [4]) and similar work ([6]) deal with planning under different forms of query and environment uncertainty. These methods can be seen as bridging Bayesian Reinforcement Learning [12] and POMDP Planning [7] whereby the methods exploit structured uncertainty and observations. However, they do not consider the temporal dynamics and concurrency induced by delayed in receiving query responses as we do in this work.

Temporal abstraction and concurrency are issues that have been addressed together before in Hierarchical Reinforcement Learning [2] and other similar work [14], but these methods do not consider environment model uncertainty explicitly, and thus do not exploit the factored relationship between physical state and knowledge state with respect to queries and actions, as JQTP does.

Our setting can be viewed as a degenerate multiagent system in which one agent (the operator, which we represented by the reward-knowledge process) has a fixed policy (i.e., when the agent queries $q$, from the agent’s viewpoint the operator responds according to $H_q(\psi|\psi)$ after a stochastic delay). This structured interaction allowed us to develop a more scalable solution than solutions developed for similar problems explored in the rich multiagent communication literature, where in the typical case agents may communicate their observation histories (or portions of them). In particular, Spaan et al. [11] analyzed multiagent communication under state uncertainty with stochastic communication delays, but assumed that communication involves exchanging entire histories of observations, eliminating the what to query question that we address. Further, their approach assumes that communication is either delayed by one time step or fails entirely, while our approach allows for any bounded communication delay distribution. Roth et al. [9] provide an algorithm for the what to query problem in a similar setting to Spaan et al. [11], but do not account for stochastic communication delays. Thus, our setting can be roughly viewed as the intersection of these settings generalized to allow for arbitrary communication effects, but specialized to the case in which uncertainty is only in rewards, and only two agents are present, with the querying-answering agent being a degenerate agent whose state merely consists of the duration of time since the currently outstanding query was asked, allowing for a more scalable solution.

A potentially interesting alternative way of framing our problem in the special case of single time step response delay is as a transition-independent Dec-MDP [3], in which one agent controls the CMP and the other agent controls the reward-knowledge process, and the joint reward function depends on the states of both. Whether algorithms for solving transition independent Dec-MDPs can be fruitfully adapted to our setting is a subject we leave for future work.

Lastly, [1] have explored clustering observations in POMDPs. In their work, the aim is to reduce the size of a large observation set offline, which they accomplish by clustering observations according to their emission probabilities across all possible states. Our clustering approach, while related in that we aim to reduce the number of possible posterior beliefs that the agent reasons over, instead clusters possible posterior beliefs the agent might encounter online during the planning stage. Further, our setting is more specialized in that the agent’s uncertainty is in the reward function rather than an arbitrary state space, which we exploit by clustering posterior beliefs according to the distance between their induced mean reward functions.

6. CONCLUSION AND FUTURE WORK

In this paper we investigated the problem of how an agent can act optimally when it has reward uncertainty but can query its human operator to receive a response (after a stochastic delay) that reduces its uncertainty. We formulated a solution to the problem as a joint optimization over the space of queries and transient policies, and provided a Joint Query and Transient Policy (JQTP) algorithm that optimally solves the myopic approximation of this problem by exploiting factoring between the agent’s knowledge state and physical state with respect to queries asked and actions taken. We then proposed a clustering algorithm for flexibly reducing the computation required by JQTP in exchange for performance loss, and presented empirical results showing potential benefits of JQTP and that the clustering algorithm can effectively reduce computation while incurring only small performance loss relative to JQTP.

Our ongoing work is in two directions. First, we are investigating the utility of leveraging further domain structure in order to more effectively cluster query-response pairs, using ideas of influence-based abstractions [16]. Second, we are looking to apply our method to a physical robot and human operator situation. Our long-term efforts are to extend our framework to multiagent sequential decision problems involving more querying agents (vying for the attention of a single human/agent responder), and hence where the policy of the responder (over how to prioritize queries) should be optimized jointly with the policies of the querying agents. Lastly, it would
be interesting to explore other forms of uncertainty (such as in the transition function) in the context of delayed-response queries.

Acknowledgments. We thank the anonymous reviewers for numerous helpful suggestions. This research was supported in part by the Ground Robotics Reliability Center (GRRC) at the University of Michigan, with funding from government contract DoD-DoA W56H2V-04-2-0001 through the US Army Tank Automotive Research, Development, and Engineering Center. UNCLASSIFIED: Dist. A. Approved for public release.

7. REFERENCES


APPENDIX

Here we show the derivation of Equation 3, which represents the expected discounted sum of mean-rewards of the agent which asks query \( q \), follows transient policy \( \pi \), and then acts according to its new knowledge state without asking any more queries. Let the random time at which the query returns be \( \tau \). Let \( \{ s_t, a_t \} \) be the infinite random sequence of state-action pairs generated by following transient policy \( \pi \) until the query returns and following the optimal decision-making policy thereafter, and let \( \{ \psi_\tau \} \) be the infinite random sequence of knowledge states the agent encounters given that it queries \( q \) and then asks no more queries. Note that \( \{ \psi_\tau \} \) is \( \psi \) until the (random) time step at which the query returns, after which it is some (random) \( \psi' \). Then,

\[
Q_\psi(s, q, \pi) = c(q) + \mathbb{E}_{(s_t, a_t) \sim \{ \psi_\tau \}} \left[ \sum_{t=0}^{\infty} \gamma^t T_{\pi_\tau}(s_t, a_t) | s_0 = s, \psi_0 = \psi, q, \pi \right]
\]

\[
= c(q) + \sum_{\tau=1}^{\infty} F_q(\tau) \mathbb{E}_{(s_t, a_t) \sim \{ \psi_\tau \}} \left[ \sum_{t=0}^{\tau-1} \gamma^t T_{\pi_\tau}(s_t, a_t) | s_0 = s, \psi_0 = \psi, q, \pi \right]
\]

\[
+ \sum_{\tau=1}^{\infty} F_q(\tau) \mathbb{E}_{(s_t, a_t) \sim \{ \psi_\tau \}} \left[ \sum_{t=\tau}^{\infty} \gamma^{t-\tau} T_{\pi_\tau}(s_t, a_t) | s_0 = s, \psi_0 = \psi, q, \pi_{\psi'} \right].
\]

The second term is \( r_s^\pi \), defined in the main text. The third term represents the rewards collected by the agent after \( q \) has been returned, at which point it updates its knowledge state to \( \psi' \), never to be updated again (the myopic assumption), and begins selecting actions and queries according to \( \pi^*_{\psi'} \):

\[
\sum_{\tau=1}^{\infty} F_q(\tau) \mathbb{E}_{(s_t, a_t) \sim \{ \psi_\tau \}} \left[ \sum_{t=\tau}^{\infty} \gamma^{t-\tau} T_{\pi_\tau}(s_t, a_t) | s_0 = s, \psi_0 = \psi, \pi_{\psi'} \right]
\]

\[
= \sum_{\sigma' \in S} T^{\psi}(s', s, \pi) \sum_{\psi' \in \Psi(\psi, q)} H_q(\psi') \mathbb{E}_{(s_t, a_t) \sim \{ \psi_\tau \}} \left[ \sum_{t=0}^{\infty} \gamma^t T_{\pi_\tau}(s_t, a_t) | s_0 = s, \psi_0 = \psi, \pi_{\psi'} \right]
\]

where \( T^{\psi}(s', s, \pi) \) is the (random) time step at which the query returns, after which it is some (random) \( \psi' \). Then,

\[
Q_\psi(s, q, \pi) = c(q) + \mathbb{E}_{(s_t, a_t) \sim \{ \psi_\tau \}} \left[ \sum_{t=0}^{\infty} \gamma^t T_{\pi_\tau}(s_t, a_t) | s_0 = s, \psi_0 = \psi, q, \pi \right]
\]

\[
+ \sum_{\tau=1}^{\infty} F_q(\tau) \mathbb{E}_{(s_t, a_t) \sim \{ \psi_\tau \}} \left[ \sum_{t=\tau}^{\infty} \gamma^{t-\tau} T_{\pi_\tau}(s_t, a_t) | s_0 = s, \psi_0 = \psi, q, \pi_{\psi'} \right]
\]

and so finally we have

\[
Q_\psi(s, q, \pi) = c(q) + r_s^\pi(s, \pi) + \sum_{s' \in S} T^\psi(s', s, \pi) \sum_{\psi' \in \Psi(\psi, q)} H_q(\psi') V^\psi_{\psi'}(s').
\]
ABSTRACT

Decentralized partially observable Markov decision processes (Dec-POMDPs) are rich models for cooperative decision-making under uncertainty, but are often intractable to solve optimally (NEXP-complete). The transition and observation independent Dec-MDP is a general subclass that has been shown to have complexity in NP, but optimal algorithms for this subclass are still inefficient in practice. In this paper, we first provide an updated proof that an optimal policy does not depend on the histories of the agents, but only the local observations. We then present a new algorithm based on heuristic search that is able to expand search nodes by using constraint optimization. We also show experimental results comparing our approach with the state-of-the-art Dec-MDP and Dec-POMDP solvers. These results show a reduction in computation time and an increase in scalability by multiple orders of magnitude in a number of benchmarks.

INTRODUCTION

There has been substantial progress with formal models for multiagent sequential decision making represented as decentralized partially observable Markov decision processes (Dec-POMDPs) [12, 14, 5, 2]. Algorithms that are able to exploit domain structure when it is present have been particularly successful [16, 1, 17]. Unfortunately, because the general Dec-POMDP problem is NEXP-complete [6], even these methods cannot solve moderately sized problems optimally.

The decentralized Markov decision process (Dec-MDP) with independent transitions and observations represents a general subclass of Dec-POMDPs that has complexity in NP rather than NEXP [4]. A few algorithms for solving this Dec-MDP subclass have been recently proposed [4, 15]. While these approaches can often solve much larger problems than Dec-POMDP methods, they cannot solve truly large problems or those with more than 2 agents.

In this paper, we present a novel algorithm for optimally solving Dec-MDPs with independent transitions and observations that combines heuristic search and constraint optimization. We show that one can cast any Dec-MDP with independent transitions and observations as a continuous deterministic MDP where states are probability distributions over states in the original Dec-MDP we call state occupancy distributions. This allows us to adapt continuous MDP techniques to solve decentralized MDPs. Following this insight, we designed an algorithm where the state occupancy exploration is performed similarly to learning real-time $A^*$ [9] and the policy selection is in accordance with decentralized POMDP techniques [1, 10]. The result is an approach that is able to leverage problem structure through heuristics, limiting the space of policies that are explored by bounding their value and efficiently generating greedy policies with the use of constraint optimization. This algorithm (termed learning Markov policy or LMP), is shown to be a much more efficient algorithm than any other approach that can be used in Dec-MDPs with independent transitions and observations.

The remainder of this paper is organized as follows. First, we provide a motivating example discussing some properties of Dec-MDPs with independent transitions and observations. Next, we describe the Dec-MDP framework and discuss the related work. We then present theoretical results, showing that the optimal policy for Dec-MDPs with independent transitions and observations does not depend on the agent histories. While this has been proven before, we offer a new perspective on this proof that offers additional insight. Next, we describe the learning decentralized Markov policy algorithm, which combines constraint optimization and heuristic search to more efficiently produce optimal solutions for Dec-MDPs with independent transitions and observations. Finally, we present an empirical evaluation of this algorithm with respect to the state-of-the-art solvers that apply in decentralized MDPs, showing the ability to solve problems that are multiple orders of magnitude larger and those that include up to 6 agents.

Motivating example

To illustrate the characteristics of decentralized partially observable Markov decision processes (Dec-POMDPs) that we are interested in, consider a simple two-agent “meeting-in-a-grid under uncertainty” domain in Figure 1. In this scenario, two agents want to meet as soon as possible on a two-dimensional grid. Each agent’s possible actions include moving north, south, west, east and staying in the same place. The actions of a given agent do not affect the other agents. After taking an action, each agent can sense its own location. Here, each agent’s own partial information is insufficient to determine the global state of the world. This is because agents are not permitted to explicitly communicate their local locations to each other. However, if this (instantaneous and noise-free) communication were allowed the agents’ partial information together would reveal the true state of the world, (i.e., the agents’ joint location). It is the presence of this joint full observability property that differentiates Dec-MDPs and Dec-POMDPs.

More generally, in partially observable models including Dec-POMDPs, the agents’ partial information together can map to multiple different states of the world. As a consequence, decisions in such models depend on the entire past histories of actions and observations the agents ever experienced. In the meeting-in-a-grid under uncertainty problem, since both transitions and observations are independent, each agent’s decision depends only on its last piece of partial information, (i.e., the agent’s own location) [4].
BACKGROUND AND RELATED WORK

In this section, we review the decentralized MDP model, the assumptions of transition and observation independence, the associated notation, and related work.

Definition 1 (The decentralized MDP). A 2-agent decentralized MDP \((S, A, p, \tau)\) consists of: A finite set \(S = Z_1^1 \times Z_2^2\) of states \(s = \langle z_1^1, z_2^2 \rangle\), where \(Z_i^j\) denotes the set of local observations \(z_i^j\) of agent \(i\); A finite set \(A = A_1^1 \times A_2^2\) of joint actions \(a = \langle a_1^1, a_2^2 \rangle\), where \(A_i^j\) is the set of local actions \(a_i^j\) of agent \(i\); A transition function \(p(s, a, s')\), which denotes the probability of transitioning from state \(s = \langle z_1^1, z_2^2 \rangle\) to state \(s' = \langle z_1', z_2' \rangle\) when taking joint action \(a = \langle a_1^1, a_2^2 \rangle\); A reward function \(r: S \times A \rightarrow \mathbb{R}\), where \(r(s, a)\) denotes the reward received when executing joint action \(a\) in state \(s\).

The decentralized MDP is parameterized by the initial state distribution \(\pi_0\). When the agents operate over a bounded number of steps (or the horizon) \(T\), the model is referred to as a finite-horizon decentralized MDP. Solving a decentralized MDP for a given planning horizon \(T\) and start state distribution \(\pi_0\) can be seen as finding \(n\) individual policies that maximize the expected cumulative reward over the steps of the problem.

Additional Assumptions

We are interested in decentralized MDPs that exhibit two assumptions. The first is the transition independence assumption where the local observation of each agent depends only on its previous local observation and the local action taken by that agent.

Definition 2 (The transition independent assumption). A 2-agent decentralized MDP is said to be transition independent if there exists local probability functions \(p^i: Z_i^j \times A_i^j \rightarrow [0, 1]\) and \(p^2: Z_2^2 \times A_1^1 \times Z_1^1 \rightarrow [0, 1]\) such that \(p(s, a, s') = p^1(z_1^1, a_1^1, z_1'^1) \cdot p^2(z_2^2, a_2^2, z_2'^2)\), where \(s = \langle z_1^1, z_2^2 \rangle\) and \(s' = \langle z_1', z_2' \rangle\) and \(a = \langle a_1^1, a_2^2 \rangle\).

We also implicitly assume observation independence, which states that the observation function of each agent does not depend on the dynamics of the other agents. That is, \(P(z_1^1, z_2^2 | s, a, a_2^2) = P(z_1^1 | s, a_1^1)P(z_2^2 | s, a_2^2)\). Because we are assuming a Dec-MDP with the state factored into local observations then this becomes the same as transition independence \(P(z_1^1 | z_1'^1, a_1^1)P(z_2^2 | z_2'^2, a_2^2)\).

Preliminary Definitions and Notations

The goal of solving a Dec-MDP is to find a decentralized deterministic joint policy \(\pi = \langle \pi_1^1, \ldots, \pi_n^i \rangle\). An individual policy \(\pi_i^j\) is a sequence of decision rules \(\pi_i^j = \langle \sigma_0^j, \ldots, \sigma_{T-1}^j \rangle\). In addition, we call decentralized decision rule \(\sigma_t^j\) at time \(\tau\) an \(n\)-tuple of decision rules \((\sigma_0^j, \ldots, \sigma_T^j)\), for \(\tau = 0, 1, \ldots, T - 1\). In this paper, we distinguish between history-dependent and Markov decision rules.

Each history-dependent decision rule \(\sigma_t^j\) at time \(\tau\) maps from \(\tau\)-step local action-observation histories \(h_\tau^j = \langle a_0^j, z_1^1, \ldots, a_{\tau-1}^j, z_{\tau}^2 \rangle\) to local actions: \(\sigma_t^j(h_\tau^j) = a_t^j\), for \(\tau = 0, 1, \ldots, T - 1\). A sequence of history-dependent decision rules defines a history-dependent policy.

In contrast, each Markov decision rule \(\sigma_t^j\) at time \(\tau\) maps from local observations \(z_{\tau}^2\) to local actions: \(\sigma_t^j(z_{\tau}^2) = a_t^j\), for \(\tau = 0, 1, \ldots, T - 1\). A sequence of Markov decision rules defines a Markov policy. Moreover, it is worth noticing that decentralized Markov policies are exponentially smaller than decentralized history-dependent ones.

The state occupancy is another important notion in this paper. The \(\tau\)-th state occupancy of a system under the control of a decentralized Markov policy \((\sigma_0^1, \sigma_1^1, \ldots, \sigma_T^1)\), denoted \(\sigma_0^1, \sigma_1^1, \ldots, \sigma_T^1\), and starting at \(\pi_0\) is given by: \(\eta_{\tau}(s) = P(s | \sigma_0^1, \sigma_1^1, \ldots, \sigma_T^1, \pi_0)\), for all \(\tau \geq 1\). Moreover, the current state occupancy \(\eta_\tau\) depends on the past decentralized Markov policy \((\sigma_0^1, \sigma_1^1, \ldots, \sigma_T^1)\) only through previous state occupancy \(\eta_{\tau-1}\) and decentralized Markov decision rule \(\sigma_\tau\). That is, \(\eta_\tau(s') = \sum_s p(s | \sigma_{\tau-1}, \eta_{\tau-1}(s)) \cdot \eta_{\tau-1}(s)\), for all \(\tau \geq 1\). This update-rule is denoted \(\eta_\tau\) = \(\chi_\tau(\eta_{\tau-1}, \sigma_\tau)\) for the sake of simplicity. We also denote \(\Delta_{\pi_0}\), the state occupancy space at the \(\tau\)-th horizon, that is the standard \([|S|]\)-dimensional simplex.

Distinction with belief states. The state occupancy may be thought of as a belief state, but there are differences. Formally, a belief state \(b_{\tau}\) is given by \(b_{\tau}(s) = P(s | h_\tau^1, \sigma_0^1, \sigma_1^1, \ldots, \sigma_{\tau-1}^1, \pi_0)\), for all \(\tau \geq 1\). That is, in belief states, the information agents have about states is typically conditioned on a single joint action-observation history \(h_\tau^1\). From the total probability property, we then have that \(\eta_{\tau}(s) = \sum_{b_{\tau}} P(s | h_\tau^1, \sigma_0^1, \sigma_1^1, \ldots, \sigma_{\tau-1}^1, \pi_0) \cdot P(h_\tau^1 | \sigma_0^1, \sigma_1^1, \ldots, \sigma_{\tau-1}^1, \pi_0)\). Overall, the \(\tau\)-th state occupancy summarizes all the information about the world states contained in all belief states at horizon \(\tau\). In other words, the doubly exponentially joint action-observation beliefs are summarized in a single state occupancy that does not make use of local information.

Related Work

In this section, we focus on approaches for solving Dec-MDPs with independent transitions and observations as well as other relevant solution methods. For a thorough introduction to solution methods in Dec-POMDPs, the reader can refer to [16, 14, 5, 2].

Becker et al. [4] were the first to describe the transition and observation independent Dec-MDP subclass and solve it optimally. Their approach, called the coverage set algorithm, consists of three main steps. First, sets of augmented MDPs are created which incorporate the joint reward into local reward functions for each agent. Then, all best responses for any of the other agent policies are found using these augmented MDPs. Finally, the joint policy that has the highest value from all agent’s best responses is returned. While this algorithm is optimal, it keeps track of the complete set of policy candidates for each agent, requiring a large amount of time and memory.

Petrik and Zilberstein [15] reformulated the coverage set algorithm as a bilinear program, thereby allowing optimization approaches to be utilized. The bilinear program can be used as an anytime algorithm, providing online bounds on the solution quality at each iteration. The representation is also better able to take advantage of sparse joint reward distributions by representing independent rewards as linear terms and compressing the joint reward matrix. This results in greatly increased efficiency in many cases, but when the agents’ rewards often depend on the other agents the bilinear program can still be inefficient due to lack of reward sparsity.

In general Dec-POMDPs, approximate approaches have attempted to scale to larger problems and horizons by not generating the full set of policies that may be optimal.
These approaches, known as memory-bounded algorithms, were introduced by Seuken and Zilberstein [16] and then successively refined [10]. Memory-bounded algorithms sample forward a bounded number of belief states, and back up (i.e., generate next step policies for) one decentralized history-dependent policy for each belief state. To avoid the explicit enumeration of all possible policies, Kumar and Zilberstein [10] perform the backup by solving a corresponding constraint optimization problem (COP) [7], that represents the decentralized backup. Although, memory-bound techniques are suboptimal, the decentralized backup can be applied in exact settings as we demonstrate when designing our algorithm.

More specifically, the decentralized backup can build a horizon-$\tau$ decentralized policy that is maximal with respect to a belief state and horizon-$(\tau + 1)$ policies available for each agent. The associated COP is given by: a set of variables, one for each local observation of each agent; a set of domains, where the domain for the variables corresponding to an agent is the set of horizon-$(\tau + 1)$ policies available for that agent; a set of soft constraints, one for each joint observation. The soft constraint maps assignments to real values. Intuitively, these values represent the expected reward accrued when agents together perceive a given joint observation and follow a given horizon-$(\tau + 1)$ decentralized policy. Since horizon-$\tau$ decentralized policies consist of horizon-$(\tau + 1)$ policies, it is easy to see that maximizing the sum of the soft constraints yields a maximal horizon-$\tau$ decentralized policy.

Closer to our model is the ND-POMDP framework [13]. It aims at modeling multiagent teamwork where agents have strong locality of interaction, often through binary interactions. That is, the reward model in such domains is decomposed among sets of agents. There has been a substantial body of work that extend general Dec-POMDP techniques (discussed above) to exploit the locality of interaction [13, 11]. Nair et al. [13] introduced the only optimal algorithm for this model, namely General Optimal Algorithm (GOA). When the domain does not contain binary interactions, there is no reason to expect GOA to outperform general Dec-POMDP algorithms, as all methods use similar strategy in selecting policy candidates. However, when the domain contain significant binary interactions, GOA will outperform general Dec-POMDP algorithms.

It is worth noting that ND-POMDPs and transition and observation independent Dec-MDPs make the same assumptions about transition and observation independence, but make different assumptions about the reward model and partial observability. More specifically, ND-POMDPs assume the reward can be decomposed into the sum of local reward models for sets of agents, while the reward model for transition and observation independent Dec-MDPs is more general, allowing global rewards for all agents (i.e., considering all agents in one set). Dec-MDPs assume that the state is jointly fully observable (that the state if fully determined by the combination of local observations of all agents), while ND-POMDPs do not make this limiting assumption. Both models therefore make different assumptions to address complexity and the choice of model depends on which assumptions best match the domain being solved.

THEORETICAL PROPERTIES

In this section, we demonstrate the main theoretical results of this paper.

A decentralized MDP solver aims to calculate an optimal decentralized policy $\pi^*$ that maximizes the expected cumulative reward:

$$\pi^* = \arg\max_\pi \mathbb{E}[\sum_{t=0}^{\tau-1} r(s_t, a_t) | \pi, \pi_0].$$

(1)

The following theorem proves that decentralized Markov policies yield the optimal performance in decentralized MDPs with independent transitions.

THEOREM 1 (OPTIMALITY OF MARKOV POLICIES). In decentralized MDPs with transition independence, optimal policies for each agent depend only on the local state and not on agent histories.

PROOF. Without loss of generality, we construct a proof by induction for two agents, 1 and 2, from agent 1’s perspective. We first show that on the last step of the problem, agent 1’s policy does not depend on its local history.

While proof of this theorem has already been shown [8], we prove it in a different manner that more directly relates policies to values. This may also be more clear to some readers.

Let $h_1^\tau = (a_1^0, z_1^1, \ldots, a_1^\tau, z_1^{\tau+1})$ be a local action-observation history of agent $i = 1, 2$ at step $\tau$. Agent 1’s local policy on the last step is then: $\sigma_{1}^{\tau-1}(h_{1}^{\tau-1}) = \arg\max_{a_1^\tau} \sum_{h_1^{\tau}} P(h_{1}^{\tau} | h_{1}^{\tau-1}) \cdot R(s, a_1, \sigma_{1}^{\tau-1}(h_{1}^{\tau-1}))$, which chooses a local action to maximize value based on the possible local histories of agent 2 and resulting states of the system $s = (z_{1}^{\tau-1}, z_2^{\tau+1})$.

Based on transition and observation independence and the use of decentralized policies, it can be shown that $P(h_2^{\tau} | h_1^{\tau-1}) = P(h_2^{\tau-1} | h_1^{\tau-1})$. Due to space limitations, we do not include full proof of this claim. Intuitively it holds because each agent does not receive any information about the other agents’ local histories because of transition independence. Therefore, we can represent agent 1’s policy on the last step as $\sigma_{1}^{\tau}(h_{1}^{\tau-1}) = \arg\max_{a_1^\tau} \sum_{h_1^{\tau}} P(h_{1}^{\tau} | h_{1}^{\tau-1}) \cdot R(s, a_1, \sigma_{1}^{\tau-1}(h_{1}^{\tau-1}))$ which no longer depends on the history $h_2^{\tau-1}$. Therefore, the policy on the last step for either agent does not depend on history.

This allows us to define the value function on the last step as $v_{\pi_{1}}(s_{\pi_{1}}(h_{1}^{\tau-1}), \sigma_{1}^{\tau-1}(h_{1}^{\tau-1}))$. Then for the induction step, we can show that if the policy at step $\tau + 1$ does not depend on history, then the policy at step $\tau$ also does not depend on its local history. Again, we show this from agent 1’s perspective.

Agent 1’s policy on step $\tau$ can be represented by: $\sigma_{1}^{\tau}(h_{1}^{\tau}) = \arg\max_{a_1^\tau} \sum_{h_1^{\tau}} P(h_{1}^{\tau} | h_{1}^{\tau-1}) \cdot v_{\pi_{1}+1}(s, a_1^\tau, \sigma_{1}^{\tau}(h_{1}^{\tau}))$, where the value function $v_{\pi_{1}+1}$ is assumed to not depend on history. We can again show that $P(h_2^{\tau} | h_1^{\tau}) = P(h_2^{\tau})$ because of transition independence and represent agent 1’s policy on step $\tau$ as:

$$\sigma_{1}^{\tau}(h_{1}^{\tau}) = \arg\max_{a_1^\tau} \sum_{h_1^{\tau}} P(h_{1}^{\tau} | h_{1}^{\tau-1}) \cdot v_{\pi_{1}+1}(s, a_1^\tau, \sigma_{1}^{\tau}(h_{1}^{\tau}))$$

which no longer depends on the local history $h_2^{\tau}$. Therefore, the policy of either agent does not depend on local history for any step of the problem. □

We now establish the sufficient statistic for the selection of decentralized Markov decision rules.

THEOREM 2 (SUFFICIENT STATISTIC). The state occupancy is a sufficient statistic for decentralized Markov decision rules.

PROOF. We build upon the proof of the optimality of decentralized Markov policies in Theorem 1. We note that an optimal decentralized Markov policy starting in $\pi_0$ is given by:

$$\pi^* = \arg\max_{\pi} \sum_{s_\pi} \sum_{\omega_\tau} P(\omega_\tau | \sigma_0, \pi_0, \eta_\pi) \cdot r(s_\pi, \sigma_\pi(s_\pi)).$$

The substitution of $\omega_\tau$ by $\langle \omega_{\tau-1}, s_\pi \rangle$ plus the sum over all joint observation-histories $\omega_{\tau-1}$ yields

$$\pi^* = \arg\max_{\pi} \sum_{s_\pi} \sum_{\omega_{\tau-1}} \sum_{s_\pi} P(\sigma_\pi | \sigma_0, \pi_0, \eta_\pi) \cdot r(s_\pi, \sigma_\pi(s_\pi)).$$

We denote $\eta_\pi = P(\sigma_\pi | \sigma_0, \pi_0, \eta_\pi)$ the state occupancy distribution decentralized Markov $\pi$ produced at horizon $\tau$. And hence,

$$\pi^* = \arg\max_{\pi} \sum_{s_\pi} \sum_{s_\pi, \in S} \eta_\pi(s_\pi) \cdot r(s_\pi, \sigma_\pi(s_\pi)).$$
So, state occupancy $\eta_\pi^\tau$ summarizes all possible joint observation-histories $\omega_\tau$ decentralized Markov policy $\pi$ produced at horizon $\tau$ for the estimate of joint decision rule $\sigma_\tau$. Thus, the state occupancy is a sufficient statistic for decentralized Markov decision rules since their estimates depend only upon a state occupancy, and no more on all possible joint observation-histories.

States, belief states, and multi-agent belief states are all sufficient to select directly actions for MDPs, POMDPs, and decentralized POMDPs, respectively. This is mainly because all these statistics summarize the information about the world states from a single agent perspective. The state occupancy, instead, summarizes the information about the world states from the perspective of a team of agents, that are constrained to execute their policies independently from each other. In such a setting, joint actions cannot be selected independently, instead, they are selected jointly through decentralized Markov decision rules.

With these results as a background, we define the value function of decentralized Markov policy $\pi$ as follows:

$$u_\tau(\eta_\pi) = E_{(\eta_\pi, \cdots, \eta_{\tau - 1})} \left[ \sum_{s=0}^{T-1} r_s(\eta_\sigma, \sigma_\tau) \mid \eta_\pi \right],$$

where the $\tau$-th reward function $r_\tau(\eta_\sigma, \sigma_\tau)$ is given by $r_\tau(\eta_\sigma, \sigma_\tau) = E_{\eta_\sigma} [r(s, \sigma_\tau(s))]$. We then define the $\tau$-th value function $v_{\eta_\pi} = E_{\eta_{\tau + 1}} \left[ \sum_{s=0}^{T-1} \Delta_s(\eta_\sigma, \sigma_\tau) \right]$ as follows:

$$v_{\eta_\pi} = r_s(\eta_\sigma, \sigma_\tau) + \sum_{s=0}^{T-1} \Delta_s(\eta_\sigma, \sigma_\tau),$$

where quantity $u_{\eta_\pi}$ denotes the expected sum of rewards attained by starting in state occupancy $\eta_\pi$, taking one joint action according to $\sigma_\tau$, and executing the new joint action according to $\sigma_{\tau+1}$, and so on. We slightly abuse notation to write the $\tau$-th value function under the control of an “unknown” decentralized Markov policy $\sigma_{\tau+1}$ using $v_{\eta_\pi}$, $\Delta_s(\eta_\sigma, \sigma_\tau)$. We further note $\mathcal{V}_s$ the space of bounded value functions at the $\tau$-th horizon. As such, the $\tau$-th value function $v_\tau$ can be built from a $(\tau + 1)$-th value function $v_{\tau+1}$ as follows:

$$v_\tau(\eta_\sigma) = \max_{\sigma_\tau} \left[ \mathcal{L}_\sigma v_{\tau+1}(\eta_\sigma) \right],$$

using backup operator $\mathcal{L}_\sigma, \mathcal{V}_s \mapsto \mathcal{V}_s$ such that $\left[ \mathcal{L}_\sigma v_{\tau+1} \right](\eta_\sigma) = r_s(\eta_\sigma, \sigma_\tau) + \max_{\sigma_\tau} \left[ \mathcal{L}_\sigma v_{\tau+1}(\eta_\sigma, \sigma_\tau) \right]$. In our setting, Equations (2) denote the optimality equations, where $v_\tau(\eta_\sigma) = 0$. The decentralized Markov policy $\pi = (\sigma_0, \cdots, \sigma_{\tau-1})$ returned is greedy with respect to value functions $u_{\eta_0}, \cdots, u_{\eta_{\tau-t}}$.

**LEARNING MARKOV POLICY**

In this section, we compute optimal decentralized Markov policy $(\sigma_0, \cdots, \sigma_{\tau-1})$ given initial state occupancy $\eta_\pi$ and planning horizon $T$. Note that while state occupancies are used to calculate heuristics in this algorithm, the choices at each step do not depend on the state occupancies. That is, the result is a nonstationary policy for each agent mapping local observations to actions at each step.

We cast decentralized MDPs $(S, A, p, r)$ as continuous and deterministic MDPs where: states are state occupancy distributions $\eta_\pi$; actions are decentralized Markov decision rules $\sigma_\tau$; the update-rules $\chi_{\tau}(\cdot, \sigma_{\tau-1})$ define transitions; and mappings $r_\tau(\cdot, \sigma_{\tau})$ denote the reward function. So, techniques that apply in continuous and deterministic MDPs also apply in decentralized MDPs. For the sake of efficiency, we focus only on optimal techniques that exploit the initial information state $\eta_\pi$.

The learning real-time $A^*$ (LRTA*) algorithm can be used to solve deterministic MDPs [9]. This approach updates only states that agents actually visit during the planning stage. Therefore, it is suitable for continuous state spaces. Algorithm 1, namely learning Markov policy (LMP), illustrates an adaptation of the LRTA* algorithm for solving decentralized MDPs. The LMP algorithm relies on lower and upper bounds $\bar{v}_\tau$ and $\bar{v}_\tau$, on the exact value functions for all planning horizons $\tau = 0, \cdots, T - 1$.

We use the following definitions. $Q$-value functions $q_\tau(\eta_\sigma, \sigma_\tau)$ denote rewards accrued after taking decision rule $\sigma_\tau$ at state occupancy $\eta_\sigma$ and then following the policy defined by upper-bound value functions for the remaining planning horizons. We denote $\Psi_\tau(\eta_\sigma) = \{\sigma_\tau\}$ the set of all stored decentralized Markov decision rules for state occupancy $\eta_\sigma$. Thus, $\bar{v}_\tau(\eta_\sigma) = \max_{\tau_\sigma \in \Psi_\tau(\eta_\sigma)} q_\tau(\eta_\sigma, \tau_\sigma)$ represents the upper-bound value at state occupancy $\eta_\sigma$. Formally, we have that $q_\tau(\eta_\sigma, \sigma_\tau) = [\bar{v}_\tau, \bar{v}_{\tau+1}](\eta_\sigma)$.

Next, we describe two variants of the LMP algorithm. The exhaustive variant replaces states by state occupancy distributions, and actions by decentralized Markov decision rules in the LRTA* algorithm. The second variant uses a constraint optimization program instead of the memory demanding exhaustive backup operation that both the LRTA* algorithm and the exhaustive variant use.

**The exhaustive variant**

The exhaustive variant consists of three major steps: the initialization step (line 1); the backup operation step (line 5); and the update step (lines 6 and 8). It repeats the execution of these steps until convergence ($\bar{v}_\tau(\eta_\sigma) - \bar{v}_\tau(\eta_\sigma) > \epsilon$). At this point, an optimal decentralized Markov policy has been found.

**Algorithm 1: The LMP algorithm.**

1. Initialize bounds $\bar{v}$ and $\bar{v}$.
2. while $\bar{v}(\eta_\sigma) - \bar{v}_\tau(\eta_\sigma) > \epsilon$ do
3. LMP-TRIAL$(\eta_\sigma)$
4. while $\bar{v}_\tau(\eta_\sigma) - \bar{v}_\tau(\eta_\sigma) > \epsilon$ do
5. $\sigma_{\text{greedy}, \tau} \leftarrow \arg\max_{\sigma_\tau} q_\tau(\eta_\sigma, \sigma_\tau)$
6. Update the upper bound value function.
7. LMP-TRIAL$(\eta_\sigma, \sigma_{\text{greedy}, \tau})$
8. Update the lower bound value function.

**Initialization.** We initialize lower bound $\bar{v}_\tau$ with the $\tau$-th value function of any decentralized Markov policy, such as a randomly generated policy $\pi_{\text{rand}} = (\pi_{\text{rand}}, \cdots, \pi_{\text{rand}, T-1})$, where $v_\tau \leq v_{\pi_{\text{rand}}, \cdots, \pi_{\text{rand}, T-1}}$. We initialize the upper bound $\bar{v}_\tau$ with the $\tau$-th value function of the underlying MDP. That is, $\pi_{\text{mdp}} = (\pi_{\text{mdp}}, \cdots, \pi_{\text{mdp}, T-1})$. We use $\bar{v}_\pi = v_{\pi_{\text{rand}}, \cdots, \pi_{\text{rand}, T-1}}$.

**The exhaustive backup operation.** We choose decentralized Markov decision rule $\sigma_{\text{greedy}, \tau}$, which yields the highest value $\bar{v}_\tau(\eta_\sigma)$ through the explicit enumeration of all possible decentralized Markov decision rules $\sigma_\tau$. We first store all decentralized Markov decision rule $\sigma_\tau$ for each visited state occupancy $\eta_\sigma$ together with corresponding value $q_\tau(\eta_\sigma, \sigma_\tau)$. Hence, the greedy decentralized Markov decision rule $\sigma_{\text{greedy}, \tau}$ is $\arg\max_{\sigma_\tau} q_\tau(\eta_\sigma, \sigma_\tau)$ at state occupancy $\eta_\sigma$.

**Update of lower and upper bounds.** We update the lower bound value function based on decentralized Markov policies $\pi_{\text{greedy}} = (\pi_{\text{greedy}, 0}, \cdots, \pi_{\text{greedy}, T-1})$ selected at each trial. If $\pi_{\text{greedy}}$ yields a value higher than that of the current lower bound, $\bar{v}_\tau(\eta_\sigma) < v_{\pi_{\text{greedy}}(\eta_\sigma)}$, we set $\bar{v}_\tau = v_{\pi_{\text{greedy}}, \cdots, \pi_{\text{greedy}, T-1}}$ for $\tau = 0, \cdots, T - 1$, otherwise we leave the lower bound unchanged. We update the upper bound value function based on decentralized Markov decision rules $\sigma_{\text{greedy}, \tau}$ and the $(\tau + 1)$-th upper-bound value function $\bar{v}_{\tau+1}$, as follows: $\bar{v}_{\tau+1}(\eta_\sigma) = [\bar{v}_{\text{greedy}, \tau+1}, \bar{v}_{\tau+1}](\eta_\sigma)$.

**Theoretical guarantees.** The exhaustive variant of LMP yields both advantages and drawbacks. On the one hand, it inherits the-
oretical guarantees from the LRTA* algorithm. In particular, it ter-
minates with a decentralized Markov policy within ε never underestimate the exact value
at any state occupancy ητ. This is because we update the upper bound value at each state occupancy based upon a greedy decision
rule for this state occupancy. On the other hand, the exhaustive vari-
ant algorithm requires the exhaustive enumeration of all possible
decentralized Markov decision rules at each backup (Algorithm 1, line 5). In MDP techniques, the exhaustive enumeration is not pro-
hibitive since the action space is often manageable. In decentralized
MDP planning, however, the space of all decentralized Markov de-
cision rules increases exponentially with increasing observations.
So, the exhaustive variant can scale only to problems with a few observations.

The constraint optimization formulation

To overcome the memory limitation of the exhaustive variant, we use constraint optimization instead of the exhaustive backup oper-
ation. More precisely, our constraint optimization program returns
a greedy decentralized Markov decision rule σgreedy,τ for each state
occupancy ητ visited, but without performing the exhaustive enu-
meration.

In our constraint optimization formulation, variables are associ-
ated with σi∗(zτ) for all agents i = 1, . . . , n and all local observ-
ations zτ ∈ Zτ. The domain for each variable σi∗(zτ) is action
space Aτ. For each state s ∈ S, we associated a single soft con-
straint cvi(s, ·) : A → R. Each of which assigns a value cvi(s, a) =
ητ(s asymptotic to) + ητ(s, a) · rmdp,1 · ητ(a, s′) . . . · ητ(a, sτ − 1) · ητ(s τ). The objective of our constraint
optimization model is to find an assignment σgreedy,τ of actions aτ
variables σi∗(zτ) such that the aggregate value is maximized.
Stated formally, we wish to find σgreedy,τ = arg maxσa τ ctv(s, ·) · cσi(s, ·) . . . · cστ(s, ·)�τ · ητ(s τ). The objective of our constraint
optimization program, note that by the definition of mapping qmdp we have
that ητ(s τ) = cσi(s, ·) · cστ(s, ·) . . . · cστ(s, ·) . . . · cστ(s τ). Hence, if we use
qmdp(ητ, ·) instead of ητ, workforce on the decision to apply cσi(s, ·)
Thus, our constraint optimization program
returns a decentralized Markov decision rule with the highest upper-
bound value. Techniques that solve our constraint optimization
formulation abound in the literature of constraint programming [7].

Theoretical guarantees. The constraint optimization variant yields
the same guarantees as the exhaustive variant without the major
drawback. Indeed, it replaces the exhaustive enumeration of all de-
centralized Markov decision rules. Instead, it uses a constraint op-
timization formulation that returns a greedy decentralized Markov
decision rule. And hence, cutting off the algorithm at anytime, the
solution is within ε = η0(τ n) − η0(τ 0) of an optimal decentralized
Markov policy.

Comparison to the CBPB algorithm. Kumar and Zilberstein
[10] were the first to replace the exhaustive backup operation by a
constraint optimization formulation in decentralized control set-
tings using an algorithm called CBPB. Their constraint optimiza-
tion program computes a decentralized history-dependent policy
for a given belief state. We adapted this formulation to calculate a
greedy decentralized Markov decision rule given a state occu-
pancy. While many other alternatives to the exhaustive backup oper-
ation exist, including branch-and-bound techniques [1], so far the
constraint optimization formulation is the most efficient in practice
[10].

Next, we provide an execution of the learning Markov policy
algorithm using our motivating example.

Meeting-in-a-grid example

Figure 2 demonstrates the main steps of the algorithm for a meeting-
in-a-grid scenario on a 2×2 grid (as shown in Figure 3a) over 4
planning horizons. The action transitions are noisy and reward is
0 unless both agents are in the same square and choose a stay
action. We depicted the heuristic values (upper and lower bounds) for
joint locations of the agents and each planning horizon in 4×4
grids (with states as shown in Figure 3b), one grid for each plan-
ning horizon. We illustrate each trial with two sequences of grids in
Figure 2: the first sequence (in blue) represents upper-bound val-
ues when the LMP algorithm traverses the search tree forward; the
second sequence (in orange) represents lower-bound values when
the LMP algorithm traverses the policy generated during the for-
ward search backward. The LMP algorithm starts with a uniform
state occupancy η0 where both agents may be in all possible loca-
tions, that is η0(τ) ≃ 0.00625 initially, and the upper-bound value
function is the MDP value function.

![Figure 3: The 2×2 grid (a) denotes all locations of each agent. The 4×4 grid (b) represents the joint locations of the agents.](image-url)
Figure 2: The learning Markov policy planning phase for a meeting-grid under uncertainty scenario on a $2 \times 2$ grid over 4 planning horizons. The node grids illustrated here denote heuristic value functions: upper bound value functions are blue grids; lower-bound value functions are orange grids. The dashed grid represents a node where the stopping criterion is satisfied. Grids of the initial upper-bound value function are labelled by the name of the hyperplane at each planning horizon. The remaining nodes are labelled by their value at the current state occupancy.
The LMP algorithm then updates the lower-bound value functions backward. At horizon $\tau = 3$, the lower-bound value function is such that $\bar{v}_2(s) = r(s, \bar{\sigma}_2[s])$, which is equal to the upper-bound value function at horizon $\tau = 3$ no matter the state occupancy distribution. This is specific to the problem at hand because, in general, it might be the case that a decision rule achieves good performance for one state occupancy but performs poorly for another. Since both lower and upper bounds at horizon $\tau = 3$ are equal for all states, the lower-bound value function $\bar{v}_3$ is the optimal value function at horizon $\tau = 3$. As such, the LMP algorithm in the succeeding trials will no longer perform updates on horizon $\tau = 3$, since we already have the optimal solution. At horizon $\tau = 2$, we update the lower-bound value function at state occupancy $\eta_2$ as follows: $\bar{v}_2(\eta_2) = r(\eta_2, \bar{\sigma}_2) + \bar{v}_3(\chi(\eta_2, \bar{\sigma}_2)) \simeq 1.88$, and the lower bound is calculated in a similar way for all states. Note that while $\bar{v}_2(\eta_2) = \bar{q}_2(\eta_2, \sigma_2)$, however, the upper and lower bound value functions are not equal for all state occupancy distributions (and thus an optimal solution for this horizon has not been determined). A horizon $\tau = 1$ and $\tau = 0$, the algorithm updates the lower-bound values similarly to horizon $\tau = 2$. The upper-bound value at $\eta_0$ is given by $\bar{v}_0(\eta_0) \simeq 2.68$. At the first trial, we have that $\bar{v}_0(\eta_0) = \bar{q}_0(\eta_0, \sigma_0) \simeq 0.061$. As such, the LMP algorithm performs another trial since upper and lower bounds are not within a sufficiently small scalar.

**Second Trial.** During the second trial, the agents coordinate to meet at another location than the one used in the first trial. This is because the upper-bound value will decrease in previously visited state occupancy distributions and joint decision rules, but not among unvisited state occupancies. To better understand this, consider planning for horizon $\tau = 0$ as the algorithm proceeds similarly for subsequent horizons. The LMP algorithm distinguishes between joint decision rules whose next state occupancy $\chi(\eta_0, \sigma_0) \in \Delta$ has already been visited in previous trials and the other joint decision rules. In selecting the greedy joint decision rule, the LMP algorithm selects the joint decision rule that maximizes $r(\eta_0, \sigma) + \bar{v}_1(\chi(\eta_0, \sigma))$. There is only one joint decision rule whose successor state occupancy is already visited: the joint decision rule selected during trial 1 at horizon $\tau = 0$, and denoted $\sigma_0$. Recall that, its value was $\bar{q}_0(\eta_0, \sigma_0) \simeq 2.74$. However, since we know its successor state occupancy $\chi(\eta_0, \sigma_0) = \eta_1$, it is updated as follows: $\bar{q}_0(\eta_0, \sigma_0) = r(\eta_0, \sigma_0) + \bar{v}_1(\chi(\eta_0, \sigma_0)) = r(\eta_0, \sigma_0) + \bar{v}_1(\eta_1) \simeq 0.25 + 2.44332975$

At trial 2, $\sigma_0$ achieves reward $\bar{q}_0(\eta_0, \sigma_0) \simeq 2.68$ which is lower than at trial 1. This is because the upper-bound value at $\eta_1$ is tighter than that at the initial trial. The agents then choose to coordinate to meet at a different location. They can choose location 1, 2 or 3, but here we assume they chose location 1. To do so, they take action stay at location 1; they take action up at locations 0 and 3; and they take action left at location 3. This decision rule is greedily selected at horizon $\tau = 0$, 1 and 2. The decision rules are chosen to allow agents to progressively increase their probability to meet at location 1. At horizon $\tau = 3$, the agents take action stay no matter the location. Besides the greedy selection of decision rules, the LMP algorithm updates both the upper and lower bound value functions. The update of the upper-bound value function is illustrated in Figure 2, at horizon $\tau = 0$, since $\chi(\eta_0, \sigma_0)$ is visited for the first time, we have that $\bar{q}_0(\eta_0, \sigma_0) = r(\eta_0, \sigma_0) + \bar{v}_1(\chi(\eta_0, \sigma_0)) = r(\eta_0, \sigma_0) + v_{\text{up}}(\eta_1) \simeq 0.25 + 2.49$. That is, $\bar{q}_0(\eta_0, \sigma_0) \simeq 2.74$. Because $\bar{q}_0(\eta_0, \sigma_0) > \bar{q}_0(\eta_0, \sigma_0)$, the LMP algorithm can replace the joint decision rule $\sigma_0$ selected during the first trial by joint decision rule $\bar{\sigma}_0$. The LMP algorithm proceeds similarly at horizons $\tau = 1$ and 2. Notice that the second trial ends at horizon $\tau = 2$, because the lower-bound value function at horizon $\tau = 3$ is the optimal. The LMP algorithm then proceeds to update the lower-bound value function backward. It starts at horizon $\tau = 2$, then $\tau = 1$, down to $\tau = 0$. While the lower-bound values for visited state occupancy distributions during trial 1 and 2 achieve the same performance, lower-bound value functions are different. This means that both sequences of joint decision rules yield equivalent value. The upper-bound value at $\eta_0$ is given by $\bar{v}_0(\eta_0) \simeq 2.68$. At the second trial, we still have that $\bar{v}_0(\eta_0) - \bar{v}_0(\eta_0) = 0.061$. As such, the LMP algorithm needs to perform succeeding trials. In the remaining trials, the LMP algorithm selects a joint decision rules that allows agent to coordinate to meet on a single location 0, 1, 2 or 3 that was not selected so far. This results in cycling through the possible locations and tightening the bounds until a final solution is found after 12 trials.

**EMPIRICAL EVALUATIONS**

We evaluated our algorithm using many benchmarks from the decentralized MDP literature. For each benchmark, we compared our algorithms with state-of-the-art algorithms for decentralized MDPs. We report on each benchmark the best recorded value $v_0(\eta_0)$ and the running time in seconds for different planning horizons. The LMP variants were ran on a Mac OSX machine with 2.4GHz Dual-Core Intel and 2GB of RAM available. We solved the constraint optimization problems using the aolib library ¹. The bilinear programming approach (listed as BLP) was run on a 2.8GHz Quad-Core Intel Mac with 2GB of RAM with a time limit of 3 hours. We used the best available version of the bilinear program approach which was the iterative best response version with standard parameters. This is a generic solution method which does not perform as well as the more specialized approaches in [15], but we do not expect results to differ by more than a single order of magnitude. We do not compare to the coverage set algorithm because the bilinear programming methods have been shown to be more efficient for all available test problems.

We provide values for the exhaustive variant, exh, on small problems and constraint optimization formulation, COP, for all problems. We tested our algorithms on six benchmarks: recycling robot, meeting grid 3x3 and 8x8; and navigation problems². These are the largest and hardest benchmarks we could find in the literature. We compare our algorithms with: GMAA*-ICE, IPG, and BLP. The GMAA*-ICE [17] heuristic search consistently outperforms other generic exact solvers such as MAA*. GMAA*. The IPG [1] algorithm is a competitive alternative to the GMAA* approach and performs well on problems with reduced reachability.

In all benchmarks, the COP variant outperforms the other algorithms. The results, illustrated in Table 1, show that the COP variant produces the optimal policies in much less runtime for all tested benchmarks. As an example consider the meeting in a 3x3 grid for $T = 5$: the COP variant computed the optimal policies in about 33,4358 and 4580 times faster than the GMAA*-ICE, BLP and IPG algorithms, respectively. We also note that the COP variant is very useful for the medium and large domains. For example all large domains, the exh. variant ran out of memory while the COP variant computed the optimal solutions over hundreds of horizons. Yet, the exh. variant can compute the optimal solution of small problem faster than the COP variant. For example in the recycling robot at horizon $T = 1000$, the exh. variant computed the optimal solution in about 5 times faster than the COP variant in all of LMP algorithm.

While preliminary, we also note that the constraint formulation, allows us to deal with larger numbers of agents. Indeed, we ran

¹The aolib library is available at http://graphmod.ics.uci.edu/group/aolibWCSP/
²All problem definitions are available at http://users.isr.ist.utl.pt/~mtjspaan/decpomdp/
the COP variant of LMP on decentralized MDPs with independent transitions and observations where each agent’s local MDP is based on the recycling-robot problem. The COP variant was able to scale up to 6 agents at horizon 10 in about 10,000 seconds. Despite this high running time, LMP is the first generic algorithm that scales to teams of more than two agents. For example the BLP algorithm as it currently stands can only solve two-agent problems.

There are many different reasons for these results. The LMP outperforms GMMA+-ICE and IPG mainly because they perform a policy search in the space of decentralized history-dependent policies. Instead, the LMP algorithm performs its policy search in the space of decentralized Markov policies, which is exponentially smaller than that of the decentralized history-dependent policies. The LMP outperforms the BLP algorithm mainly because of the dimensionality of its sufficient statistic. Indeed, the BLP algorithm uses a sufficient statistic that is \( T \) times larger. More specifically, the number of bilinear terms in the BLP approach grows polynomially in the horizon of the problem, causing it to not perform well for large problems and large horizons with tightly coupled reward values.

### CONCLUSION AND FUTURE WORK

This paper explores new theory and algorithms for solving independent transition and observation Dec-MDPs. We provide a new proof that optimal policies do not depend on agent histories in this subclass. We also describe a novel algorithm that combines heuristic search and constraint optimization to more efficiently produce solutions. This new algorithm, termed learning Markov policy or LMP, was shown to scale up to large problems and planning horizons, reducing computation time by multiple orders of magnitude over previous approaches.

In the future, we plan to extend the existing LMP algorithm to other classes of problems and larger teams of agents. For instance, we may be able to produce an optimal solution to more general classes of Dec-MDPs or provide approximate results for Dec-POMDPs. Furthermore, the scalability of our approach to larger numbers of agents is encouraging and we will pursue methods to increase this even further.

### ACKNOWLEDGEMENTS

The authors would like to thank Frans Oliehoek, Marek Petrik and the anonymous reviewers for their helpful feedback.

### REFERENCES


Automated Equilibrium Analysis of Repeated Games with Private Monitoring: A POMDP Approach

YongJoon Joe1, Atsushi Iwasaki1, Michihiro Kandori2, Ichiro Obara3, and Makoto Yokoo1
1: Kyushu University, Japan, {yongjoon@agent, iwasaki@, yokoo}@inf.kyushu-u.ac.jp
2: University of Tokyo, Japan, kandori@e.u-tokyo.ac.jp,
3: UCLA, California, ichiro.obara@gmail.com

ABSTRACT
The present paper investigates repeated games with imperfect private monitoring, where each player privately receives a noisy observation (signal) of the opponent’s action. Such games have been paid considerable attention in the AI and economics literature. Since players do not share common information in such a game, characterizing players’ optimal behavior is substantially complex. As a result, identifying pure strategy equilibria in this class has been known as a hard open problem. Recently, Kandori and Obara (2010) showed that the theory of partially observable Markov decision processes (POMDP) can be applied to identify a class of equilibria where the equilibrium behavior can be described by a finite state automaton (FSA). However, they did not provide a practical method or a program to apply their general idea to actual problems. We first develop a program that acts as a wrapper of a standard POMDP solver, which takes a description of a repeated game with private monitoring and an FSA as inputs, and automatically checks whether the FSA constitutes a symmetric equilibrium. We apply our program to repeated Prisoner’s dilemma and find a novel class of FSA, which we call k-period mutual punishment (k-MP). The k-MP starts with cooperation and defects after observing a defection. It restores cooperation after observing defections k-times in a row. Our program enables us to exhaustively search for all FSAs with at most three states, and we found that 2-MP beats all the other pure strategy equilibria with at most three states for some range of parameter values and it is more efficient in an equilibrium than the grim-trigger.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Multi-agent systems; J.4 [Social and Behavioral Sciences]: Economics

General Terms
Algorithms, Economics, Theory

Keywords
Game theory, repeated games, private monitoring, POMDP

1. INTRODUCTION

We consider repeated games with imperfect private monitoring, where each player privately receives a noisy observation (signal) of the opponent’s action. This class of games represents long-term relationships among players and has a wide range of applications, e.g., secret price cutting and agent planning under uncertainty. Therefore, it has been paid considerable attention in the AI and economics literature. In particular, for the AI community, the framework has become increasingly important for handling noisy environments. In fact, Ng and Seah examine protocols in multi-hop wireless networks with self-interested agents [9], and Tennenholtz and Zohar consider repeated congestion games where an agent has limited capability in monitoring the actions of her counterparts [12].

Analytical studies on this class of games have not been quite successful. The difficulty comes from the fact that players do not share common information under private monitoring, and finding pure strategy equilibria in such games has been known as a hard open problem [8]. Under private monitoring, each player cannot observe the opponents’ private signals, and he or she has to draw statistical inferences about the history of the opponents’ private signals. The inferences quickly become very complicated over time, even if players adopt relatively simple strategies [5]. As a result, finding a profile of strategies which are mutual best replies after any history, i.e., finding an equilibrium, is a quite demanding task.

Quite recently, Kandori and Obara show that the theory of the partially observable Markov decision process (POMDP) can be used to identify equilibria, when equilibrium behavior is described by a finite state automaton (FSA) [6]. This result is significant since it implies that by utilizing a POMDP solver, we can systematically determine whether a given profile of finite state automata can constitute an equilibrium. Furthermore, this result is interesting since it connects two popular areas in AI and multi-agent systems, namely, POMDP and game theory.

Traditionally, in the AI literature, the POMDP framework is a popular approach for single-agent planning/control, and game theory has been extensively used for analyzing multi-agent interactions. However, these two areas have not been well-connected so far, as mentioned in the most recent edition of a popular AI textbook “... game theory has been used primarily to analyze environments that are at equilibrium, rather than to control agents within an environment” [11]. As one notable exception, Doshi and Gmytrasiewicz investigate the computational complexity and subjective equi-
In a subjective equilibrium, a player may not perfectly know the opponent’s strategy. As a result, the definition of a subjective equilibrium is involved, and they show that reaching a subjective equilibrium is difficult under the limit of computational complexity. In contrast, Kandori and Obara examine if simple behavior described by FSA can be mutual best replies [6]. They proposed a general method to check if a given profile of FSA constitutes an equilibrium. Also, Hansen et al. deal with partially observable stochastic games (POSGs) and develop an algorithm that iteratively eliminates dominated strategies [3]. POSGs can be considered a generalization of repeated games with private monitoring, since agents might play different games at each stage. However, this algorithm can be applied only for a finite horizon, and it cannot guarantee to identify an equilibrium.

Unfortunately, the results of [6] have not yet been widely acknowledged in the AI and agent research communities. Furthermore, for the time being, there exists no work that actually applies this innovative method to identify equilibria of repeated games even in the economics/game-theory field.

The main difficulty for utilizing the result is that, although Kandori and Obara presented a general theoretical idea, based on POMDP, to identify equilibria of repeated games with private monitoring, they do not show how to implement their idea computationally [6]. Moreover, it has not yet been confirmed that this approach is really feasible when analyzing problem instances that are complex enough to represent realistic and meaningful application domains. In particular, we found that there exist one non-trivial difference between the POMDP model and the model for repeated games with private monitoring. More precisely, in a standard POMDP model, we usually assume that an observation depends on the current action and the next state. On the other hand, in the model of repeated games, we assume that an observation depends on the current action and the current state. As a result, applying/extend the results of Kandori and Obara [6] is difficult for researchers in game theory, as well as those in the AI and agent research communities.

To overcome this difficulty, we first develop a program that acts as a wrapper of a standard POMDP solver. This program takes a description of a repeated game with private monitoring and an FSA as inputs. Then, this program automatically creates an input for a POMDP solver, by taking into account the differences in the models described above. Next, this program runs a POMDP solver, analyzes the obtained results, and answers whether the FSA constitutes a symmetric equilibrium.

Furthermore, as a case study to confirm the usability of this program, we identify equilibria in an infinitely repeated prisoner’s dilemma game, where each player privately receives a noisy signal about each other’s actions. First, we consider the situation where an opponent’s action is observed with small observation errors. This case is referred to as the nearly-perfect monitoring case. Although the monitoring structure is quite natural, systematically finding equilibria in such structure has not been possible without utilizing a POMDP solver. We exhaustively search for simple FSAs with a small number of states and find a novel class of FSA called k-period mutual punishment (k-MP). Under this FSA, a player first cooperates. If she observes a defection, she also defects, but after the observation of k consecutive defections, she returns to cooperation. We can control the forgiveness of k-MP by changing the parameter k. Note that k-MP incorporates grim-trigger and the well-known strategy Pavlov [7] as a special case (k = ∞ or k = 1). Although it is somewhat counter-intuitive, requiring such mutual defection periods is beneficial in establishing a robust coordination among players in the nearly-perfect monitoring case. In contrast, in the almost-public monitoring case, the tit-for-tat (TFT) can better coordinate players’ behavior; TFT can be an equilibrium, while k-MP is not. In both cases, the grim-trigger can be an equilibrium. Accordingly, our program helps us to gain important insights into the way players coordinate their behavior under different private monitoring structures.

2. REPEATED GAMES WITH PRIVATE MONITORING

2.1 Model

We model a repeated game with private monitoring according to [6]. We concentrate on two-player, symmetric games (where a game is invariant under the permutation of players’ identifiers). However, the techniques introduced in this paper can be easily extended to n-player, non-symmetric cases.

Player \(i \in \{1, 2\}\) repeatedly plays the same stage game over an infinite horizon \(t = 1, 2, \ldots\). In each period, player \(i\) takes some action \(a_i\) from a finite set \(A\), and her expected payoff in that period is given by a stage game payoff function \(g_i(a)\), where \(a = (a_1, a_2) \in A^2\) is the action profile in that period. Within each period, player \(i\) observes her private signal \(\omega_i \in \Omega\). Let \(\omega\) denote an observation profile \((\omega_1, \omega_2) \in \Omega^2\) and let \(o(\omega \mid a)\) be the probability of private signal profile \(\omega\) given an action profile \(a\). We assume that \(\Omega\) is a finite set, and we denote the marginal distribution of \(\omega_i\) by \(o(\omega_i \mid a)\).

It is also assumed that no player can infer which action was taken (or not taken) by another player for sure; to this end, we assume that each signal profile \(\omega \in \Omega^2\) occurs with a positive probability for any \(a \in A^2\).

Player \(i\)’s realized payoff is determined by her own action and signal and denoted \(\pi_i(a_i, \omega_i)\). Hence, her expected payoff is given by \(g_i(a) = \sum_{\omega \in \Omega^2} \pi_i(a_i, \omega_i) o(\omega \mid a)\). This formulation ensures that the realized payoff \(\pi_i\) conveys no more information than \(a_i\) and \(\omega_i\) do. Note that the expected payoff is determined by the action profile, while the realized payoff is determined solely by her own action and signal.

Let us motivate this model by an example. Assume players are managers of two competing stores. The action of each player is to determine the price of an item in her store. The signal of a player represents the number of customers who visit her store. The signal is affected by the action of another player, i.e., the price of the competing store, but the realized payoff is determined solely by her own action and signal, i.e., the price and the number of customers.

The stage game is to be played repeatedly over an infinite time horizon. Player \(i\)’s discounted payoff \(G_i\) from a sequence of action profiles \(a^1, a^2, \ldots\) is \(\sum_{t=1}^{\infty} \delta^t g_i(a^t)\), with discount factor \(\delta \in (0, 1)\). Also, the discounted average payoff (payoff per period) is defined as \((1 - \delta)G_i\).

2.2 Repeated game strategies and finite state automata

We now explore several ways to represent repeated game
strategies. We start with the conventional representation of strategies in the repeated game defined above. A private history for player \( i \) at the end of time \( t \) is the record of player \( i \)'s past actions and signals, \( h^i_t = (a^i_0, \omega^i_0, \ldots, a^i_t, \omega^i_t) \in H^i \). To determine the initial action of each player, we introduce a dummy initial history (or null history) \( h^i_0 \), and let \( H^i_0 \) be a singleton set \( \{h^i_0\} \). A pure strategy \( s_i \) for player \( i \) is a function specifying an action after any history, or, formally, \( s_i : H^i \rightarrow A \), where \( H^i = \bigcup_{t \geq 0} H^i_t \).

A finite state automaton (FSA) is a popular approach for compactly representing the behavior of a player. An FSA \( M \) is defined by \( (\Theta, \hat{\theta}, f, T) \), where \( \Theta \) is a set of states, \( \theta \in \Theta \) is an initial state, \( f : \Theta \rightarrow A \) determines the action choice for each state, and \( T : \Theta \times \Omega \rightarrow \Theta \) specifies a deterministic state transition. Specifically, \( T(\theta^t, \omega^i) \) returns the next state \( \theta^{t+1} \) when the current state is \( \theta^t \) and the private signal is \( \omega^i \). We call an FSA without the specification of the initial state, i.e., \( m = (\Theta, f, T) \), a finite state preautomaton (pre-FSA). Now, we introduce a symmetric pure finite state equilibrium (SPFSE).

Definition 1. A symmetric pure finite state equilibrium (SPFSE) is a pure strategy sequential equilibrium of a repeated game with private monitoring, where each player’s behavior on the equilibrium path is given by an FSA \( M = (\Theta, f, T) \).

A sequential equilibrium is a refinement of Nash equilibrium for dynamic games of imperfect information. In this definition, we require that an FSA specifies only the behavior of a player on the equilibrium path (please consult [6] for details). We briefly describe this point later.

It must be emphasized that if an FSA \( M \) constitutes an equilibrium, it means that as long as player 2 acts according to \( M \), player 1’s best response is to act according to \( M \). Here, we do not restrict the possible strategy space of player 1 at all. More specifically, \( M \) is the best response not only within strategies that can be represented as FSAs but also within all possible strategies, including strategies that require an infinite number of states.

2.3 Monitoring structures in repeated prisoner’s dilemma

We apply the POMDP technique to the prisoner’s dilemma model analyzed by [6]. The stage game payoff is given as follows.

<table>
<thead>
<tr>
<th>( a^1 )</th>
<th>( a^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C )</td>
<td>1, 1</td>
</tr>
<tr>
<td>( D )</td>
<td>1 + ( x ), ( -y )</td>
</tr>
</tbody>
</table>

Each player’s private signal is \( \omega^i \in \{g, b\} \) (good or bad), which is a noisy observation of the opponent’s action. For example, when the opponent chooses \( C \), player \( i \) is more likely to receive the correct signal \( \omega^i = g \), but sometimes an observation error provides a wrong signal \( \omega^i = b \). Let us introduce the joint distribution of private signals \( s(\omega \mid a) \) for the prisoner’s dilemma model. When the action profile is \((C, C)\), the joint distribution is given as follows (when the action profile is \((D, D)\), \( p \) and \( s \) are exchanged).

<table>
<thead>
<tr>
<th>( w^1 )</th>
<th>( w^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g )</td>
<td>( p )</td>
</tr>
<tr>
<td>( b )</td>
<td>( s )</td>
</tr>
</tbody>
</table>

Notice that the probability that players 1 and 2 observe \((g, g)\) is \( p \), and the probability that they observe \((g, b)\) is \( q \).

Similarly, when the action profile is \((C, D)\), the joint distribution of private signals is given as follows (when the action profile is \((D, C)\), \( v \) and \( u \) are exchanged).

<table>
<thead>
<tr>
<th>( w^1 )</th>
<th>( w^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g )</td>
<td>( t )</td>
</tr>
<tr>
<td>( b )</td>
<td>( v )</td>
</tr>
</tbody>
</table>

These joint distributions of private signals require only the constraints of \( p + q + r + s = 1 \) and \( t + u + v + w = 1 \).

Repeated games with private monitoring is a generalization of infinitely repeated games with conventional imperfect monitoring. By changing signal parameters, the joint distributions can represent any monitoring structure in repeated games. Let us briefly explain several existing monitoring structures. First, we say monitoring is perfect if each player perfectly observes the opponent’s action in each period, i.e., \( p = v = 1 \) and \( r = s = t = u = w = 0 \) hold. Second, we say monitoring is public if each player always observes a common signal, i.e., \( p + s = t + w = 1 \) and \( q = r = u = v = 0 \) hold. Third, we say monitoring is almost-public if players are always likely to get the same signal (after \((C, D)\)), for example, players are likely to get \((g, g)\) or \((b, b)\), i.e., \( p + s = t + w \approx 1 \) and \( q = r \approx u = v \approx 0 \).

2.4 Existing FSAs

Let us summarize the existing FSAs in the literature of repeated games. First, grim-trigger (GT) is a well-known FSA under which a player first cooperates, but as soon as she observes defection, she defects forever. As shown in Fig. 1, this FSA has two states, i.e., \( R \) (reward) and \( P \) (punishment). Player \( i \) takes \( a^i = C \) in state \( R \) and \( a^i = D \) in state \( P \). GT can often constitute an equilibrium under perfect and imperfect monitoring.

Second, tit-for-tat (TFT) is another well-known FSA in Fig. 2. It is well known that TFT does not prescribe mutual best replies after a deviation (hence it is not a subgame perfect Nash equilibrium) under perfect monitoring. This problem does not arise under public and almost-public monitoring, and TFT can be a sequential equilibrium under public monitoring.

Finally, 1-period mutual punishment (1-MP) in Fig. 3 is known as Pavlov [7] or “win-stay, lose-shift [10].” According to this FSA, a player first cooperates. If her opponent defects, she also defects, but after one period of mutual defection, she returns to cooperation.

Pavlov is frequently used in the literature of evolutionary simulation, e.g., [7; 10]. They examine several extensions of Pavlov in the repeated prisoner’s dilemma, where a player’s action is subject to noise (trembling hands). It is well-known that Pavlov can constitute a subgame perfect Nash equilibrium under perfect monitoring. However, this has not been investigated well in the setting of private monitoring. To the best of our knowledge, 1-MP/Pavlov has not yet been identified as an equilibrium in repeated games with private monitoring. We will again discuss TFT and 1-MP under our
monitoring structures in Section 4.

3. PROGRAM FOR EQUILIBRIUM ANALYSIS

In this section, we describe our newly developed program that checks whether an FSA $M = (\Theta, \theta, f, T)$ constitutes an SPFSE according to Fig. 4.¹

3.1 Main Procedure

Let us describe the main procedures of our program indicated as “Equilibrium Analyzer” and “Standard POMDP solver” in Fig. 4. First, by assuming each player acts according to an FSA $M$, we can create a joint FSA. The expected discounted payoff of this joint FSA for player 1 is given as $V_{\hat{\theta}}$, where $V_{\hat{\theta}}$ can be obtained by solving a system of linear equations defined as follows.

$$V_{\hat{a}_1, \hat{a}_2} = g_1((f(\hat{\theta}_1), f(\hat{\theta}_2))) + \delta \sum_{(\omega_1, \omega_2) \in \Omega^2} o_1(\omega_1, \omega_2) | (f(\hat{\theta}_1), f(\hat{\theta}_2)) | - V_{T(\hat{\theta}_1, \omega_1), T(\hat{\theta}_2, \omega_2)}.$$

Now, let us consider how to obtain the best response for player 1, assuming player 2 acts according to $M$. Player 1 confronts a Markov decision process, where the state of the world is represented by the state of player 2’s FSA. However, player 1 cannot directly observe the state of player 2. Thus, this problem is equivalent to finding an optimal policy in POMDP.

More precisely, the POMDP of this problem is defined by $(\Theta, A, \Omega, O, P, R)$, where $\Theta$ is a set of states of player 2, $A$ is a set of actions of player 1, $\Omega$ is a set of observations of player 1, $O$ represents an observation probability function, $P$ represents a state transition function, and $R$ is a payoff function. $\Theta$, $A$, and $\Omega$ are already defined. $O(\omega_1 | a_1, \theta^t)$ represents the conditional probability of observing $\omega_1$ after performing an action $a_1$ at a state $\theta^t$ (of player 2), which is defined as: $O(\omega_1 | a_1, \theta^t) = o_1(\omega_1 | (a_1, f(\theta^t)))$.

Note that in a standard POMDP model, we usually assume that the observation probability depends on the next state $\theta^{t+1}$ rather than on the current state $\theta^t$. We present this alternative model here, since it is more suitable for representing repeated games with private monitoring. In the next subsection, we show how to map this model into the standard formulation of POMDP.

$$P(\theta^{t+1} | \theta^t, a_1)$$ represents the conditional probability that the next state is $\theta^{t+1}$ when the current state is $\theta^t$ and the action of player 1 is $a_1$, which is defined as:

$$P(\theta^{t+1} | \theta^t, a_1) = \sum_{\omega_2 \in \Omega | T(\theta^t, \omega_2) = \theta^{t+1}} o_2(\omega_2 | (a_1, f(\theta^t))).$$

An expected payoff function $R : A \times S \rightarrow \mathbb{R}$ is given as: $R(a_1, \theta^t) = g_1((a_1, f(\theta^t)))$.

We can check whether an FSA $M = (\Theta, \theta, f, T)$ constitutes an SPFSE by using the following procedure. This procedure is based on the general ideas presented in [6], but our description is concrete and clearly specifies a way of utilizing an existing POMDP solver.

1. First solve a system of linear equations of a joint FSA and obtain the expected discounted payoff of player 1, i.e., $V_{\hat{\theta}}$, when both players follow $M$.

²Our software will be publicly available after the review period.

2. Obtain an optimal policy $\Pi^*$ (which is given as a pre-FSA) and its value function $v(\cdot)$ for the POMDP $(\Theta, A, \Omega, O, P, R)$. Since our POMDP model is different from the standard POMDP model, we cannot directly use a standard POMDP solver such as [4]. We describe how to absorb this difference in the next subsection. In general, this computation might not converge and no optimal policy can be represented as a pre-FSA. In such a case, we terminate the computation and obtain a semi-optimal policy.²

3. Let us denote the belief of player 1 such that player 2 is in $\theta^t$ for sure, as $b_{\theta^t}$. If $v(b_{\theta^t}) = V_{\hat{\theta}, \hat{\theta}}$, then the FSA $M = (\Theta, \hat{\theta}, f, T)$ constitutes an SPFSE.

To be more precise, due to the cancellation of the significant digit, checking whether $v(b_{\theta^t}) = V_{\hat{\theta}, \hat{\theta}}$ holds can be difficult. To avoid this problem, we need to check the obtained optimal policy $\Pi^*$ as well. Note that even if $\Pi^*$ is not exactly the same as a pre-FSA $m$ of $M$, the FSA can constitute an SPFSE. This is because there can be a belief state that is unreachable when players act according to $M$. $m$ does not need to specify the optimal behavior in such a belief state, while $\Pi^*$ does specify the optimal behavior for all possible belief states.

To verify whether $M$ constitutes an SPFSE, we first find the initial state $\theta^t$ in $\Pi^*$ that is optimal when the other player employs $M$. Next, we examine a part of $\Pi^*$, i.e., the states that are reachable from $\theta^t$, and check whether this part is coincident with $M$. Then, $M$ is a best response to itself and thus it constitutes an SPFSE. In general, there can be multiple optimal policies and a POMDP solver usually returns just one optimal policy. To overcome this problem, we use $m$ as an initial policy and make sure that $\Pi^*$ includes $m$ as long as $M$ constitutes an SPFSE.

²When the obtained policy is semi-optimal but $v(b_{\theta^t}) = V_{\hat{\theta}, \hat{\theta}}$ holds, we run a procedure described in [6] to check $v(b_{\theta^t})$ remains the same in an optimal, non-FSA policy.
3.2 Procedure for Handling Model Differences
In this subsection, corresponding with “Model Translator” in Fig. 4, we describe a method for translating a POMDP description \((\Theta, A, \Omega, O, P, R)\) in our model, into a standard model \((\Theta', A, \Omega', O', P', R')\). Here, the possible set of actions \(A\) and observations \(\Omega\) are the same in these two models.

The key idea of this translation is to introduce a set of new combined states \(\Theta'\), where \(\Theta' = \Theta^2\). Namely, we assume that a state \(\theta^t\) in the standard POMDP model represents the combination of the previous and current states \((\theta^{t-1}, \theta^t)\) in our model present in the previous subsection. For example, assume player 1 acts according to an FSA called grim-trigger (GT) defined in Fig. 1. There are two states in the original model. Consequently, in the standard model, there are \(2 \times 2 = 4\) states, i.e., \(\Theta' = \{(R, R), (R, P), (P, R), (P, P)\}\). Among these four states, \((P, R)\) is infeasible, and thus there exists no state transition to \((P, R)\).

A new state transition function \(P'(\theta^{t+1} | \theta^t, a_1)\) is equal to \(P(\theta^{t+1} | \theta^t, a_1)\) in the original model if \(\theta^{t+1} = (\theta^t, \theta^{t+1})\) and \(\theta^t = (\theta^{t-1}, \theta^t)\), i.e., the previous state in \(\theta^{t+1}\) and the current state in \(\theta^t\) are identical. Otherwise, it is 0. Next, let us examine how to define \(O'(\omega^t | a_1, \theta^t, \theta^{t+1})\). This is identical to the posterior probability that the observation was \(\omega^t\), when the state transits from \(\theta^t\) to \(\theta^{t+1}\). Thus, this is defined as:

\[
O'(\omega^t | a_1, \theta^t, \theta^{t+1}) = \frac{\sum_{\omega^t \in \Omega} O(\omega^t, \omega_{t+1} | (a_1, f(\theta^t)))}{\sum_{\omega^t \in \Omega} O(\omega^t, \omega_{t+1} | (a_1, f(\theta^t)))}
\]

where \(\Omega' = \{(\omega^t | T(\theta^t, \omega^t) = \theta^{t+1}\}\). For example, let us consider that player 1 takes \(a_1 = C\) when player 2, who acts according to GT, is in state \((R, R)\). The probability that player 1 observes \(w_1 = g\) is given by

\[
O'(g | C, (R, R)) = \frac{O(g, g | (C, C)) + O(b, g | (C, C))}{O(g, g | (C, C)) + O(b, g | (C, C))}
\]

Finally, the expected payoff function, \(R'(a_1, \theta^{t-1}, \theta^t)\), is given as \(R(a_1, \theta^t)\).

This translation does not affect the optimal policy. More specifically, by solving the translated POMDP \((\Theta', A, \Omega', O', P', R')\), we obtain an optimal policy \(\Pi'\) (which is described as a pre-FSA) and its value function \(v'(b_y)\). Then, an optimal policy \(\Pi^*\) of the original POMDP \((\Theta, A, \Omega, O, P, R)\) is identical to \(\Pi'^*\). Also, from \(b_y\), which is a belief over \(\theta' = (\theta^{t-1}, \theta^t)\), we can extract \(b_{y'}\), i.e., a belief over the current state. Then, \(v'(b_{y'}) = v(b_{y'})\) holds.

3.3 Program Interface
This program takes the discount factor, the description of a stage game, a monitoring structure defined by \(o(\omega | a)\), i.e., the probabilistic of private signal profile \(a\) given an action profile \(a\), and an FSA, as “Inputs” of Fig. 4. Let us show an example. The meanings of these descriptions are self-explanatory.

discount: 0.9
actions: C D
# payoff matrix
PM:C:C: 1: 1
PM:D:C: 2: 1
PM:C:D: -1: 2
PM:D:D: 0: 0

observations: g b
# observation probability
O:g:C:C: 0.97
O:b:C:C: 0.01
O:g:b:C:C: 0.01
O:b:b:C:C: 0.01

... # FSA description of Grim-trigger
states: R P
start: R
T:R: g: R
T:R: b: P
T: R: g: P
T: P: b: P

4. REPEATED PRISONER’S DILEMMA WITH NOISY OBSERVATION
This section first defines a monitoring structure that is nearly-perfect. We say monitoring is nearly-perfect if each player is always likely to perfectly observe the opponent’s action in each period, i.e., \(p = v, q = r = 1 = w, s = u = 1 - p - 2q, \) where \(p\) is much larger than \(q\) or \(s\). Throughout our paper, we basically use the default setting: \(x = 1, y = 1,\) and the discount factor \(\delta = 0.9\). We assume \(p \in (1/2, 1)\) and \(q \in (0, 1/4)\) under the constraint \(p + 2q + s = 1\) and that \(n_i(a_i, \omega_i)\) is chosen so that \(g_i(a_i)\) is constant. Next, this section identifies signal parameters where GT, TFT, and 1-MP constitute an SPFSE according to our program.

4.1 Grim-trigger
This subsection examines a representative FSA, called grim-trigger (GT). When both players act according to GT, a joint FSA has four states: RR, RP, PR, and PP. The system of linear equations for this joint FSA is given as

\[
\begin{pmatrix}
V_{RR} \\
V_{RP} \\
V_{PR} \\
V_{PP}
\end{pmatrix} = \begin{pmatrix}
1 & -\delta & p & q & q & s \\
-\delta & 1 & 0 & q+s & 0 & p+q \\
0 & -\delta & 1 & 0 & 0 & q+s & p+q \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix} \begin{pmatrix}
V_{RR} \\
V_{RP} \\
V_{PR} \\
V_{PP}
\end{pmatrix}
\]

By solving this, we obtain

\[
V_{RR} = \frac{1 - \delta s}{(1 - \delta) p - (1 - \delta s - \delta q)}
\]

Figure 12 illustrates the range of signal parameters over which GT constitutes a SPFSE. The \(x\)-axis indicates \(p\), the probability that signals are correct, e.g., \(o((g, g)|(c, c))\) or \(o((b, g)|(c, d))\). The \(y\)-axis indicates \(q\), the probability that signals have exactly one error, e.g., \(o((g, b)|(c, c))\) or \(o((b, g)|(d, d))\). When \(p\) is large, the signals of the two players tend to be correct, i.e., the player is likely to observe \(g/b\) when her opponent cooperates/defects. When \(q\) is small, the signals are strongly correlated, i.e., if the signal of a player is wrong, the signal of her opponent is also likely to be wrong.

Basically, GT constitutes an SPFSE where \(p\) is large and \(q\) is small, i.e., the signals are accurate and strongly correlated. Suppose \(p\) is large but \(q\) is not small, and assume that player 1 observes \(b\). Player 1 is quite sure that this is an error.

Furthermore, since the correlation is not so strong, player 2 is likely to receive a correct signal. Thus, for player 1, it is better to deviate from GT and to keep cooperation. When \(p\) is relatively small, in contrast, the probability that the opponent observes \(b\) is large. Therefore, it is better to start with defection. A shortcoming of GT is that it is too unforgiving and thus generates a low payoff. For example, when
Figure 5: Joint FSA for TFT under nearly-perfect monitoring, Figure 7: Joint FSA for 3-MP under nearly-perfect monitoring, Figure 8: Joint FSA for 1-MP under nearly-perfect monitoring.

$p = 0.9$, $q = 0.01$, and $\delta = 0.9$, the expected discounted payoff is about 3.31, while if players can keep cooperating, the expected discounted payoff would be 10.

4.2 TFT and 1-MP

TFT in Fig. 2 is well-known as a more forgiving strategy than GT. However, if two players use TFT, an observation of defection leads to poorly coordinated behavior. Figure 5 shows the joint FSA for TFT under nearly-perfect monitoring. Thick/dotted/thin lines represent the transition with probabilities $p$, $q$, and $s$, respectively. Notice that we assume $p$ is much larger than $q$ or $s$. We can see that after an observation error players largely alternate between $(C, D)$ and $(D, C)$. In such a situation, a player is better off deviating to $p$ to end this cycle and returning to $(C, C)$. For this reason, TFT does not constitute an SPFSE under nearly-perfect monitoring. Note that, basically for the same reason, TFT does not constitute a subgame perfect Nash equilibrium under perfect monitoring. Furthermore, the payoff associated with TFT is low. After an observation error, it is difficult to go back to $(C, C)$, as Fig. 5 shows. In fact, the probability of $(C, C)$ in the invariant distribution is 0.25, as long as $q > 0$ and $s > 0$.

Let us turn our attention to almost-public monitoring for a moment. We examined whether TFT is an SPFSE or not under almost-public monitoring within a wide range of signal parameters by utilizing our developed software. We confirmed TFT is an SPFSE only if $q$ is smaller than about 0.0001 in our parameterization. If two players use TFT under almost-public monitoring, an observation of defection leads to coordinated behavior. Figure 6 shows the joint FSA. Thick/thin/dotted lines represent the transition with probabilities $p$ ($w$), $s$ ($t$), and $q$, respectively. We can see that after an observation error players no longer alternate between $(C, D)$ and $(D, C)$. Although they will likely transit or stay at the mutual punishment state $PP$, they are more likely to return to the mutual cooperation state $RR$ than under nearly-perfect monitoring. Notice that the similar argument can be applied to the public monitoring case. Furthermore, our software enables us to exhaustively search for all FSAs with at most three states that can constitute an equilibrium under almost-public monitoring. We found that TFT is the most efficient in SPFSE among all FSAs, including GT.

Now, let us consider the FSA in Fig. 3, which we call 1-period mutual punishment (1-MP). As we noted, traditionally, this FSA is known as Pavlov [7]. Recall that, according to this FSA, a player first cooperates. If her opponent defects, she also defects, but after one period of mutual defection, she returns to cooperation. Figure 7 shows the joint FSA of 1-MP. We can see that after one observation error occurs, players can quickly return to the mutual cooperation state $RR$. The expected probability (in the invariant distribution) that players are in state $RR$ is about $p - 2q$.

Unfortunately, 1-MP does not constitute an SPFSE in our parameterization, since it is too forgiving. Basically, 1-MP punishes a deviator by one period of mutual defection. The gain from defection $x$ is exactly equal to the loss in the next period $y$ ($x = y = 1$). Therefore, as long as a player discounts future payoff, 1-MP cannot be an SPFSE, even under perfect monitoring. Also, 1-MP does not constitute an SPFSE under almost-public monitoring. Figure 8 illustrates that an observation of defection leads to poorly coordinated behavior, as in TFT under nearly-perfect monitoring.

5. K-PERIOD MUTUAL PUNISHMENT

This section generalizes the idea of 1-MP to $k$-period mutual punishment ($k$-MP). Under this FSA, a player first cooperates. If her opponent defects, she also defects, but after $k$ consecutive periods of mutual defection, she returns to cooperation.

Figure 9 shows the FSAs of 2-MP. 2-MP is less forgiving than 1-MP, since it cooperates approximately once in every three periods to the opponent who always defects. By increasing $k$, we can make this strategy less forgiving. When $k = \infty$, this strategy becomes equivalent to GT. Figure 11 shows a joint FSA for 2-MP. For simplicity, we show thick lines that represent the transition with probability $p$. We can see that after some observation errors occur players can quickly return to the mutual cooperation state $RR$.

Figure 12 illustrates the range of signal parameters over which 2-MP is an SPFSE. For comparison, we show the range where GT is an SPFSE. We can see that even for $1-MP$ is a subgame perfect Nash equilibrium under perfect monitoring only if $\frac{1-q}{1-2q} < \frac{1-y}{1-\delta}$.
k = 2, k-MP can be an SPFSE in a reasonably wide range of signal parameters, though the size of the range is smaller than GT. When the correlation of signals is quite strong (q ≈ 0), 2-MP constitutes an SPFSE in the range of signal correctness p ∈ [0.82, 1]. As the correlation becomes slightly larger (i.e., q > 0.04), 2-MP is no longer an SPFSE. When q = 0.04, 2-MP constitutes an SPFSE in the range of correctness p ∈ [0.86, 0.91]. It is significant that GT is more sensitive to the correlation than 2-MP when p is sufficiently large. When the correctness p exceeds 0.86, there is a range of correlation where GT is not an SPFSE but 2-MP is. Figure 12 also shows the range of signal parameters over which 3-MP (Fig. 10) is an SPFSE. The SPFSE range of 3-MP includes almost all that of 2-MP.

Now, let us examine the average payoff of GT and k-MP. In Fig. 13, the x-axis indicates the correctness of signal p, while the correlation q is fixed at 0.01. The y-axis indicates the average payoff per period. Note that average payoff is 1 if mutual cooperation is always achieved. It is clear that 2-MP significantly outperforms GT and 3-MP regardless of signal correctness. We also placed two points on each line. Within the range between the two points, an FSA constitutes an SPFSE. We can see that the size of the range becomes wider by increasing k, but the efficiency becomes lower.

One obvious question is whether there is any FSA (except k-MP) that constitutes an SPFSE and achieves a better efficiency. To answer this question, we exhaustively search for small-sized FSAs that can constitute an equilibrium. We enumerate all possible FSAs with at most three states, i.e., \(|A|^{s_1} \cdot |\Theta|^{s_2} |\Theta|^{s_3}| = 5832\) FSAs, and check whether they constitute an SPFSE. We found that only eleven FSAs (after removing equivalent ones) could be an SPFSE in a reasonably wide range of signal parameters. Furthermore, among them, 2-MP is the only FSA that is more efficient than GT.

6. EXTENSION WITH A RANDOM PRIVATE SIGNAL

Let us assume that agents can observe additional signals which (i) do not affect payoffs, (ii) convey no information about players’ actions, and (iii) are strongly correlated. Interestingly, players can achieve better coordination by utilizing such “irrelevant” almost public signals. More specifically, let us assume that a player observes whether a particular event happens or not before each stage game. We assume with probability \(p’\), that both players observe the event, with probability \(s’\) that neither players observes the event, and with probability \((1 – p’ – s’)/2\) that player 1 or 2 observes the event but player 2 or 1 does not, respectively. We assume \(p’\) is relatively small (not too frequent), and \((1 – p’ – s’)/2\) is much smaller than \(p’\), i.e., if one player observes the event, it is very likely that the other player also observes the event. Then, how can players utilize (or disregard) this signal?

Let us assume a parameter setting where GT constitutes an SPFSE. Since this signal is totally independent from the utilities/observation of players, disregarding this signal never hurts. Thus, GT (which disregards the signal) still constitutes an equilibrium.

Now let us assume player 2 uses the following strategy: as long as the event is not observed, play GT, but when the event is observed, move to state \(R\). Then, assuming player 2 uses this strategy, for player 1, using the same strategy as player 2 would be a best response. This is because if player 1 observes the event, it is very likely that player 2 also ob-
Figure 14: Ranges of signal parameters over which GT/2-MP and GT-s/2-MP-s are SPFSE.

Figure 15: Average payoff per period of GT/2-MP and GT-s/2-MP-s (q=0.01).

7. CONCLUSION

This paper investigates repeated games with imperfect private monitoring. Although analyzing such games has been considered as a hard problem, we develop a program that automatically checks whether a given profile of FSAs can constitute an SPFSE. Our program is based on the ideas presented in [1] and utilizes an existing POMDP solver. This program enables non-experts of the POMDP literature, including researchers in the game theory, AI, and agent research communities, to analyze the equilibria of repeated games.

Furthermore, as a case study to confirm the usability of this program, we identify equilibria in an infinitely repeated prisoner’s dilemma game with imperfect private monitoring, where the probability of an error is relatively small. We first examine how observation errors affect the behavior of GT, TFT, and 1-MP (Pavlov). Then we propose the k-MP strategy, which incorporates GT and Pavlov as a special case, and show that k-MP constitutes an SPFSE in a reasonably wide range of observation errors. Its efficiency is better than that of GT. We exhaustively search for simple FSAs with at most three states and confirm that no other FSA constitutes an equilibrium in a reasonably wide range of signal parameters nor is more efficient than GT. In our future work, we hope to investigate other games, such as congestion games, which can model various application problems including a packet routing problem, by utilizing our program.

8. REFERENCES

Exploiting Sparse Interactions for Optimizing Communication in Dec-MDPs

Francisco S. Melo
INESC-ID
Instituto Superior Técnico
Porto Salvo, Portugal
fmelo@inesc-id.pt

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

Stefan Witwicki
INESC-ID
Instituto Superior Técnico
Porto Salvo, Portugal
witwicki@inesc-id.pt

ABSTRACT

Decentralized partially observable Markov decision processes (Dec-POMDPs) provide powerful modeling tools for multi-agent decision-making in the face of uncertainty, but solving these models comes at a very high computational cost. Two avenues for side-stepping the computational burden can be identified: structured interactions between agents and intra-agent communication. In this paper, we focus on the interplay between these concepts, namely how sparse interactions reflect in the communication needs. A key insight is that in domains with local interactions the amount of communication necessary for successful joint behavior can be heavily reduced, due to the limited influence between agents. We exploit this insight by deriving local POMDP models that optimize each agent’s communication behavior. Our experimental results show that our approach successfully exploits sparse interactions: we can effectively identify the situations in which it is beneficial to communicate, as well as trade off the cost of communication with overall task performance.

1. INTRODUCTION

Decentralized partially observable Markov decision processes (Dec-POMDPs) provide powerful modeling tools for multiagent decision-making with limited sensing capabilities in stochastic environments. However, the prohibitive computational cost required to compute an optimal decision rule renders them intractable except for the smallest of problems.

In the literature, two avenues for side-stepping the computational burden can be identified: localized interactions between agents [2, 16, 21, 23, 24] and intra-agent communication [7, 18, 20, 26]. In this paper, we focus on the interplay between these concepts, namely how sparse interactions reflect in the communication needs.

A key insight is that in domains with local interactions the amount of communication necessary for successful joint behavior can be heavily reduced, due to the limited influence between agents. Several previous works have implicitly relied on this observation, exploring sparse interactions by having agents share information locally [11, 13, 18, 21, 23]. In this work, we explicitly reason about the benefits of communication/information sharing in scenarios with sparse interactions. Sparse interactions enable, to some extent, decoupling the decision-process of the different agents. We leverage such decoupling to derive local models that optimize each agent’s communication behavior, allowing it to overcome partial observability in those situations where decoupled decisions are not possible.

We provide a new way of optimizing communication by proposing a model in which agents need to plan about when to query other agents’ local states, which we call Query-POMDP. We observe that to execute optimal joint policies in fully observable scenarios—policies which can be computed efficiently—agents will generally need to reason about the state of other agents. Our agents construct a local POMDP model of the environment from the fully observable joint policy of all other agents. Solving this POMDP model allows the agent not only to determine how to solve the task at hand but also to determine when to query the local state of the environment. Our approach thus allows the agents to explicitly reason about communication, without incurring in the prohibitive computational cost of Dec-POMDP models that include communication [17]. Furthermore, in contrast to many methods in the literature [18, 26], QueryPOMDP can properly handle noisy communication channels, and does not require strong independence assumptions [1]. Our empirical analysis on benchmark problems demonstrates the efficacy of QueryPOMDP in balancing communication costs with coordination benefits.

2. BACKGROUND

We start by reviewing decentralized partially observable Markov decision processes (Dec-POMDPs) and related decision theoretic models. An N-agent Dec-POMDP \( \mathcal{M} \) is specified as a tuple \( \mathcal{M} = (N, \mathcal{X}, (\mathcal{A}_i), (\mathcal{Z}_i), \mathcal{P}, (\mathcal{O}_i), \mathcal{r}, \gamma) \), where:

- \( \mathcal{X} \) is the joint state-space;
- \( \mathcal{A} = \times_{i=1}^N \mathcal{A}_i \) is the set of joint actions, with each \( \mathcal{A}_i \) the individual action set for agent \( i, i = 1, \ldots, N \);
- Each \( \mathcal{Z}_i \) represents the set of possible local observation for agent \( i, i = 1, \ldots, N \);
- \( \mathcal{P}(y | x, a) \) represents the transition probabilities from joint state \( x \) to joint state \( y \) when the joint action \( a \) is taken;
- Each \( \mathcal{O}_i(z_i | x, a) \) represents the probability of agent \( i \) making the local observation \( z_i \) when the joint state is \( x \) and the last joint action taken was \( a \);

The Seventh Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2012), held in conjunction with AAMAS, June 2012, in Valencia, Spain.
• $r(x, a)$ represents the expected reward received by all agents for taking the joint action $a$ in joint state $x$;
• The scalar $\gamma$ is a discount factor.

An $N$-agent Decentralized Markov decision process (Dec-MDP) is a particular class of Dec-POMDP in which the state is jointly fully observable, however, optimal solving of the model is in the same complexity class as the Dec-POMDP model. Formally this can be translated into the following condition: for every joint observation $z \in \mathcal{Z}$, with $\mathcal{Z} = \times_{i=1}^N \mathcal{Z}_i$, there is a state $x \in \mathcal{X}$ such that

$$P[X(t) = x | Z(t) = z] = 1,$$

where $X(t)$ is the joint state of the process at time $t$ and $Z(t)$ the corresponding joint observation. A partially observable Markov decision process (POMDP) is a 1-agent Dec-POMDP and a Markov decision process (MDP) is a 1-agent Dec-MDP. Finally, an $N$-agent multiagent MDP (MMDP) is an $N$-agent Dec-MDP that is fully observable, i.e., for every individual observation $z_i \in \mathcal{Z}_i$ there is a state $x \in \mathcal{X}$ such that

$$P[X(t) = x | Z_i(t) = z_i] = 1.$$

In this partially observable multiagent setting, an individual (non-Markov) policy for agent $i$ is a mapping

$$\pi_i : \mathcal{H}_i \rightarrow \Delta(\mathcal{A}_i),$$

where $\Delta(\mathcal{A}_i)$ is the space of probability distributions over $\mathcal{A}_i$ and $\mathcal{H}_i$ is the set of all possible finite histories for agent $i$. The purpose of all agents is to determine a joint policy $\pi$ that maximizes the total sum of discounted rewards. In other words, considering a distinguished initial state $x^0 \in \mathcal{X}$ that is assumed common knowledge among all agents, the goal of the agents is to maximize

$$V^\pi = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r(X(t), A(t)) \mid X(0) = x^0 \right].$$

For a more detailed introduction to Dec-POMDPs and related models see, e.g., [19].

3. A MOTIVATING EXAMPLE

We motivate our ideas in a simple navigation scenario, corresponding to the environment depicted in Fig. 1. In this scenario, two robots (Robot 1 and Robot 2) must navigate to their corresponding goal states (marked as Goals 1 and 2). At the same time, they must avoid colliding in the narrow doorway (the central state), since it leads to a large penalty. Each robot has 4 possible actions (namely “Move North”, “Move South”, “Move East” and “Move West”) that move the robot in the corresponding direction. The motion of one robot does not depend on the position or action of the other robot except in the doorway: if the robots collide in the doorway, then their actions have an increasing failure probability. Complicating matters, initially each robot starts uniformly at random in one of the 10 locations on its side of the doorway.

In a fully observable situation, the agents will move toward their respective goals. When reaching the doorway, if the other robot is also close to the doorway one of the two will stop so that the other can safely traverse.\(^2\) It will then resume its trajectory to its goal.

In order for the agents to actually execute the policy just described, they only need to reason about the state of the other agent when reaching the darker area in their starting side of the environment. And then, once one robot is in the doorway, it can just proceed toward its goal, independently of the state of the other robot. Moreover, even if the robots are generally unable to observe the position of the other robot, but they are able to query it, they can reasonably assume that the other robot will behave more or less as in the fully observable scenario. This observation is the departing point for the model and approach proposed in this paper and described in the continuation.

4. A MODEL FOR STATE QUERYING

We depart from an $N$-agent Dec-MDP model, and address the problem of when communication can be beneficial to improve the performance in such a model. For the purposes of our study, we momentarily focus on the decision processes of all except one agent, which we refer to as agent $k$. Unlike other communication-based approaches to Dec-POMDPs (e.g., [18, 27]), we adopt a relatively general communication model, in which the messages received by an agent are taken as part of its local (noisy) observation. Also, messages received by agent $k$ depend on explicit information-querying actions executed by $k$.

Throughout this section, we represent the (finite) state-space of the Dec-MDP as a set $\mathcal{X}$ and assume that it can be factorized as $\mathcal{X} = \mathcal{X}_k \times \mathcal{X}_{-k}$, where the elements $x_k \in \mathcal{X}_k$ correspond to agent $k$’s local state. The state at time $t$, $X(t)$, is thus a pair $(X_k(t), X_{-k}(t))$. We also assume that the observations of each agent do not depend on the actions of the remaining agents, i.e.,

$$P[Z_i(t) = z_i | X(t), A_i(t)] = P[Z_i(t) = z_i | X(t), A_i(t)],$$

for all $i = 1, \ldots, N$. Therefore, we can simply write the observation probabilities as $O_i(z_i | x, a_i), i = 1, \ldots, N$.\(^2\)

\(^2\)Which one stops is determined by the joint policy they adopt.
4.1 Query Actions and Observations

For the purpose of allowing our agent to reason about communication, we assume that each agent has the ability to query the other agents for their local state information. In order to make this explicit, we differentiate between communication actions and the remaining actions—henceforth referred as primitive actions, and write the set of individual actions for agent \( k \) as the Cartesian product of the set of communication actions, \( A^C_k \), and the set of primitive actions, \( A^P_k \), i.e., \( A_k = A^C_k \times A^P_k \). We also assume that transition probabilities are independent of the communication actions,
\[
P(y \mid x, (a_{-k}, (a^C_k, a^P_k))) = P(y \mid x, (a_{-k}, (b^C_k, a^P_k)))
\]
for any \( x, y \in X, a_{-k} \in A_{-k}, a^C_k \in A^C_k \) and \( a^P_k, b^C_k \in A^C_k \).

We also differentiate between communication observations (i.e., observations that result from communication actions) and primitive observations, that do not depend on the communication actions. Formally, we write the set of individual observations for agent \( k \) as the Cartesian product of the set of communication observations, \( Z^C_k \), and primitive observations, \( Z^P_k \), i.e., \( Z_k = Z^C_k \times Z^P_k \). We consider that communication observations do not depend on primitive actions, and that primitive observations do not depend on communication actions. This means that we can decouple the observation probabilities as
\[
O_k((z^C_k, z^P_k) \mid x, (a^C_k, a^P_k)) = O^C_k(z^C_k \mid x, a^C_k)O^P_k(z^P_k \mid x, a^P_k),
\]
where
\[
O^C_k(z^C_k \mid x, a^C_k) = P[Z^C_k(t) = z^C_k \mid X(t) = x, A^C_k(t) = a^C_k],
\]
\[
O^P_k(z^P_k \mid x, a^P_k) = P[Z^P_k(t) = z^P_k \mid X(t) = x, A^P_k(t) = a^P_k].
\]

Finally, we assume that the reward function can also be decomposed as the sum of two components. The first component, denoted \( r^C \), concerns the cost of communication and is independent on the primitive actions of agent \( k \) and on the actions of the other agents. The second component, denoted as \( r^P \), corresponds to the regular (or domain-level) reward defining the overall goal of the agents. It is assumed independent of the communication actions of agent \( k \). Formally, if \( a = (a_{-k}, a_k) \) and \( a_k = (a^C_k, a^P_k) \), this means that the reward \( r \) can be written as
\[
r(x, a) = r^P(x, (a_{-k}, a^P_k)) + r^C(x, a^C_k).
\]

Figure 2 depicts a dynamic Bayesian network that summarizes all above considerations.

Following the discussion in Section 3, and for the purpose of its planning process, agent \( k \) will treat all remaining agents as if they follow a Markov policy, \( \pi_{-k} \), that corresponds to the optimal policy for the underlying MMDP. This policy, being Markovian, depends only on the state of the system at time \( t \), \( X(t) \), i.e.,
\[
\begin{align*}
P[A_{-k}(t) = a_{-k} \mid H(t)] &= P[A_{-k}(t) = a_{-k} \mid X(t) = x] = \pi_{-k}(x, a_{-k}),
\end{align*}
\]
where \( A_{-k}(t) \) denotes the action taken by all agents other than \( k \) at time \( t \), \( H(t) \) denotes the whole history of the process up to time \( t \) and \( a_{-k} \in A_{-k} \). From this perspective, the decision process for agent \( k \) can be modeled as a (single-agent) POMDP that we describe in the next section.

4.2 POMDP Model for a Single Agent

Let \( \mathcal{M} = (\mathcal{N}, \mathcal{X}, (A_k), (Z_k), P, (O_k), r, \gamma) \) be a Dec-MDP that meets the above assumptions. Let \( \pi_{-k} \) denote the (state-dependent) joint MMDP policy for all agents other than \( k \). The single-agent POMDP model for agent \( k \) is a tuple \( \mathcal{M}_k = (\mathcal{X}, \mathcal{A}_k, Z_k, P_k, O_k, r_k, \gamma) \), where:

- \( \mathcal{X} \) corresponds to the original Dec-MDP state-space.
- \( A_k \) is the individual action-space for agent \( k \).
- \( Z_k \) is the individual observation-space for agent \( k \).
- \( P_k \) are the transition probabilities obtained from the original transition probabilities. In particular, given an action \( a_k = (a^C_k, a^P_k) \), we have
\[
P_k(y \mid x, a_k) = \sum_{a_{-k} \in A_{-k}} \pi_{-k}(x, a_{-k})P(y \mid x, (a_{-k}, a^P_k)).
\]
- \( O_k \) are the observation probabilities for agent \( k \), that match the original Dec-MDP observation probabilities. In particular, given an action \( a_k = (a^C_k, a^P_k) \), we have
\[
O_k(z_k \mid x, a_k) = O^C_k(z^C_k \mid x, a^C_k)O^P_k(z^P_k \mid x, a^P_k),
\]
where \( z_k = (z^C_k, z^P_k) \).
- \( r_k \) is the reward function obtained from the original Dec-MDP reward function after averaging over the other agents’ policy, \( \pi_{-k} \), i.e.,
\[
r_k(x, a_k) = \sum_{a_{-k} \in A_{-k}} \pi_{-k}(x, a_{-k})r(x, (a_{-k}, a_k)).
\]

Given this POMDP model, we can use standard POMDP solution techniques to explore the trade-off between the costs and benefits of communication for agent \( k \).

4.3 Results for the H-environment Example

Continuing the example of Section 3, the application of our model allows us to better understand under which circumstances the benefits of using communication compensate for its costs. For this purpose, we fix the policy of Agent 2 as shown in Fig. 1, which corresponds one possible joint MMDP policy for this environment. As explained above, given such a policy we can construct a POMDP from the point of view of Agent 1, in which it can query Agent 2’s states at any time step, at a particular communication cost. For illustration purposes, the initial state of Agent 2 is selected randomly on the right half of the environment. We
test several experimental conditions that include the presence or absence of transition noise and different costs for the communication actions.

We examine in which states Agent 1 queries Agent 2’s state. When communication is free (Figs. 3(a) and (b)), Agent 1 queries in all the states it passes through. With a communication cost of 0.3 (Fig. 3(d)), however, it only queries when near to and left of the doorway. In these states it is crucial to know Agent 2’s location to avoid potential collisions, an intuition that is exploited automatically by our model.

The use of a POMDP model in this context ensures that the agent explicitly reasons about information gathering actions which, in our setting, translates in weighting the benefits of communication in terms of the overall task against the costs associated with it. It is also worth noting that the optimal communication policy depends on the communication costs, on the task at hand, the transition model, and the query model (i.e., the observation probabilities associated with the query actions). The planning framework proposed takes into consideration all these aspects, and the right trade-off between communication costs and benefits comes out as a natural outcome of the policy computed by the agent.

4.4 Computing Policies for Multiple Agents

In the previous section we proposed using a POMDP model to compute the policy for one agent $k$, treating all other agents as if they were following the optimal joint policy for the underlying MMDP. Given this POMDP model for agent $k$ we can compute the corresponding optimal policy using any preferred POMDP solution technique. We use this approach to better understand the communication needs of one agent in a simple multiagent navigation scenario, and to determine in which situations the cost of communication outweighs its value.

We now want to extend these ideas and actually compute the policy for all agents in the Dec-MDP. The idea of using POMDP models to plan in multiagent scenarios has been previously explored in the Dec-POMDP literature [6, 14]. The general difficulty with these approaches arises from the fact that each agent has only a local observation of the joint state of the world. This implies that, when planning for agent $k$, the POMDP model necessary to properly capture the behavior of all agents other than $k$ can either be prohibitively large, require agent $k$ to reason about how the other agents reason about agent $k$’s state, leading to infinitely nested beliefs, or both [6, 14].

In our approach, we rely on the intuition discussed in Section 3, according to which the use of active communication allied with sparse interactions may actually alleviate the difficulties associated with planning in multiagent systems with partial observability. We plan for each agent $k$ while treating all other agents as if following the optimal joint MMDP policy. In scenarios where interactions are sparse, the general behavior of the agents is expected to roughly follow the MMDP policy, as discussed in Section 3 and in those situations where coordination is necessary, agents can resort to communication, but weighting the benefits of such communication with the associated costs.

Several previous works have already studied the benefits of exploiting communication and structured interactions separately (see, for example, [7, 21, 24]). The novelty in our approach lies precisely on the fact that we can explicitly exploit the interplay between these two aspects (communication and sparse interactions) to attain efficient planning in multiagent problems. Section 5 describes the application of our approach in several navigation scenarios of different dimensions. Our results empirically show that our approach is indeed able to make effective use of communication and attain a performance that indeed approaches that observed in fully observable settings.

5. EXPERIMENTS

In this section we illustrate the application of our method to several navigation scenarios from the POMDP and Dec-POMDP literature. We use robot navigation scenarios (see Fig. 4), since our model is particularly suited for modeling multi-robot problems. Furthermore, results can be easily visualized and interpreted in this class of problems. We consider only two-agent scenarios for ease of interpretation although, as argued, our approach can be used with any number of agents. Moreover, for our purposes (i.e., investigating how the proposed use of communication can alleviate the computational burden of solving Dec-MDPs), two-agent scenarios are already illustrative, as they are NEXP-complete [4].

---

3We note that, due to the transition noise, an agent can remain in the same state more than one consecutive time-step, and hence the values larger than 1.

4We refer to Section 6 for a detailed discussion of related approaches.
Experimental Setup

In each of the test scenarios, two robots must each reach one specific state, marked either by a boxed number or a cross, ×. Each robot has 4 actions that move the robot in one of the four possible directions with probability 0.8 and fail with probability 0.2, plus a fifth “NoOp” action. As for the communication model, the agent’s query actions will provide the local state of the other agent with probability 0.8 in the shaded areas. With a probability 0.2, communication will fail and the query action will provide no observation. Outside the shaded areas, the query actions always yield no communication. Since the communication model is available to the agents, planning takes into consideration communication limitations.\(^5\)

The darker cells correspond to states where the agents receive a penalty of −20 when standing there simultaneously, in which case the rate of action failure is also increased to 0.4 for both agents.

All agents have full local state observability. When an agent queries another agent, it incurs a cost of −0.1 and, in the shaded areas, successfully observes the local state of the queried agent with a probability of 0.8. With a probability of 0.2 it receives no observation about the state of the other. In the white cells, an agent is never able to perceive the state of the other, but still incurs a penalty of −0.1 if it attempts to communicate. When an agent reaches its goal position, it receives a reward of 10 and moves to a rewardless absorbing state. Throughout the experiments, we used \(\gamma = 0.95\).

For each of the test scenarios, following the approach in Section 4, we compute the optimal MMDP joint policy that we use to determine a POMDP model describing the decision process for each individual agent. This POMDP is then solved using the PERSEUS approximate solver [22]. We test our QUERYPOMDP policy for 100 independent trials of 250 steps each and measure the obtained performance in terms of total discounted reward. We also test the performance of other sets of agents that communicate at different (but fixed) frequencies (see Table 1):

- “NEVER COMM” agents never communicate. These agents observe only their local state, and each follows the optimal policy for the underlying single-agent MDP obtained by disregarding the other agent in the environment;

- “ALWAYS COMM” agents communicate at every time-step, incurring the corresponding penalty. As QUERYPOMDP agents, they are subject to communication errors/limitations and, as such, are not always able to perceive the state of the other agent. When communication fails, the agent observes only its local state and adopts the individual MDP policy. When communication succeeds, it adopts the underlying MMDP policy.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERYPOMDP</td>
<td>Variable POMDP</td>
<td>POMDP</td>
<td>POMDP</td>
</tr>
<tr>
<td>NEVER COMM</td>
<td>Never</td>
<td>–</td>
<td>Indiv. MDP</td>
</tr>
<tr>
<td>ALWAYS COMM</td>
<td>1 step MMDP</td>
<td>Indiv. MDP</td>
<td>Indiv. MDP</td>
</tr>
<tr>
<td>COMM (k = 2)</td>
<td>2 steps MMDP</td>
<td>Indiv. MDP</td>
<td>Indiv. MDP</td>
</tr>
<tr>
<td>COMM (k = 3)</td>
<td>3 steps MMDP</td>
<td>Indiv. MDP</td>
<td>Indiv. MDP</td>
</tr>
<tr>
<td>COMM (k = 4)</td>
<td>4 steps MMDP</td>
<td>Indiv. MDP</td>
<td>Indiv. MDP</td>
</tr>
</tbody>
</table>

\(^5\)We note that, although for the experimental results we consider a relatively simple communication model, it already includes state-dependent noise in the resulting observations. Generally, the framework accommodates for arbitrary communication models.
Table 2: Total discounted reward for each set of agents in each of the test-scenarios. Entries in italic in the same column are not statistically different.

<table>
<thead>
<tr>
<th>Environment</th>
<th>MAP 1</th>
<th>MAP 2</th>
<th>MAP 3</th>
<th>CIT</th>
<th>ISR</th>
<th>MIT</th>
<th>PENT.</th>
<th>SUNY</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMDP</td>
<td>5.787</td>
<td>5.253</td>
<td>6.608</td>
<td>5.305</td>
<td>6.817</td>
<td>3.182</td>
<td>7.606</td>
<td>5.297</td>
</tr>
<tr>
<td>Never Comm</td>
<td>−1.834</td>
<td>0.900</td>
<td>1.917</td>
<td>5.306</td>
<td>6.663</td>
<td>2.959</td>
<td>5.641</td>
<td>5.283</td>
</tr>
<tr>
<td>Comm k = 2</td>
<td>−0.069</td>
<td>1.097</td>
<td>3.001</td>
<td>4.306</td>
<td>5.839</td>
<td>2.141</td>
<td>5.578</td>
<td>4.294</td>
</tr>
<tr>
<td>Comm k = 3</td>
<td>−0.127</td>
<td>1.707</td>
<td>1.564</td>
<td>4.666</td>
<td>6.114</td>
<td>2.426</td>
<td>5.246</td>
<td>4.646</td>
</tr>
<tr>
<td>Comm k = 4</td>
<td>−0.785</td>
<td>1.289</td>
<td>3.295</td>
<td>4.324</td>
<td>5.760</td>
<td>2.106</td>
<td>5.448</td>
<td>4.317</td>
</tr>
</tbody>
</table>

- “Comm k = 2, 3, 4” agents query the state of the other agent every k steps. Except for the different communication frequency, they are otherwise similar to “Always Comm” agents.

Comparisons between these different agents will allow us to analyze (i) the impact that communication costs can have on performance, if communication is not optimized; and (ii) the impact that communication can have in mitigating partial observability. Comparing directly against other methods such as the one of Roth et al. [18] would not be very informative, as they do not attempt to trade off communication costs with task performance.

Results and Discussion

The performance of the 6 agent groups in terms of total discounted reward is summarized in Table 2. As a reference against which to assess the quality of our computed policy we also provide the results for the MMDP optimal policy in the different environments, providing a performance upper bound [15]. The QUERYPOMDP approach performs very favorably, outperforming all other policies and coming close to the MMDP upper bound in several of the tested scenarios.

The results in Table 2 prompt several interesting observations. First, comparing the performance of the MMDP policy against that of the group that never communicates provides an important indication of how critical coordination is in a given scenario. NeverComm agents act individually, disregarding the existence of other agents in the environment. In environments where coordination is critical, NeverComm agents will perform poorly. MMDP agents, on the other hand, always act in a perfectly coordinated manner, in which coordination does not come at a cost. In an environment where little coordination is needed, the difference between these two groups is going to be small. In contrast, scenarios that require significant coordination will cause the performance of the two groups to significantly differ.

From Table 2, we can see that coordination is critical in the smaller environments (Maps 1-3). In the larger environments, such as CIT, MIT and SUNY, coordination is less critical. The results in the smaller environments illustrate the impact of effective communication in mitigating the effects of partial observability. Our method is actually able to attain a performance very close to that of the MMDP agents, even paying for communication. Additionally, our approach uses communication efficiently, since the performance of all other communicating agents is significantly inferior. In contrast, in CIT, MIT and SUNY, non-communicating agents actually attain optimal performance. The difference in performance to the communicating groups can be explained by the communication penalty. Again, in these scenarios, our approach is able to effectively manage communication needs, as it performs similarly to non-communicating agents.

A second observation is that the MMDP performance is an upper bound on the optimal Dec-MDP performance. This means that in those scenarios where our approach performs close to or as well as the MMDP group, we can immediately conclude that it is also performing close to or as well as the optimal Dec-MDP policy. A general comparison of the performance of our method against that of the MMDP group indicates that our method, if not optimal, must be very close to optimal in most scenarios tested. This, in turn, indicates that approximating the behavior of our agents with that of MMDP agents does provide a solid basis for planning.

Summarizing, our results show that, in scenarios with sparse interactions like the ones analyzed, our agents behave approximately as MMDP agents, effectively using communication to mitigate the effects of partial observability.

6. RELATED WORK

In the Dec-POMDP literature, early approaches introduced the idea of transition and reward independence as forms of simplified interactions [3]. Further examples of models with sparse interactions include interaction-driven Markov games [11, 21], distributed POMDPs with coordination locales [23], transition-decoupled POMDPs [24], factored Dec-POMDPs [16], and models relying on event-driven interactions [2].

Our representation is closest to interaction-driven Markov games (IDMGs) [11, 21]. This model leverages the independence between the different agents in a Dec-POMDP to decouple the decision process in significant portions of the joint state-space. In those situations in which the agents interact, IDMGs rely on communication to bring down the computational complexity of the joint decision process. The use of communication to overcome partial observability differentiates this approach from other approaches that also exploit local interactions among the agents. However, communication is assumed to always take place and to be error-free [21]. In our case, we add explicit query actions to each agent’s action repertoire, enabling it to query another agent’s state, subject to certain constraints. For instance, two robots may only be able to share information when they are physically close. Furthermore, we assume that communication is subject to errors and comes at a cost that must be considered.

Explicit communication in multiagent planning was already addressed in [17], where the proposed Com-MTDP model allows agents to explicitly reason about communica-
tion in Dec-POMDP scenarios. However, being a generalization of Dec-POMDPs, it shares the discouraging computational complexity of the latter model. The actual process of communication has been investigated in [8]. Roth et al. [18] propose to exploit a factored Dec-MDP model and policy representation, in which agents query other agents’ local states when this knowledge is required for choosing their local actions. Although this work already seeks to optimize communication, this optimization is conducted parallel with the underlying decision process. Therefore, the cost of communication does not directly translate in the agent’s task performance, as in our proposed approach, rendering the tradeoff between communication costs and benefits unclear.

Another closely related work is that of Wu et al. [26] where communication is used as a means to decrease the planning complexity in Dec-POMDP models. Like in our proposed approach, this work considers that communication may not always be available. However, unlike our approach, this work does not consider explicitly optimization of communication. Finally, Mostafa and Lesser [13] do optimize communication, while considering the presence of communication limitations. However, this optimization is also conducted parallel with the underlying decision process, without directly impacting in the agent’s task performance. Also, none of the aforementioned methods considers noisy communication channels.

A key point in our approach is that, although we use the MMDP policy in our planning, its computation is significantly more efficient than computing a centralized policy for the actual partially observable decision problem. The fact that we plan individually for each agent is somewhat related to several works that use round-robin policy optimization to individually optimize the policy of different agents in Dec-POMDP settings. One of the early examples is the JESP algorithm [14], which also models agents individually as POMDPs, but does not use communication. Round-robin policy optimization has been used to learn communication primitives in Dec-POMDPs whose base models are transition and observation independent [20], but which are coupled through the communication actions agents can choose to execute. In that case, however, agents have to learn when sending a particular message will be beneficial for team performance, which is far from trivial given that the policy of the receiving agent does not exploit the information provided by incoming messages. In our case, however, agents can opt to query other agents’ states, and it is much easier to determine when doing so improves performance. Secondly, we consider a much richer model where agents also “physically” influence each other, instead of only through communication.

7. CONCLUSIONS

In this paper, we analyzed the interplay between sparse interactions and communication in multiagent planning. We observed that, in scenarios where interactions among agents are sparse (i.e., intra-agent action coordination is only infrequently necessary), the distributed execution of an MMDP policy seldom requires full-state information. As such, if each agent is (individually) allowed to query other agents for their local state information when necessary, it may be possible to partly mitigate partial state observability and leverage more efficient planning approaches.

Relying on this insight, we proposed the use of a POMDP model to analyze the communication needs of an agent in a Dec-MDP scenario where the interaction between the agents is sparse. Our model accommodates communication costs and failures—the agent must explicitly reason about these factors in its decision process. QueryPOMDP allows agents to optimize communication, explicitly trading-off its costs with its benefits in terms of the underlying task.

We used our approach to optimize communication in the simple scenario of Fig. 1, where our approach was successfully able to capture the intuition that the fundamental states for coordination are those around the doorway. We further explored the usefulness of this approach in computing policies for larger and more general Dec-MDPs. We built POMDP models for each agent by considering the other agents to behave as if in an MMDP, and use the obtained POMDP optimal policies. Our results show that our agents are able to effectively using communication to mitigate the effects of partial observability, behaving approximately as MMDP agents.

This work raises several interesting research questions. First of all, it would be interesting to generalize these techniques beyond Dec-MDPs, accommodating scenarios in which agents can query other agents’ observations instead of states. Another important issue is that, currently, we assume that the query observation model is known. While this assumption may be reasonable in a planning setting, the performance of the QueryPOMDP agents is critically dependent on its communication capabilities. Therefore, if the agent designer is given the possibility of enabling communication in critical situations, how could these be computed? Early works [5,9,10] proposed methods that allow agents to automatically learn where information about others may be useful. A more recent method proposed the use of constrained optimization to explicitly determine when communication can be used to construct informative local beliefs [12]. An interesting avenue for future work is the extension of such approaches to the QueryPOMDP setting.

Finally, one other important aspect that will be explored in future work is the extent to which sparsity of interaction affects the optimality of our planning approach. In fact, while our results indicate that our approach is able to effectively leverage the particular structure that may exist in multiagent settings, it is not clear how the performance of our approach depends on the sparsity of interactions. In fact, our method leverages such existing structure only implicitly. Characterizations such as the one in [25] may provide insights into which situations can be successfully addressed using our approach.

8. ACKNOWLEDGMENTS

The authors acknowledge the helpful suggestions by the anonymous reviewers. This work was funded in part by Fundação para a Ciência e a Tecnologia (INESC-ID multimodal funding) through the PIDDAC Program funds and the project CMU-PT/SIA/0023/2009 under the Carnegie Mellon-Portugal Program. M.S. is funded by the FPT Marie Curie Actions Individual Fellowship #275217 (FP7-PEOPLE-2010-IEF).

9. REFERENCES


Tree-based Solution Methods for Multiagent POMDPs with Delayed Communication

Frans A. Oliehoek
MIT CSAIL / Maastricht University
Maastricht, The Netherlands
frans.oliehoek@maastrichtuniversity.nl

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

ABSTRACT
Planning under uncertainty is an important and challenging problem in multiagent systems. Multiagent Partially Observable Markov Decision Processes (MPOMDPs) provide a powerful framework for optimal decision making under the assumption of instantaneous communication. We focus on a delayed communication setting (MPOMDP-DC), in which broadcasted information is delayed by at most one time step. This model allows agents to act on their most recent (private) observation. Such an assumption is a strict generalization over having agents wait until the global information is available and is more appropriate for applications in which response time is critical. From a technical point of view, MPOMDP-DCs are quite similar to MPOMDPs. However, value function backups are significantly more costly, and naive application of incremental pruning, the core of many state-of-the-art optimal POMDP techniques, is intractable. In this paper, we show how to overcome this problem by demonstrating that computation of the MPOMDP-DC backup can be structured as a tree and introducing two novel tree-based pruning techniques that exploit this structure in an effective way. We experimentally show that these methods have the potential to outperform naive incremental pruning by orders of magnitude, allowing for the solution of larger problems.

1. INTRODUCTION
This paper focuses on computing policies for multiagent systems (MAS) that operate in stochastic environments and that share their individual observations with a one step delay. While dynamic programming algorithms date back to the seventies [29, 3, 10, 12], computational difficulties have limited the model’s applicability. In particular, the backup operator under delayed communication has an additional source of complexity when compared to settings with instantaneous communication. In this paper, we take an important step in overcoming these challenges by showing how this additional complexity can be mitigated effectively.

Our efforts are part of the greater agenda of multiagent planning under uncertainty. The task faced by a team of agents is complicated by partial or uncertain information about the world, as well as stochastic actions and noisy sensors. Especially settings in which agents have to act based on their local information only have received a large amount of attention in the last decade [22, 4, 25, 30]. The Dec-POMDP framework [5] can be used to formulate such problems, but decentralization comes at a high computational cost: optimally solving a Dec-POMDP is NEXP-complete.

Communication can be used to mitigate the problem of decentralized information: when agents can share their local observations, each agent can reason about the optimal joint action given this global information state, called joint belief, and perform its individual component of this joint action. That is, when instantaneous communication is available, it allows one to reduce the problem to a special type of POMDP [22], called a Multiagent POMDP (MPOMDP), which has a lower computational complexity than a Dec-POMDP (PSPACE-complete) and will generally lead to a joint policy of a higher quality.

However, assuming instantaneous communication is often unrealistic. In many settings communication channels are noisy and synchronization of the information states takes considerable time. Since the agents cannot select their actions without the global information state (called ‘joint belief’), this can cause unacceptable delays in action selection. One solution is to assume that synchronization will be completed within \( k \) time steps, selecting actions based on the last known joint belief. Effectively, this reduces the problem to a centralized POMDP with delayed observations [2]. However, such formulations are unsuitable for tasks that require a high responsiveness to certain local observations, such as applications for collision avoidance, ambient intelligence or other more general forms of human-computer interaction and distributed surveillance. Another prime example is decentralized protection control in electricity distribution networks by so-called Intelligent Electronic Devices (IED). Such IEDs not only decide based on locally available sensor readings, but can receive information from other IEDs through a communication network with deterministic delays [32]. When extreme faults such as circuit or cable failures occur, however, no time can be wasted waiting for information from other IEDs to arrive.

We therefore consider an alternative execution model that assumes that while communication is received with a delay of one stage, the agents can act based on their private observation as well as the global information of the last stage.\(^1\) That is, all agents broadcast their individual observations, but in contrast to a delayed observation POMDP they do not wait with action selection until this communication phase has ended. Instead, they go ahead taking an action based

\(^1\)This is also referred to as a one step delayed sharing pattern in the decentralized control literature.
on the global information of the previous stage and the individual observation of the current stage. This results in a strict generalization of the delayed observation POMDP.

Thereby, this model uses communication to mitigate the computational complexity and achieve higher performance than Dec-POMDPs, while preventing delays in action selection. Moreover, the optimal solution of the MPOMDP-DC is useful in settings with longer communication delays [27], can be used as a heuristic in Dec-POMDPs [26], or to upper bound the expected value of MASs without communication (e.g., in the setting of communication channels) [10, 20].

From a technical perspective, the MPOMDP is equivalent to a POMDP with a centralized controller, which means that all POMDP solution methods apply. Many of these exploit the fact that the value function is piecewise-linear and convex (PWLC) over the joint belief space. Interestingly, the value function of an MPOMDP-DC exhibits the same property [12]. As such one would expect the same computational methods to be applicable. However, incremental pruning (IP) [6], that performs a key operation, the so-called cross-sum, more efficiently, is not directly able to achieve the same improvements for MPOMDP-DCs. A problem is the need to loop over a number of decision rules that is exponential both in the number of agents and in the number of observations. This means that the backup operator for MPOMDP-DC model is burdened with an additional source of complexity when compared to the regular POMDP backup.

In this paper, we target this additional complexity by proposing two novel methods that operate over a tree structure. The first method, called TBP-M for tree-based pruning with memoization, avoids duplicate work by caching the result of computations at internal nodes and thus accelerates computation at the cost of memory. The second, branch and bound (TBP-BB), is able to avoid unnecessary computation by making use of upper and lower bounds to prune parts of the tree. Therefore it provides a different space/time trade-off. The empirical evaluation of our proposed methods on a number of test problems shows a clear improvement of a naive application of incremental pruning. TBP-M provides speedups of up to 3 orders of magnitude. TBP-BB does not consistently outperform the baseline, but is still able to provide large speedups on a number of test problems, while using little memory.

2. BACKGROUND

Here we present background on the relevant formal models and their solutions methods.

2.1 Models

In this section we formally introduce the multiagent POMDP (MPOMDP) model and describe the planning problem.

**Definition 1.** A multiagent partially observable Markov decision process \( M = \langle n, S, A, P, R, O, O, h, O, C \rangle \) consists of

- a finite set of \( n \) agents;
- \( S = \times_i S_i \) is a finite set of states;
- \( A = \times_i A_i \), is the set \( \{ a_1, \ldots, a_J \} \) of \( J \) joint actions. \( A_i \) is the set of actions available to agent \( i \). Every time step one \( a = (a_1, \ldots, a_n) \) is taken;
- \( P \), the transition function. \( P^a(s'|s) \) is the probability of transferring from \( s \) to \( s' \) under \( a \);
- \( R \) is the reward function. We write \( R^a(s) \) for the reward accumulated when taking \( a \) from \( s \);
- \( O = \times_i O_i \) is the set \( \{ o_1, \ldots, o_K \} \) of \( K \) joint observations. Every stage an \( o = (o_1, \ldots, o_n) \) is observed;
- \( O \) is the observation function. We write \( O^a(o|s) \) for the probability of \( o \) after taking \( a \) and ending up in \( s \);
- \( h \) is the horizon, the number of time steps or stages that are considered when planning.

The special case with 1 agent is called a (regular) POMDP. Execution in an MPOMDP is as follows. At every stage \( t \), each agent \( i \):

1. observes its individual \( o_i \),
2. broadcasts its own observation \( o_i \),
3. receives observations \( o_{-i} = (o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n) \) from the other agents,
4. uses the joint observation \( o^t = (o_i, o_{-i}) \) and previous joint action \( a^{t-1} \) to update the new joint belief \( b^t = BU(b^{t-1}, a^{t-1}, o^t) \),
5. looks up the joint action for this stage in the joint policy \( a^t = \pi(b^t) \),
6. and executes its component \( a_i^t \).

The belief update function \( BU \) updates the previous joint belief using Bayes rule:

\[
b^t(s') = BU(b, a, o) = \frac{1}{P^a(o|b)} O^a(o|s') \sum_{s} P^a(s'|s) b(s),
\]

where \( P^a(o|b) \) is a normalizing factor. We denote the set of all joint beliefs by \( B \). In the remainder of this paper, we will refer to a joint belief simply as ‘belief’.

The joint policy \( \pi = (\delta^0, \delta^1, \ldots, \delta^{h-1}) \) is a sequence of joint decision rules mapping beliefs to joint actions. The goal of the multiagent planning problem for an MPOMDP is to find an optimal joint policy \( \pi^* \) that maximizes the total expected sum of rewards, defined an arbitrary \( \pi \) as

\[
V(\pi) = E \left[ \sum_{t=0}^{h} R(s^t, a^t) \mid \pi, b^0 \right],
\]

where \( b^0 \) is the initial state distribution. In this paper we will consider planning over a finite horizon \( h \). While (1) optimizes for a given \( b^0 \), we will be concerned with computation of all the optimal \( \pi \) for all possible \( b^0 \), which is beneficial for tasks in which \( b^0 \) is not known in advance.

The value can be expressed recursively as a function of the joint belief space. In particular, the \( h - t \) steps-to-go action-value of \( b \) is

\[
Q^t(b, a) = R^a_0(b) + \sum_a \sum_{o} P^a(o|b) \max_{a'} Q^{t+1}(b', a'),
\]

where \( R^a_0(b) = \sum_o R^a_o(s|b) \) and \( b' = BU(b, a, o) \). An optimal joint decision rule for stage \( t \), \( \delta^t \), selects the maximizing \( a \) for each \( b \) and thereby defines the value function:

\[
V^t(b) = \max_a Q^t(b, a) = Q^t(b, \delta^t(b)).
\]

When no communication is available, the multiagent planning problem can be formalized as a Dec-POMDP. This problem is significantly different as agents must base their decisions solely on local observations and have no means to compute the joint belief resulting in NEXP-complete complexity.

In the remainder of this paper we focus on settings with delayed communication.
Definition 2. An MPOMDP with delayed communication (MPOMDP-DC) is an MPOMDP where communication is received with a one-step delay.

Execution in an MPOMDP-DC is as follows. At decision point $t$, each agent $i$:

1. has received the previous-stage observations $o_{i,t-1}$ and actions $a_{i,t-1}$ of the other agents,
2. observes its individual observation $o_{i,t}$,
3. computes $b_{t-1} = BU(b_{t-2}, a_{t-2}, o_{t-1})$, using the previous joint observation $o_{t-1} = (o_{1,t-1}, o_{2,t-1})$ and joint action $a_{t-2}$ (remembered from stage $t-2$ when it received $o_{i,t-2}$),
4. looks up the individual action for this stage in the individual policy $a_{i,t} = \pi_i(b_{t-1}, o_{t-1}, t)$,
5. broadcasts its own observations $o_{i,t}$ and action $a_{i,t}$,
6. and executes $a_{i,t}$.

From this it is clear that there are quite a few differences with the MPOMDP formulation. Most notably, the form of the joint policy is different. In an MPOMDP-DC a joint decision rule specifies an individual decision rule for each agent $\delta_i = (\delta_{i1}, \delta_{i2}, \delta_{i3})$, where each $\delta_{ij} : B^{i-1} \times A \times O_i \rightarrow A_i$ is a mapping from a $(b_{i-1}, a_{i-1}, o_{i})$-tuple to an individual action $a_{i,t}$.

As such, it may come as a surprise that we can still define the optimal value of an MPOMDP-DC as a function of joint beliefs:

$$Q^t(b,a) = R^a(b) + \max_{\beta \in B} \sum_{o} P^o(b|a)Q^{t+1}(b', \beta(o)),$$

where $B$ is the set of decentralized control laws $\beta = (\beta_1, \ldots, \beta_n)$ which the agents use to map their individual observations to actions: $\beta(o) = (\beta_1(o_1), \ldots, \beta_n(o_n))$. Essentially we have decomposed a joint decision rule $\delta^t$ into a collection of $\beta$ one for each $(b,a)$-pair. In fact, the maximization that (3) performs for each such $(b,a)$-pair corresponds to solving a collaborative Bayesian game [17] or, equivalently, a Team Decision Problem and is NP-complete [28]. We also note that the set of possible $\beta$ is a strict super set of the set of joint actions; control laws that ignore the private observation and just map from the previous joint belief (and joint action) to a joint action are included. As such, this approach gives a strict improvement over assuming delayed joint observations [2].

### 2.2 Solution Methods

For both instantaneous and delayed communication, we can define a value function over joint beliefs. The problem with computing them, however, is that these spaces are continuous. Here we discuss methods that overcome this problem. We focus on the case of MPOMDPs and defer the necessary extensions for MPOMDP-DC to the next section.

Many methods for solving a (multiagent) POMDP make use of the fact that (2) is piecewise-linear and convex (PWLC) over the belief space. That is, the value at stage $t$ can be expressed as a maximum inner product with a set of vectors:

$$Q^t(b,a) = \max_{v_{a} \in \mathcal{V}_a} b \cdot v_{a} = \max_{v_{a} \in \mathcal{V}_a} \sum_s b(s)v_{a}(s).$$

We will also write $\mathcal{V}^t = \bigcup_{a \in A} \mathcal{V}^t_a$. We restrict the value function to the complete set of vectors that represent the value function. Each of these vectors $v_a$ represents a conditional plan (i.e., policy) starting at stage $t$ with joint action $a$ and has the following form:

$$v^t_a = R^a + \sum_{o} g_{ao} \beta(o),$$

with $g_{ao}$ the back-projection of $v^t_a$, the $i$-th vector in $\mathcal{V}^{t+1}$. The set of such gamma vectors is denoted $\mathcal{G}_{ao}$.

Monahan’s algorithm [16] simply generates all possible vectors by, for each joint action $a$, for each possible observation selecting each possible next-stage vector:

$$\mathcal{V}_a^t = \{ R^a \} \oplus \mathcal{G}_{ao^1} \oplus \cdots \oplus \mathcal{G}_{ao^K}$$

where the cross-sum $A \oplus B = \{ a + b \mid a \in A, b \in B \}$.

A problem in this approach is that the number of vectors generated by it grows exponentially; at every backup, the algorithm generates $|\mathcal{V}^{t+1}| = \sum_j |\mathcal{V}_j^t|\K$ vectors. However, many of these vectors are dominated, which mean that they do not maximize any point in the belief space. Formally, a vector $v \in \mathcal{V}$ is dominated if

$$\exists b \ s.t. \ b \cdot v > b \cdot v', \ \forall v' \in \mathcal{V}.$$

The operation Prune removes all dominated vectors by solving a set of linear programs [6, 8]. That way, the parsimonious representation of $\mathcal{V}^t$ can be computed via

$$\mathcal{V}_a^t = \text{Prune}(\mathcal{V}_a^t \cup \cdots \cup \mathcal{V}_a^K),$$

$$\mathcal{V}^t_a = \text{Prune}(\mathcal{V}_a^t \cup \mathcal{V}_a^{t+1} \cup \cdots \cup \mathcal{V}_a^K)$$

$$\mathcal{V}_{ao} = \text{Prune}(\mathcal{G}_{ao}).$$

The technique called incremental pruning (IP) [6] speeds up the computation of the value function tremendously by realizing that (9), the bottleneck in this computation, can be re-written to interleave pruning and cross-sums:

$$\text{Prune}(\mathcal{V}_a^t \oplus \cdots \oplus \mathcal{V}_{ao}^K) =$$

$$\text{Prune}(\text{Prune}(\text{Prune}(\cdots \oplus \mathcal{V}_a^t \oplus \mathcal{V}_{ao} \oplus \cdots) \oplus \mathcal{V}_{ao}^K)).$$

That is, we can first prune parts of the larger cross sum to yield the same result.

### 3. COMPUTING DC VALUE FUNCTIONS

In this section, we show how the methods mentioned in the previous section can be extended to the MPOMDP-DC.

#### 3.1 Vector Representations

As an MPOMDP, we can represent the value function under delayed communication using vectors [12]. However, in the MPOMDP-DC case, not all combinations of next-stage vectors are possible; the actions they specify should be consistent with an admissible decentralized control law $\beta$. That is, we define vectors $g_{ao^{t+1}} \in \mathcal{G}_{ao^{t+1}}$ analogously to (6), but now restricting $v^t_a$ to be chosen from $\mathcal{V}_a^{t+1}$. From these we construct

$$\mathcal{V}_a^t = \{ R^a + g_{ao^1}\beta(o^1) + g_{ao^2}\beta(o^2) + \cdots + g_{ao^K}\beta(o^K)$$

$$\mid \forall \beta, g_{ao\beta(o)} \in \mathcal{G}_{ao\beta(o)} \}$$

$$= \{ R^a \} \oplus \mathcal{G}_{ao^1}\beta(o^1) \oplus \cdots \oplus \mathcal{G}_{ao^K}\beta(o^K)$$

(12)
Note that it is no longer possible to collect all the vectors in one set $V^t$, since we will always need to discriminate which joint action a vector represents.

In the following, we will also represent a $\beta$ as a vector of joint actions $(a_1, \ldots, a_K)$, where $a_k$ denotes the joint action selected for the $k$-th joint action.

Proposition 1. The (not pruned) set $V^t_{a,DC}$ of vectors under delayed communication is a strict subset of the set $V^t_{a,P}$ of MPOMDP vectors: $\forall a \ V^t_{a,DC} \subseteq V^t_{a,P}$.

Proof. To see this, realize that
\[
\mathcal{G}_{ao} = \bigcup_{a'} \mathcal{G}_{a'o}
\]
and that therefore (7) can be rewritten as
\[
y_{a,P}^t = \bigcup_{a} \bigcup_{a'} \bigcup_{t} \bigcup_{t'} \bigcup_{a''} \bigcup_{t''} \bigcup_{a'''} \bigcup_{t'''} \bigcup_{a''''} \bigcup_{t''''} \bigcup_{a'''''} \bigcup_{t'''''},
\]
where the observation follows from the fact that the set of admissible $\beta \in B$ is a subset of $A^K$; each $\beta$ can be represented as a vector of joint actions $(a_1, \ldots, a_K)$, but not every such vector is a valid $\beta$. □

This means that the number of vectors grows less fast when performing exhaustive generation. However, an effective method for doing the backup, such as incremental pruning for POMDPs, has not been developed.

3.2 Naive Incremental Pruning

An obvious approach to incremental pruning in MPOMDP-DCs is given by the following equations, which we will refer to as NAIVE IP:

\[
y_{a,P}^t = \operatorname{Prune} \left( \bigcup_{\beta \in B} y_{a,\beta}^t \right),
\]
\[
y_{a,\alpha}^t = \operatorname{Prune} \left( \{ R^t \} \oplus G_{a,\alpha}^t \oplus \cdots \oplus G_{a,\alpha}^t \right),
\]
\[
n_{a'o} = \operatorname{Prune} \left( G_{a'o} \right),
\]

where (15) uses incremental pruning.

Note that it is not possible to prune the union over joint actions as in (8), because this would correspond to a maximization over joint actions in (3). This is not possible, because under delayed communication, the joint action is that joint action that was taken in the previous time step. In particular, the sets of vectors $Y_{a,P}^t$ are used as follows. At a stage $t$, an agent knows $b^{t-1}$ and $a^{t-1}$. It uses this information to determine the vector $v \in Y_{a,P}^{t-1}$ that maximizes $v \cdot b^{t-1}$. It retrieves the maximizing $\beta$ for $v$, and executes $\beta(a)$.\n
There are two problems with the computation outlined above. First, it iterates over all possible $\beta \in B$, which is exponential both in the number of agents and in the number of observations. Performing the iteration in this way in practice means that performing one backup takes nearly a factor $|B|$ as much as a POMDP backup. Second, it performs a lot of duplicate work. E.g., there are many $\beta$ that specify $\beta(a^t) = a$, $\beta(a^t) = a'$, but for each of them $\operatorname{Prune} (G_{a,\alpha}^t \oplus G_{a',\nu}^t)$ is recomputed.

It is tempting to side step the problem by generating the POMDP value function using incremental pruning and then throw away the vectors that do not correspond to a valid $\beta$, i.e., throw away the vectors that are not admissible. This, however, is not correct: many vectors corresponding to valid $\beta$s may have been pruned away because they were dominated by (a combination of) other vectors that are not admissible, which clearly is not desirable.

4. TREE-BASED PRUNING

In order to overcome the drawbacks of the naive approach outlined above, we propose a different approach. Rather than creating sets $V_{a,P}^t$ for each $\beta \in B$, we directly construct
\[
V_{a}^t = \operatorname{Prune} \left( \bigcup_{\beta \in B} \{ R^t \} \oplus G_{a,\alpha}^t \oplus \cdots \oplus G_{a,\alpha}^t \right).
\]

As mentioned, we can interpret $\beta$ as a vector of joint actions. This allows us to decompose the union over $\beta$s into dependent unions over joint actions, as illustrated in Fig. 1(a).

The resulting equation (17) defines a computation tree, as illustrated in Fig. 1(b) in the context of a fictitious 2-action $(x$ and $y)$ 2-observation (1 and 2) MPOMDP-DC. The root of the tree, $V_a^t$, is the result of the computation. There are two types of internal, or operator, nodes: crossover and union. All the leaf nodes are sets of vectors. An operator node $n$ takes as input the sets from its children, computes $V_n$, the set resulting from application of its operator, and propagates this result up to its parent. When a union node is the $j$-th union node on a path from root to leaf, we say it has depth $j$. A depth-$j$ union node performs the union over $a(j)$ and thus has children corresponding to different assignments of a joint action to $a'(j)$ (indicated by the gray bands). It is important to realize that the options available for $a(j)$ depend on the action choices $(a_1, \ldots, a_{j-1})$ made higher up in the tree; given those earlier choices, some $a(j)$ may lead to conflicting individual actions for the same individual observation. Therefore, while there are 4 children for $\cup_{i \in 1,2}$, union nodes deeper in the tree have only 2 or even just 1.

Now, to compute (18) we propose tree-based (incremental) pruning (TBP): it expands the computation tree and, when the results are being propagated to the top of the tree, prunes dominated vectors at each internal node. However, Fig. 1(b) shows another important issue: there are identical sub-trees in this computation tree, as indicated by the dashed green ovals, meaning that we would be doing unnecessary work. We address this problem by memoization, i.e., caching of intermediate results, and refer to the resulting method as TBP-M. Note that the sub-tree under a node is completely characterized by a specification of which joint action assignments are still possible for the unspecified joint observations. For instance, the nodes inside the ovals can characterize as $\langle \cdot, \cdot, \cdot, \cdot \rangle$, the set of $(x,y)$ wildcards and similar for $\langle \cdot, \cdot, \cdot \rangle$. We call such a characterization the ID string and it can be used as the key into a lookup table. This way we only have to perform the computation just once for each ID string.

5. BRANCH & BOUND

Here we introduce tree-based pruning using branch and bound (TBP-BB), which is a different method to overcome the problems of naive incremental pruning as described in Section 3.2. The motivation of introducing another method is twofold. First, while TBP-M effectively addresses the issue of performing duplicate work (i.e., it addresses the second problem in Section 3.2), it does not fully address the
\[ V_a^f = \bigcup_{(a_1, \ldots, a_k) \in B} \left( \sum_{1}^{k} G_{ao_1 a_1(1)} \oplus \cdots \oplus G_{ao_k a_k(k)} \right) \]

\[ = \sum_{1}^{k} G_{ao_1 a_1(1)} \oplus \cdots \oplus G_{ao_k a_k(k)} \]

\[ = \left( \sum_{1}^{k} G_{ao_1 a_1(1)} \oplus \cdots \oplus G_{ao_k a_k(k)} \right) \cup \left( \sum_{1}^{k} G_{ao_1 a_1(1)} \oplus \cdots \oplus G_{ao_k a_k(k)} \right) \]

\[ = \left( \sum_{1}^{k} G_{ao_1 a_1(1)} \oplus \cdots \oplus G_{ao_k a_k(k)} \right) \cup \left( \sum_{1}^{k} G_{ao_1 a_1(1)} \oplus \cdots \oplus G_{ao_k a_k(k)} \right) \]

\[ \vdots \]

\[ \text{(a) Rewriting (18). } B|_{a_1(1) a_2(2)} \text{ denotes the set of } \beta \text{ consistent with } a_1(1), a_2(2), \text{ and } a_1(1) \cdots a_{(k-1)} \text{ denotes the set of joint actions (for the } k\text{-th joint observation) that result in a valid } \beta. \]

---

**Figure 1**: (a) By decomposing the joint union and moving it inwards, we create a tree-formed computation structure. (b) The computation tree of \( V_a^f \). See text for description.

other problem. Every leaf of a depth-\( K \) union node corresponds to exactly one \( \beta \). Even though we hope to avoid visiting many leaves due to memoization, the number of \( \beta \) and thus number of sub-trees that will need to be computed at least once may still be prohibitive. TBP-BB can overcome this problem by pruning complete parts of the search tree, even if they have not been computed before. Second, as mentioned above, TBP-M needs to cache results in order to avoid duplicate work which may lead to memory problems. TBP-BB does not perform caching, but still can avoid expanding the entire tree, creating an attractive alternative to trade off space and time complexity.

Branch and bound (BB) is a well-known method to prune parts of search trees. It computes an \( f \)-value, for each visited node \( n \) in the tree via \( f(n) = g(n) + h(n) \), where \( g(n) \) is the actual reward achieved up to the node, and \( h(n) \) is a heuristic estimate of the reward to be found in the remainder of the tree. When \( h \) is admissible (a guaranteed overestimation), so is \( f \). This means that if the \( f \)-value of a node \( n \) is less than the value of the maximum lower bound \( l \) (i.e., the best full solution found so far), we can prune the sub-tree under \( n \). Since we are not concerned with a search-tree, but rather a computation tree, this technique can not be applied directly. Still, we can generalize the idea of BB to be applicable to our setting, which requires specifying \( f, g \) and \( h \) as PWLC functions and comparing them to \( l \), the lower bound function: the PWLC function over belief space induced by the set \( L \) of all found non-dominated vectors \( v_i^f \).

This idea is illustrated in Fig. 2. The first insight that this figure illustrates is that, while the computation tree works bottom-up, we can also interpret it as top-down: by associating the null-vector \( \theta \) with the root node, we can now pass
If \( f = g + h \) does not exceed the lower bound \( l \) (induced by already found full vectors) at any point in the belief space, the node does not have to be expanded further.

In a similar way, the PWLC heuristic function \( h \) is represented by a set of vectors \( H \). The function should give a guaranteed overestimation of \( V_a \) (as formalized below) and we propose to use the POMDP vectors. That is, for a node at depth \( j \) (for which the first \( j \) joint observations' gamma vector sets are used in the cross-sum to compute \( G \)), we specify:

\[
H_j = \text{Prune}(V^f_{ao} + V^f_{b} \cup \cdots \cup V^f_{aK}) \tag{19}
\]

This can be pre-computed using incremental pruning.

With a slight abuse of notation, we will write \( f_a \) and \( F_a \) for the \( f \)-function at node \( n \) and the set of vectors representing it. We will say that \( f_a \) and \( F_a \) are admissible if they are a guaranteed over-estimation of the actual complete (alpha) vectors produced by this node. That is if

\[
\forall b \exists v \in F_a, V^a_f, V^b_{aK} : b \cdot v \geq b \cdot v'.
\]

Or, if we use \( h_a \) to denote the function induced by \( V_a \), the actual set of vectors computed by \( n \), we can more simply state this requirement as

\[
\forall b \quad f_a(b) \geq g_a(b) + h_a(b). \tag{21}
\]

**Theorem 1.** Let \( n \) be a search node at depth \( j \), then \( F_n = G_n \oplus H_j \), where \( H_j \) is defined as the POMDP heuristic (19), is admissible.

**Proof.** By \( F_n = G_n \oplus H_j \) we have that the induced function \( f_a(b) = g_a(b) + h_a(b) \). Therefore we only need to show that \( \forall b \, h_j(b) \geq h_a(b) \). This clearly is the case because \( h_j \) is induced by a cross-sum of POMDP back projections (19), while the latter is induced by cross-sums of DC back projections (union of \( V_{ao}^{f+1} \oplus \cdots \oplus V_{aoK}^{f} \)), for a subset \( B' \) of \( \beta \) and the former are supersets of the latter \((V^a_{ao} \supset V^f_{aoK,a})\) as indicated by (13).

Given \( g, h \) we want to see if we need to further expand the current node. Clearly, \( f = g + h \) is represented by the upper surface implied by the set \( F = \text{Prune}(G \oplus H) \). Therefore, to see if the current node could generate one or more vectors that are not dominated by the set \( L \) of full vectors found so far, we need to check if there is a \( v \in F \), such that \( \exists b \) such that \( v \cdot b > w \cdot b \), \( \forall w \in L \). That is, we simply need to check for each \( v \in F \) if it is dominated by the set of vectors \( L \) representing \( l \). This can be done using the standard LP for checking for dominance [6,8]. At the bottom of the tree, we add any non-dominated vectors to \( L \).

A final note on the TBP-BB method here is that, while it may be able to avoid computation of complete branches when the heuristics are tight enough, it does not perform memoization and therefore may need to perform some duplicate work in different parts of the tree. However, this does give this method the advantage of limited memory requirements.

### 6. EXPERIMENTS

We tested our methods on a set of six problems: Dec-Tiger, OneDoor, GridSmall, Cooperative Box Pushing, Dec-Tiger with Creaks [9], and MeetingGrid2x23. The main characteristics of these problems can be found in Table 1(a).4 Of particular interest is the right-most column showing the number of \( \beta \) (denoted \(|B|\)) for each problem, which is a key indicator of its complexity. As all methods compute optimal value functions, we only compare computation times.

Table 2 shows timing results for all six problems, for a set of planning horizons (depending on the problem). We can see that for all domains TBP-M outperforms Naive IP, often by an order of magnitude and up to 3 orders of magnitude. TBP-BB performs somewhat worse, but as noted before, requires much less memory.

We also compared against TBP-\textit{noM}: a strawman version of TBP-M that does not perform any memoization and re-computes duplicate parts of the tree. It allows us to see the effect of tree-based pruning, without the extra speedups provided by memoization: except for a very small problem (Dec-Tiger(5)), memoization significantly speeds up computations. The results also show that TBP-\textit{noM} still is faster than Naive IP on almost all problems.

Table 1(b) provides some statistics for TBP-M. The “Memoization” columns show how often the memoization procedure can retrieve a solution (“\# hits”), both the absolute number as well as as a percentage of the total number of calls to the cache. The “Nodes” columns show how many nodes in the search tree are actually being visited (“\# visited”), as well compared to the total number of nodes in the tree. We can see that as the problem domains grow larger in terms of \(|B|\), the percentage of successful cache hits goes down somewhat. More importantly, however, the percentage of visited nodes decreases more rapidly, as larger subtrees have been cached, leading to larger computational savings. Indeed, when comparing TBP-M vs. TBP-\textit{noM} in Table 2,

3Courtesy of Jilles Dibangoye.

---

**Figure 2:** A part of the search tree from Fig. 1(b) illustrating the heuristics for a search node. Gamma sets are abbreviated by simply \( G \). If \( f = g + h \) does not exceed the lower bound \( l \) (induced by already found full vectors) at any point in the belief space, the node does not have to be expanded further.

---

**Table 1:**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Naive IP</th>
<th>TBP-M</th>
<th>TBP-BB</th>
<th>TBP-\textit{noM}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dec-Tiger</td>
<td>1000</td>
<td>300</td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>OneDoor</td>
<td>2000</td>
<td>500</td>
<td>200</td>
<td>1000</td>
</tr>
<tr>
<td>GridSmall</td>
<td>3000</td>
<td>1000</td>
<td>500</td>
<td>2000</td>
</tr>
<tr>
<td>Cooperative Box Pushing</td>
<td>5000</td>
<td>1500</td>
<td>700</td>
<td>2500</td>
</tr>
<tr>
<td>Dec-Tiger with Creaks</td>
<td>7000</td>
<td>2000</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>MeetingGrid2x2</td>
<td>10000</td>
<td>3000</td>
<td>1500</td>
<td>5000</td>
</tr>
</tbody>
</table>
Table 1: (a) Overview of several characteristics of the problem domains. All problems have 2 agents. (b) Statistics for TBP-M (independent of \( h \)). (c) Statistics for TBP-BB (for a particular \( h \)).

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
</tr>
<tr>
<td>Dec-Tiger</td>
</tr>
<tr>
<td>OneDoor</td>
</tr>
<tr>
<td>GridSmall</td>
</tr>
<tr>
<td>MG2x2</td>
</tr>
<tr>
<td>D-T Creaks</td>
</tr>
<tr>
<td>Box Push.</td>
</tr>
</tbody>
</table>

| Problem | TBP-M | TBP-BB | NAIVE IP | TBP-nOM |
|---------|
| Dec-Tiger(5) | 0.13 | 0.09 | 0.23 | 0.09 |
| Dec-Tiger(10) | 0.31 | 0.43 | 0.73 | 0.33 |
| Dec-Tiger(15) | 0.98 | 1.44 | 2.54 | 1.19 |
| OneDoor(3) | 53.64 | 1546.73 | 304.72 | 56.53 |
| GridSmall(2) | 3.93 | 125.45 | 64.03 | 3.80 |
| MG2x2(2) | 171.07 | 2689.35 | 38209.03 | 516.03 |
| MG2x2(3) | 640.70 | 11370.40 | 1499.43 |
| MG2x2(4) | 1115.06 | 21242.30 | 2813.10 |
| D-T Creaks(2) | 63.14 | 93.16 | 109.27 | 121.99 |
| D-T Creaks(3) | 149.06 | 172.79 | 1505.17 | 471.57 |
| D-T Creaks(4) | 203.44 | 292.67 | 4030.47 | 1150.69 |
| D-T Creaks(5) | 286.53 | 619.25 | 8277.32 | 2046.73 |
| Box Push.(2) | 132.13 | 6663.04 | 1832.98 | 1961.38 |

we can see that the gap between them grows as \( |B| \) increases (c.f. Table 1(a)).

Table 1(c) shows the amount of nodes visited by TBP-BB during the search for a particular horizon. There is a clear correlation with the TBP-BB in Table 2; domains in which TBP-BB can hardly prune any branches (OneDoor and Box Push), its performance is much worse than TBP-nOM, due to the overhead of maintaining bounds. However, a tighter heuristic could change this picture dramatically. Additionally, computing the heuristic \( H_i \) is relatively costly in some domains: for GridSmall(2) it takes 83.95s, and 1280.23s for OneDoor(3).

Finally, note that computing the heuristic (19) using incremental pruning just corresponds to computing (11) and therefore to doing a POMDP backup. However, we can see that TBP-M solves the mentioned problem instances in 3.93s resp. 53.64s. That is, the DC backup is faster than the POMDP backup in these instances. While this is not a trend for all domains, this does suggest that the DC backup no longer inherently suffers from an additional complexity.

7. DISCUSSION & RELATED WORK

Here we discuss our approach, providing pointers to related work and possible directions of future work where possible.

Our experimental evaluation show very favorable results for TBP-M, but the results for TBP-BB are not that great in comparison. While the latter improves over NAIVE IP, it is only able to improve over TBP-nOM (i.e., simple execution of the entire computation tree) for one domain. A significant problem seems to be that the heuristic is not tight enough in many cases. In future research we plan to analyze what causes the big differences in effectiveness of TBP-BB between domains, which may also lead to new insights to improve the heuristics. Another interesting idea is to apply TBP-BB in approximate settings by using tighter (non-admissible) heuristics.

The suitability of the MPOMDP-DC model depends on the ratio of expected duration to synchronize the joint belief state and the duration of a joint action. It is certainly the case that there may be many situations where synchronization is expected to take multiple stages and in such settings our model will also lead to agents deferring their actions, leading to delays. However, even for such cases the techniques developed here may be useful: the one-step delayed can be used as a part of a solution with longer communication delays [27], or to provide upper bounds for such situations [19].

The MPOMDP-DC model is particularly useful for applications that require a low response time. In such applications it is also difficult to perform planning online [23], which motivates the need for offline planning as presented in this paper. We point out, however, that even in cases where online planning is feasible, having a value function to use as a heuristic for the leafs of the search tree is a valuable asset.

While IP entails a choice for the regular (vector-based) backup, an interesting other direction is the exploration of point-based backups. While we do not expect that this can directly lead to further improvements to the exact backup—IP is empirically faster than the (point-based) witness algorithm [13]—point based methods have led to state-of-the-art results for approximate POMDP methods [21, 15]. While a point-based value iteration for MPOMDP-DC has been proposed [18, 27], many questions remain open. For instance, in order to get any kind of quality guarantees for such methods, we envision research should investigate how to efficiently compute upper bounds for the DC setting. Moreover, it is still unclear how to efficiently perform the point-based backup itself, although there have been recent advances [17]. We expect that it will be possible to draw on work performed on point-based backups for the Dec-POMDP [1, 7, 14, 31].

In fact, this connection with Dec-POMDPs also works the other way around. An important direction of future work is to investigate whether it is possible to transfer TBP to Dec-POMDPs. As mentioned earlier, MPOMDP-DC value functions have also been used as upper bounds for the solution of Dec-POMDPs thereby significantly increasing the size of problems that can be addressed [19, 26]. However, there may be more direct ways in which our methods can advance Dec-POMDP algorithms. For instance, the most influential approximate solution method, MBDP [24], sam-
ples joint beliefs to admit point-based one-step backups. Our methods allow us to perform the one-step backup over the entire joint belief space and thus can find the complete set of useful sub-tree policies obviating the need to pre-specify a ‘max-trees’ parameter. We also plan to investigate whether our techniques may be useful for exact DP methods [11].

8. CONCLUSIONS

In this article we considered multiagent planning under uncertainty formalized as a multiagent POMDP with delayed communication (MPOMDP-DC). A key feature of this model is that it allows a fast response to certain local observations, relevant in time-critical applications such as intelligent power grid control. We showed that the set of legal vectors (corresponding to admissible joint policies) is a subset of the set of vectors for the MPOMDP. Still, because of the way this restriction is specified (as a union over decentralized control laws β), a naive application of incremental pruning (IP) suffers from a significant additional complexity when compared to the MPOMDP case.

In order to address this problem we presented an analysis that shows that the DC backup operator can be represented as a computation tree and presented two methods to exploit this tree structure. The first, TBP-M, is based on the original bottom-up semantics of the computation tree, and gains efficiency via memoization. The second, TBP-BB, broadens regular branch-and-bound methods by reinterpreting the computation tree in a top-down fashion and generalizing the concepts of f, g and h-values to PWLC functions.

We performed an empirical evaluation on a number of benchmark problems that indicates that TBP-M can realize speedups of 3 orders of magnitude over the NAIVE IP baseline. TBP-BB is not competitive with TBP-M on all but one domain (it can not prune enough nodes using its heuristic) but still shows the potential to significantly improve over NAIVE IP in three of six problems. These results show that we have successfully mitigated the additional complexity that the DC backup exhibits over the MPOMDP, allowing for the solution of larger problems.

Acknowledgments

Research supported by AFOSR MURI project #FA9550-09-1-0538 and NWO CATCH project #604.005.003. M. S. is funded by the FP7 Marie Curie Actions Individual Fellowship #275217 (FP7-PEOPLE-2010-IEF).

9. REFERENCES

Strategic behaviour under constrained autonomy

Zinovi Rabinovich
POB 39177, Jerusalem, 91391, Israel
zr@zinovi.net

ABSTRACT
In this paper we investigate the strategic behaviour of a self-preserving assistant robot (SPAR) with constrained autonomy. Specifically, we consider an asymmetric repeated game for two players (the User and the SPAR), where at each stage of the game the User delegates her task to the SPAR. The task has several (uncertain) methods of execution, and the User can either explicitly state a particular method or leave the choice to the SPAR. In addition to this execution asymmetry (the User chooses the method, but the SPAR executes), the game also possesses information asymmetry. Specifically, the SPAR is assumed to know exactly the costs that each execution method will incur at the current stage of the game, while the User can only observe the cost of the chosen method post-execution. In this paper, we concentrate on formalising and generating a behavioural strategy for the SPAR, while assuming that the User can be modelled as an algorithm that solves a multi-armed bandit problem. In particular, we show that the SPAR’s problem can be captured by a particular instance of expected average cost MDPs (EAC-MDP). In addition, we provide an approximate solution for the SPAR’s EAC-MDP and prove some preliminary experimental results of its efficacy.

1. INTRODUCTION
Automated support systems are ubiquitous today. They provide users with music advice, driving instructions, weather forecasts or even bidding in an on-line auctions, generally making life easier and safer. They also allow users virtual access to remote or dangerous working areas. However, automated support systems also become increasingly autonomous, and may have their own interests in performing a task that may conflict with those of system users. In fact, an arms race is unfolding.

On one hand side, researchers seek new ways to improve a user’s experience. For instance, the majority of recommender systems research is directed at maximising the user’s benefit from provided advice (see e.g. [14]). Making sure that the user receives valid information in a virtual market place has also received attention (e.g. [11, 21]). Furthermore, looking after the user’s interests, support systems employ adjustable autonomy. In other words, if the support system is not certain of the impact its action will have on the user, it will signal the user and allow her to take over. In some cases the system will even pro-actively transfer control to the User (see e.g. [15, 16, 10]).

However, while striving to best guide and support a user in her activities, researchers have also developed means to influence the user’s behaviour. In other words, the automated system can provide an advice or behave in such a way that will induce a particular response in the user. A power that has readily been used in advertisement and marketing, where both formal and applied methods (e.g. [2, 13, 8, 6]) have been developed to exploit user’s weaknesses in decision making to “nudge” her towards particular type of actions.

This has led many to take greater care when using a new assistive technology. Users consciously investigate the performance of an automated assistant, trying to assess its worth, and only gradually settle in their habit and degree of reliance on the technology or service.

As an example, consider the following two-player (let’s call them the User and the SPAR) repeated game. At every stage of the game, players are presenting with an opportunity to open one of 4 (four) doors. Opening a door will result in each player incurring some cost, which may be different for each player. The SPAR is allowed to peek behind the doors and knows exactly what costs will be incurred, but can not communicate this information to the User. The User can either order the SPAR to open a specific door (the SPAR is obliged to follow this order), or let the SPAR choose a door to open. Once a door is opened, the stage terminates and the costs of each door for each player are randomly reset. The goal of this game for each players is to minimise her/his expected average cost.

The above situation readily appears in real life, and in some virtual environments as well. For example, a home bound shopper (the User) can instruct her personal shopping assistant (the SPAR) to buy her groceries in a particular shop (open a specific door) or leave the choice to the SPAR. In addition, in RTS games a player (the User) can explicitly instruct her AI companion (the SPAR) to initiate a certain type of attack (open a specific door), or simply ask him to perform an attack by means of his choice. Finally, consider real SPARs that accompany people (e.g. the GM-NASA Robonaut). While explicit instructions given to it have to be executed exactly, when given autonomy the SPAR has to find the least costly way to do so (e.g. with the least amount of energy). Notice that in all of these scenarios the autonomous behaviour of the SPAR has to be benevolent towards the User, and we will later formally define this property.

Now, the strategic issues of the User’s behaviour in this type of interaction have been studied before, and are quite efficiently captured by an adversarial Multi-Armed Bandit (MAB) model, the details of which we will discuss in Section 2. On the other hand, to the best of our knowledge, the SPAR’s point of view on this interaction was never studied formally. Therefore, in this paper, we will be concerned with formalising and generating the SPAR’s strategy. We will start by assuming that the SPAR knows the exact strategy deployed by the User. In fact, we will assume that the User’s be-
haviour follows a known MAB solving algorithm. Furthermore, we will also assume that the SPAR not only knows each door’s cost at the current stage of the game, but also knows how these costs are generated. We will then show that, under the above assumptions, the SPAR essentially faces an instance of Expected Average Cost Markov Decision Problems (EAC-MDPs), which we will formally describe and approximately solve. In addition, we will guarantee that the User’s utility is no worse than it would be if the SPAR did not have an autonomous choice capability. In other words, that the SPAR is benevolent towards the User. It is this benevolence combined with the need to minimise selfish costs, that has led to the term Self-Preserving Assistant Robot – SPAR.

The rest of the paper is constructed as follows. In Section 2 we will formally describe the overall interaction that includes both the User and the SPAR. In particular, we will show why from the User’s point of view this interaction is equivalent to an adversarial multi-armed bandit (MAB) problem. In addition, we will also formulate a generic formalism for the User’s behaviour and instantiate it with two specific MAB solution algorithms. In Section 3 we will formally define the EAC-MDP problem faced by the SPAR, and in Section 4 we will formulate an approximate solution for it. Then Section 5 will describe an experimental evaluation of the approximate SPAR solution, and its preliminary results. Sections 6 and 7 will provide some additional discussion of related works and SPAR’s problem.

2. THE USER’S GAME

To ease the exposition of the formal model of the User’s environment, we will ground it in our game with opening doors. Recall, the User repeatedly decides whether to order the SPAR to open a specific door or allow the SPAR to choose on his own.

From the User’s point of view, this can be modelled as a selection from a set, \( I = \{1, \ldots, K + 1\}^3\), of \( K + 1\) services. In our example \( K = 4\) and services \( 1, \ldots, K\) correspond to ordering the SPAR to open specific door, while the last \( K + 1\) service naturally corresponds to allowing the SPAR an autonomous choice.

We have also mentioned that opening each door will incur costs to the User and the SPAR. While a more general model can be formulated, we will limit our discussion to costs described by a vector of reals \( s = (s_1, \ldots, s_m) \in \mathbb{R}^m\), where \( m = 2K\). In particular, \( s_1, \ldots, s_K\) will describe the cost of each door to the User, and \( s_{K+1}, \ldots, s_{2K}\) will describe the cost of each door to the SPAR. We will also assume that prior to each stage of the game \( s\) is sampled from a multi-variate Gaussian distribution, so that \( s \sim \text{Gauss}(\mu, \Sigma)\), where \( \mu \in \mathbb{R}^m\) and \( \Sigma \) is an \( m \times m\) non-negative definite matrix.

To ensure the necessary information asymmetry between the User and the SPAR, we will require that the User has no prior knowledge of \( \mu \) and \( \Sigma\), although she does know that costs have a Gaussian distribution. On the other hand, SPAR is fully informed of all cost distribution parameters. Furthermore, SPAR is always aware of the exact value of \( s\), since in our example game we allowed it to peek behind the doors. In contrast, the User can only observe which door was opened and what it has cost her.

As a result, the game proceeds as follows:

1. The costs vector is sampled, \( s \sim \text{Gauss}(\mu, \Sigma)\);
2. The SPAR is informed of \( s\);
3. The SPAR calculates \( j \in \{1, \ldots, K\}\), the service index it would use if allowed to choose;
4. The User chooses \( i \in I\), and informs the SPAR of her decision;
5. If \( i \neq K + 1\) (the User gives explicit order to SPAR)
   (a) The service \( i\) in used
   (b) The User incurs cost \( s_i\), while the SPAR incurs \( s_{K+i}\)
   (c) The User is informed if \( j = i\)
otherwise (the User allows SPAR an autonomous choice)
   (a) The service \( j\) in used
   (b) The User incurs cost \( s_j\), while the SPAR incurs \( s_{K+j}\)
   (c) The User is informed that service \( j\) was used

The above game has no pre-determined length, nor discount factor for different stages to create a discounted (or finite) horizon accumulated reward criteria. Rather, we require that both the User and the SPAR minimise their expected average cost per stage. Under the constraints on the information available to the User, the expected average cost translates into regret minimisation, i.e. the User seeks to minimise

\[
\frac{1}{n} \left( \sum_{t=1}^{n} I^{\text{usr}}(i^t, s^t) - \min_{i^t \in I} \sum_{t=1}^{n} I^{\text{usr}}(i^t, s^t) \right),
\]

where \( i^t \in I\) is the index of the service chosen at time \( t\), \( s^t\) is the costs vector at time \( t\), and \( I^{\text{usr}}(\cdot)\) is cost incurred as a result of the User’s choice\(^2\). Notice that this also aligns well with another example given in Section 1, that of the home bound shopper. In that scenario, the User naturally wishes to pay less for her groceries, but still needs to buy them. Hence the need to find the cheapest option (on average) and reduce all deviations from it.

Now, as far the User is concerned, the above setup of the game is actually well known and is termed a Multi-Armed Bandit (MAB). It refers to a particular extended form of a gambling machine with multiple levers (arms). A gambler (or the User, in our case) pulls one of such arms, and each pull generates a reward or incurs cost. Several variations have been studied over the years, including the partial observability variant (the User observes outcomes only of the arm that was actually pulled), and an adversarial MAB, where the arms are assumed to have a mind of their own, just like the SPAR (see e.g. [4]). As a result, as a part of the SPAR’s problem formalisation, it is convenient to refer to the User’s behaviour as being generated by a MAB solving algorithm.

2.1 User’s behaviour model

As we have pointed out in previous section, the User’s behaviour is assumed to be generated by a MAB solving algorithm. Given that the User has no prior knowledge of costs distribution, such algorithms have to be able to accumulate experience (information on incurred costs from each arm) and learn from this experience (change her choices accordingly).

More formally, assume that the User’s memory can hold \( n\) real numbers, and let \( x = (x_1, \ldots, x_n) \in \mathbb{R}^n\) denote the content of her memory. Denote by \( \omega = (\omega_1, \ldots, \omega_{K+1}) \in \Omega = (\mathbb{R} \cup \{\emptyset\})^{K+1}\) denote observations of costs incurred by the User, where \( \omega_i\) corresponds to costs incurred from using service \( i\). If \( \omega_i = \emptyset\) it means that no information on costs of that service is available. In particular, in our door opening game, \( (\omega_1, \omega_2, \omega_3) = (7.3, 0, 0, 0, 7.3)\)

\(^2\)Notice that \( I^{\text{usr}}\) implicitly also depends on the autonomous choice of the SPAR. However, from the User’s point of view it is completely determined by \( s^t\), hence the omission.
would imply that that service 1 (one) was used, and that SPAR
planned to use this service as his autonomous choice. \( (ω_1, ..., ω_i, ω_2) = (0, 5.1, 0, 0, 0) \) implies that the service 2 (two) was used, and the SPAR planned on a different service as its autonomous choice.

Then the following functions fully determine the User’s behaviour and how it changes over time:

- \( α : ℜ^n → Δ(1) \) describes the manner in which the service is selected given the User’s current memory content. Hence, \( α(x, i) \) is the probability of the User to select service \( i ∈ I \) if its memory is currently holds \( x ∈ ℜ^n \).

- \( ∂ : ℜ^n × I × Ω → ℜ^n \) that describes how the memory content changes. In particular, \( ∂(x, i, ω) \) is the memory content formed from the previous memory \( x ∈ ℜ^n \), and costs information \( ω \) that was received after choosing service \( i ∈ I \).

Notice that the observation \( ω \) received by the User depends on costs vector \( s ∈ ℜ^K \), the choice the SPAR would have made autonomously, and the User’s service selection. Let \( s : ℜ^n × [1 : K] × I → Ω \) describe this dependency, so that \( s(o, s, j) \) is the observation of costs the User receives if she chooses to use service \( i ∈ I \), when the costs vector is \( s ∈ ℜ^K \) and the SPAR would have chosen the service \( j ∈ [1 : K] \). Then, the User’s memory update can be described by a function of the form \( σ : ℜ^n × ℜ^n × [1 : K] → ℜ^n \), defined by

\[
σ(x, i, s, j) = ∂(x, i, s, j, i).
\]

It is important to point out that, since the SPAR has more information that the User, knowledge of \( α \) and \( σ \) would allow the SPAR to determine exactly both the content of the User’s memory at any given time and the probability of User’s service choices. Furthermore, analysis of \( α \) and \( σ \) makes it possible to predict how the User will behave if the SPAR was never allowed to make an autonomous decision (or simply had no such option). In particular, we will denote User’s memory update without SPAR’s service by \( σ(x, i, s, ∅) \).

### 2.2 User solution algorithms

Although a wide variety of algorithms have been developed to solve MABs, in this paper we will concentrate on just two: \( ϵ \)-greedy and SoftMax. While there are more advanced algorithms with strong theoretical guarantees on performance (see [4] for detailed discussion), these two simpler algorithms have been shown to work well in practice (see e.g. [9]). However, the detailed discussion of the relative strength of these algorithms is outside the scope of this paper, and we will only summarise these algorithms in terms of their representation by \( α \) and \( σ \) functions.

Both for the \( ϵ \)-greedy and the SoftMax algorithms, the memory holds the average of the accumulated losses thus far. In our game that would imply that it has the size \( n = 2(K + 1) \), and that \( x \) has the form of \( (x_1, ..., x_{2(K+1)}) = (\hat{l}_1, ..., \hat{l}_{K+1}, 1 \frac{n_1}{n_1}, ..., 1 \frac{n_{K+1}}{n_{K+1}}) \), where \( \hat{l}_i \) is the accumulated cost from using service \( i ∈ I \), and \( n_i \) is the number of times that service was used. The initial experience is commonly set so that \( \hat{l}_i = 0 \) and \( \frac{n_i}{n} = 1 \). Let \( ω \) be the observation received by the User, then \( x' = ∂(x, i, ω) \) is defined by

\[
\hat{l}_i = \begin{cases} \hat{l}_i + ω_i = 0 & \text{otherwise} \\ \hat{l}_i + ω_i & \text{otherwise} \end{cases} \quad \text{and} \quad n_i' = \begin{cases} n_i + 1 & \text{otherwise} \\ n_i \end{cases}.
\]

In turn, \( σ \) is obtained by substituting the observation function \( o(s, j, i) \) as before.

However, the action selection of \( ϵ \)-greedy and SoftMax differ. \( ϵ \)-greedy almost always follows the service with the best experience summary thus far, but with small probability \( ϵ \) chooses the service at random.

\[
α^{−greedy}(i|x) = (1 - ϵ)δ \left( i, \arg\min_k \frac{\hat{l}_k}{n_k} \right) + \frac{1}{K + 1},
\]

where \( δ(·, ·) \) is the Kronecker delta function. The SoftMax algorithm on the other hand, continues to explore the quality of different services much more aggressively.

\[
α^{SoftMax}(i|x) ∝ \exp(-\frac{\hat{l}_i}{n_i}),
\]

where \( τ \) is a parameter that expresses the importance of the cost difference between services.

### 3. SPAR’S DECISION PROBLEM

In this section, we will describe how the User’s behaviour description \( < α, σ > \) can be incorporated into an overall interaction model from the SPAR’s point of view. More specifically, an Expected Average Cost Markov Decision Problem (EAC-MDP). The state space of this EAC-MDP will reflect the memory content of the User and the current service costs vector, while the EAC-MDP transition function will capture how these vector values change over time. In turn, EAC-MDP’s utility function will describe the costs incurred by the SPAR. Finally, we will consider SPAR’s behaviour as a decision policy of this EAC-MDP.

Formally, given the User’s behaviour description \( < α, σ > \), the SPAR can formulate its interaction with the User as a Markov Decision Process, \( < SX, A, T, u, > \), defined as follows:

- \( SX = ℜ^n × ℜ^n \) is the state space of the process. \( (s, x) ∈ SX \) denotes the fact that the costs vector is currently \( s ∈ ℜ^n \) and the User’s memory content is \( x ∈ ℜ^n \).

- \( A = [1 : K] \) is the set of action available to the SPAR. In effect, this is the set of possible autonomous decisions made by SPAR.

- \( T : SX × A → Δ(SX) \) is the transition probability of the process. \( T((s', x)|(s, x), a) \) is the probability that new costs vector will become \( s' \) and the User’s memory will update from \( x \) to \( x' \), given that SPAR would have selected service \( a \) if allowed an autonomous choice. Denoting by \( p(·) \) the probability density of the Gaussian distribution \( Gauss(μ, Σ) \), we can formally define \( T \) by

\[
T((s', x)|(s, x), a) = p(s') \sum_{i ∈ I} \delta(x', σ(x, i, s, a))α(i|x)
\]

- \( u : SX × A → R \) is the utility function of the SPAR. \( u((s, x), a) \) describes the utility of an autonomous choice of service \( a \) while the costs vector was \( s \) and the User’s memory content was \( x \). Formally,

\[
α^{−greedy}(i|x) = (1 - ϵ)δ \left( i, \arg\min_k \frac{\hat{l}_k}{n_k} \right) + \frac{1}{K + 1},
\]

where the \( l^{SPAR} \) describes costs suffered by SPAR when the User invoked a particular service. Formally,

\[
l^{SPAR}(s, i, a) = \begin{cases} s_{i+1} & i ≠ K + 1 \\ s_{i+1} + a & i = K + 1 \end{cases}
\]

In this paper, we will limit our attention to time invariant Markovian policies of the form \( π : SX → Δ(A) \), and adopt the Expected

\[
T((s', x)|(s, x), a) = \sum_{i ∈ I} \delta(x', σ(x, i, s, a))α(i|x)
\]

\[
l^{SPAR}(s, i, a) = \begin{cases} s_{i+1} & i ≠ K + 1 \\ s_{i+1} + a & i = K + 1 \end{cases}
\]
Average Cost (EAC) criterion to define an optimal policy $\pi^*$. Formally, SPAR seeks to minimise the expression

$$L_{SPAR}^{\pi} = \lim_{T \to \infty} \frac{1}{E} \sum_{t=1}^{T} u(s^t, x^t, a^t),$$

where the superscript $t$ denotes that the state and action have occurred during $t$'s service request from the User. If the SPAR was fully selfish, a solution to this minimisation would constitute an optimal SPAR behaviour. However, part of SPAR’s definition was its benevolence towards the User, that is SPAR’s autonomous choice capabilities need to benefit the User. To formally define this limitation, we need an additional assumption regarding the User’s behaviour.

Specifically, we assume that for any $\pi: SX \to \Delta(A)$, the User’s memory converges to a unique limit (that depends on $\pi$). Essentially it means that there is a best response distribution $\alpha^* \in \Delta(I)$, that can be discovered by the User in response to $\pi$ and lead to memory stabilisation in long term. This is a fairly strong stabilisation assumption, and may not hold in general for mutually adapting strategies (see e.g. [20]), however we defer the detailed treatment of this issue and its relaxation to future work.

Formally, let a sequence $\{x^t\}_{t=1}^{\infty}$ be generated from $x_1$ (the initial content of the User’s memory) by repeated iteration of the following steps:

1. Sample $s \sim \text{Gauss}(\mu, \Sigma)$
2. Sample $a \sim \pi(\cdot|s, x^t)$
3. Sample $i \sim \alpha(x^t)$
4. Calculate $x^{t+1} = \sigma(x^t, i, s, a)$

We assume that $x^*_s$ exists, so that for any $\{x^t\}_{t=1}^{\infty}$ as above, holds $x^*_s = \lim_{t \to \infty} x^t$. Similarly, we assume existence of a limit point $x^*_s$, when the sequence $\{x^t\}$ is generated using $\sigma(x^t, \alpha, s, \emptyset)$, that is the User has no SPAR service. Notice that this latter assumption is a much simpler form of stabilisation, and simply means that the User can find the service with best expected cost and stick with it.

As a consequence of the above User stabilisation assumptions, we can calculate the expected stable cost to the User with ($L_{usr}^{\pi}$) and without ($L_{usr}^{\emptyset}$) the SPAR’s autonomous choice capabilities, as follows.

$$L_{usr}^{\emptyset} = \int_x \left( \sum_{s \in [1:K]} \alpha(s|x^{*}_s)s_i \right) p(s)ds$$

$$L_{usr}^{\pi} = \int_x \left( \sum_{s \in [1:K]} \alpha(s|x^{*}_s)s_i + \alpha(K+1|x^{*}_s) \sum_{a \in [1:K]} \pi(a|s, x^{*}_s)s_a \right) p(s)ds$$

Given the expected stable cost to the User, the benevolence of the SPAR is expressed by enforcing the inequality $L_{usr}^{\pi} \leq L_{usr}^{\emptyset}$. In other words, the User is never better off by completely disregarding SPAR’s autonomous decision capabilities. We now can define the optimal policy for the SPAR as a solution to the following optimisation:

$$\pi^* = \arg\min_{\pi: SX \to \Delta(A)} L_{SPAR}^{\pi}$$

s.t.

$$L_{usr}^{\pi} \leq L_{usr}^{\emptyset}$$

In words, the SPAR looks for a policy $\pi^*$ that maps current costs vector $s \in \mathbb{R}^m$ and the current content of the User’s memory $x \in \mathbb{R}^a$ into a distribution over actions, $\pi^*(\cdot|(s, x))$, and has the following two properties: a) The User’s expected cost (in the stable memory state) is not increased in comparison with a SPAR without autonomous decision capabilities; b) Given the above limitation, the SPAR’s expected (average) cost is minimised.

## 4. APPROXIMATE SOLUTION FOR SPAR

Although SPAR’s behaviour design problem is now well defined, it necessitates solution of an EAC-MDP with a high-dimensionality continuous state and non-trivial dynamics, which makes an exact optimal solution computationally hard. Now, theoretically there is a wide spectrum of kernel based methods that can yield an approximate solution via value function approximation or direct policy approximation. However, these methods are computationally complex, and their tuning (e.g. choice of the kernel base and associated hyper-parameters) can not be performed efficiently if the basic properties of the problem solution are poorly understood.

Rather than compounding the problem with complex approximation techniques, we choose to view SPAR’s problem from the control theory perspective. This enables us to use standard approximation techniques, which are characteristic of control methodology [17] and are particularly suitable for model based controller design, as is the case in our domain.

Now, from the control theoretic point the SPAR’s policy is a closed loop controller applied to a (partially) controlled dynamic system (the User). It is also convenient to view the service costs vector as the Gaussian noise that drives the system, rather than a portion of the system state. Furthermore, the assumption that the memory content of the User converges for any fixed SPAR policy means that the User-SPAR autonomic pair is self-stabilising. In the situation the common control theoretic approach is to perform the following steps:

1. Find system behaviour at the stable state. In our case this would mean estimating the expected SPAR and User costs once the User’s memory has converged.
2. Approximate system stable state behaviour as a function of policy features. In other words, determine how the SPAR and User costs would change if we were to modify some property of the SPAR policy.
3. Solve the inverse kinematics problem. That is find the stable system state that minimises the costs and calculate the policy that yields it. In our domain, this means finding the User’s memory content and the SPAR policy that supports it with minimal costs to the SPAR.
4. Apply model-following error correcting controller based on the idealised policy and system behaviour found above. Specifically, given the current disturbance (services costs vector, in our domain) and the current system state (User’s memory content), find the control signal (SPAR autonomous choice) that produces a similar output (combined User-SPAR service selection) to the system in the ideal stable state.

The first three steps of the above scheme are approximations of the complete system behaviour, and are commonly represented by the Extended Kalman Filter and the Unscented Kalman Filter. The final stage is frequently implemented heuristically, and we follow this trend as well.
Now, the details of the outlined scheme applied to our EAC-MDP are as follows. We will begin by inspecting the long term utility of the SPAR, 

\[ L_{\text{SPAR}}[\pi] = \lim_{T \to \infty} \frac{1}{T} \mathbb{E} \sum_{t=1}^{T} u((s^t, x^t), a^t). \]

**First Stage.**

Since the User’s memory content converges to \( x^* \), it holds that \( u((s^t, x^t), a^t) \to u((s^t, x^*_t), a^t) \). Second, under some mild assumptions of \( \pi \)'s continuity, we also have that \( \pi(a^t| (s^t, x^t)) \to \pi(a^t| (s^t, x^*_t)) \). This allows us to simplify \( L_{\text{SPAR}}[\pi] \) expression significantly. In fact, by substituting an explicit expression for \( u((s, x), a) \), we obtain an expression for \( L_{\text{SPAR}}[\pi] \) that has the same structure as that of \( L^{\text{usr}} \):

\[ L_{\text{SPAR}}[\pi] = \int x \left( \sum_{i \in [1:K]} \sum_{a \in [1:K]} \alpha(i|x^*_a)s_{K+i} \right) \alpha(K+1|x^*_a) p(s)ds \]

**Second Stage.**

Notice that \( L_{\text{SPAR}}[\pi], L^{\text{usr}}[\pi] \) and \( L^{\text{usr}}[\emptyset] \) are expectations over a Gaussian. We therefore use the Unscented Transform [7] to devise a small set of characteristic features for a policy \( \pi \), and formulate the expected cost as a function of those features. In addition, by applying the mechanism of the unscented filter [7], we further simplify the expressions for \( L_{\text{SPAR}} \), while maintaining a reasonable degree of approximation of the User-SPAR system behaviour. In more details, let \( s^1, ..., s^{2K} \) be the sigma points of the services costs distribution \( \text{Gauss}(\mu, \Sigma) \), and \( w^1, ..., w^{2K} \) their weights.

Then the unscented approximations \( L^{\text{SPAR}}_{\pi} \) of \( L^{\text{SPAR}}[\pi] \) (and similarly for \( L^{\text{usr}}[\pi] \) and \( L^{\text{usr}}[\emptyset] \)) can be written as:

\[ L^{\text{SPAR}}_{\pi} = \sum_{j=1}^{2K} w^j \left( \sum_{i \in [1:K]} \alpha(i|x^*_a)s_{K+i} \right) \alpha(K+1|x^*_a) \]

Finally, recall that \( x^*_x \) is a convergence point of all sequences of User’s memory content \( \{x^t\}_t \) generated by the User’s memory update function \( \sigma \). In essence, \( \{x^t\}_t \) is a random walk generated by Gaussian input noise (service vector), since all other parameters are determined by it. As a result all expected memory contents \( E(x^t) \) also converge to \( x^*_x \). Furthermore, we can assume that \( x^* \) is in fact a set point (in expectation) of the User’s memory update function \( \sigma \), under some mild continuity assumptions on \( \sigma \). Therefore, we can apply the same unscented transformation to \( x^*_x \), in other words seek the convergence point of memory content sequences \( \{x^t\}_t \) that were generated only for the set of sigma points \( \{s^i\}_i \). We will denote this approximate stable (User’s memory) state by \( x^*_S \).

Notice that for such algorithms as SoftMax and \( \epsilon \)-greedy, the above holds exactly and \( x^*_S = x^*_S \).

**Third Stage.**

The result of the approximations steps above is the following approximation formula \( L^{\text{SPAR}}_{\pi} \) for \( L^{\text{SPAR}}[\pi] \) (and similarly for \( L^{\text{usr}}[\pi] \) and \( L^{\text{usr}}[\emptyset] \)):

\[ L^{\text{SPAR}}_{\pi} = \sum_{j=1}^{2K} w^j \left( \sum_{i \in [1:K]} \alpha(i|x^*_a)s_{K+i} \right) \alpha(K+1|x^*_a) \]

Notice that \( L^{\text{SPAR}}_{\pi} \) only depends on the values of the policy \( \pi \) at points of the form \( \{s^i, x^*_S\}_i \). Furthermore these are the only values of \( \pi \) that are of interest for the calculation of \( x^*_S \) itself. In other words, knowing the SPAR’s behaviour in response to service costs vector \( s^j \) is sufficient to determine the User’s and the SPAR’s costs in the stable (User’s memory) state. Denote by \( \pi^*_S \) the policy \( \pi \) limited to the set of arguments \( \{(s^j, x^*_S)\}_j \). Then \( \pi^*_S \) can be found by standard methods from the following optimisation problem:

\[ \pi^*_S = \arg \min_{\pi \in \pi^*_S} L^{\text{SPAR}}_{\pi} \quad \text{s.t.} \quad L^{\text{usr}}[\pi] \leq L^{\text{usr}}[\emptyset] \]

**Fourth Stage.**

At this stage, a model-following controller would be applied based on \( \pi^* \). However, to do so it is necessary to extend \( \pi^*_S \) to the entire domain \( SX \). We do so heuristically, and the overall controller performs the following calculation for a given \( (s, x) \in SX \):

- Calculate \( j^* = \arg \min_{j \in [1:2K]} ||s - s^j|| \)
- Calculate \( \forall a \in [1 : K], q(a) = \alpha(a|x^*_S) + \alpha(K+1|x^*_S) \pi^*_S(a'(s^j, x^*_S)) \)
- Calculate \( \forall a, a' \in [1 : K], q(a, a') = \alpha(a'|x) + \alpha(K+1|x) \delta(a, a'), \) where \( \delta(\cdot, \cdot) \) is the Kronecker delta function.
- Find \( a^* = \arg \min_{a \in [1:K]} ||q(a) - q|| \)
- Set \( \pi^*(s^j, x) = \delta(a, a^*) \)

We term the heuristic used in the controller’s construction the \( \sigma \)-heuristic, and the overall policy produced by the above four stages the \( \sigma \)-policy.

5. EXPERIMENTS

In our experiment we have re-created the 4-doors game scenario. Recall, the User gets to pick which door the SPAR will open, or allow the SPAR to choose. Opening a door generates two distinct costs, one for the User and one for the SPAR. Once a door is opened, the costs are reset by sampling them from a multi-variate Gaussian, and the User and the SPAR repeat the decision cycle. In our case, the Gaussian is 8 (eight) dimensional, so that first four number describe costs for the User and the second four number describe the costs for the SPAR. To simplify our initial study we set the Gaussian dimensions to be independent from each other, so that the entire distribution can be described by 16 numbers (8 values for the means, and 8 values for the variances of the Gaussian). Different setting of the costs distribution parameters, 4 (four) in total, are depicted in Table 1. These setups present the User and the SPAR with different variations of conflicting preferences for different doors. For example, in costs Setup No1, their expected costs are polarly different, i.e. the better the door for the User the worse
it is for the SPAR and vice versa. On the other hand, Setup No2 presents a much more mild dilemma, where only the best and the second best door (on average) are swapped

As a performance base-line we used two other methods to generates the SPAR’s door-opening policy. Specifically, we used the selfish policy, where the SPAR always chooses the door with smallest cost to itself, disregarding any effect it may have on the User; and User-best policy, where the SPAR always chooses the door that is cheapest for the User, completely disregarding its own costs. Notice that the selfish policy actually breaks the benevolence condition imposed on a SPAR. That is the selfish policy decisions have the potential to increase the cost incurred by the User. On the other hand, the User-best policy is a trivial solution that satisfies the benevolence condition, but does not optimise SPAR’s cost.

For each combination of the SPAR policy and User algorithm we ran 75 interaction sequences of 600 steps each. Where drawn, bars represent 0.95% estimate confidence.

Table 1: Costs distributions settings

<table>
<thead>
<tr>
<th>Setup</th>
<th>User Costs</th>
<th>SPAR Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>No1 (µ)</td>
<td>5 10 15 25 25 15 10 5</td>
<td>9 9 9 9 9 9 9 9</td>
</tr>
<tr>
<td>No1 (diag Σ)</td>
<td>9 9 9 9 9 9 9 9</td>
<td>9 9 9 9 9 9 9 9</td>
</tr>
<tr>
<td>No2 (µ)</td>
<td>5 10 15 25 10 5 15 25</td>
<td>9 9 9 9 9 9 9 9</td>
</tr>
<tr>
<td>No2 (diag Σ)</td>
<td>9 9 9 9 9 9 9 9</td>
<td>9 9 9 9 9 9 9 9</td>
</tr>
<tr>
<td>No3 (µ)</td>
<td>5 10 15 25 15 10 5 25</td>
<td>9 9 9 9 9 9 9 9</td>
</tr>
<tr>
<td>No3 (diag Σ)</td>
<td>9 9 9 9 9 9 9 9</td>
<td>9 9 9 9 9 9 9 9</td>
</tr>
<tr>
<td>No4 (µ)</td>
<td>5 10 15 25 5 10 15 25</td>
<td>9 9 9 9 9 9 9 9</td>
</tr>
<tr>
<td>No4 (diag Σ)</td>
<td>9 9 9 9 9 9 9 9</td>
<td>9 9 9 9 9 9 9 9</td>
</tr>
</tbody>
</table>

Figure 1: Average proportion of strict commands and autonomous SPAR (σ-policy) decisions vs e-greedy User. Costs Setup No4.

Our heuristic solution has successfully operated versus the e-greedy User. Figure 3 shows smoothed mean (across interactions sequences) of the SPAR’s cost. Graphs show that the selfish SPAR inevitably pays greater costs, than the User-best or the sigmapolicy SPARs, however the latter two run at a similar performance level. Essentially, the e-greedy User was always able to detect the selfish SPAR, and learn to avoid its advice. On the other hand, sigma-policy SPAR was allowed to make an autonomous decision quite often, as can be seen from Figure 1 that depicts User choice frequencies in costs Setup No4.

However, if the User’s policy tends to explore and continue to explore the quality of different services, it becomes much more susceptible to exploitation by the selfish SPAR policy. While the exploration rate of the e-greedy very quickly drops and remains small (e small, as a matter of fact), the exploration rate of the SoftMax User depends on the relative difference between the qualities of services. Unless SoftMax parameter τ is increased over time or initially chosen to be very large, SoftMax will continue to switch between different services, as is demonstrated in Figure 2. As a result the selfish SPAR policy aggregated the smallest cost among the three SPAR policies vs the SoftMax User with fixed τ in all costs Setups (see Figure 3). The User continued to allow the selfish SPAR to make an autonomous decision, in spite of its negative impact on the User’s cost.

If, on the other hand, we require that the benevolence of the SPAR towards User is to be maintained, the σ-policy becomes the better choice. In most of our costs Setups the σ-policy SPAR significantly outperformed the User-best SPAR policy. In other words, while benefiting the User, the σ-policy SPAR managed to efficiently control and reduce its own costs. In fact, in several cost Setups the discrepancy in performance of the σ-policy and the selfish policy diminished over time. For instance in Setup No 1, the cost aggregated by the selfish policy grew approaching the levels of the σ-policy. At the same time, in the cost Setup No 2 the performance of the σ-policy approached that of the selfish policy.

6. RELATED WORK

The Multi-armed bandits (MABs) have been extensively used before as the core of an interaction environment model in general, and for guidance and assistance systems in particular. In the latter case, the paradigm of environment design or teaching is more commonly adopted (see e.g. [19, 12]). Specifically, the User is assigned the role of MAB’s arm selection, while the guidance/assistance system manipulates the response of different arms. For example, Chen et al [5] investigate the possibility of off-setting the cost of each arm to induce a particular pattern in the User’s arm choices. However, the possibility to employ incentives is not universally available in all situations, and other researchers have preferred User guidance by demonstration. For instance, Stone et al [18, 3] investigate environments where both the User and assistance system have to pull a MAB’s arm, thus forming an ad-hoc team. Allowing for even less involvement, [1] investigate the case where the system merely suggests a choice to the User. Notice, however, that none of these models can be directly applied to the interaction characteristic of
our 4-door game example, since they do not address the User’s delegation of the arm choice to the assistance system, where the system first performs an action and only reports on it. Interestingly enough, the opposite situation has been investigated. For example, Sofman et al. [16] discuss a model where the human User is viewed as an arm in a MABs and it is the system that chooses an arm. In their paper, the model captures the situation where the assistant system needs to decide when (if at all) to relinquish its autonomy and delegate the task back to the User.

7. CONCLUSIONS

In this paper we have introduced a new type of interaction between an adaptive User and a self-preserving assistant robot (SPAR). The interaction allows the SPAR to have action costs different from those of the User, but limits the SPAR’s autonomy. In more detail, the User chooses between explicitly instructing the SPAR to perform a specific action, or allows the SPAR to make the decision autonomously. Due to SPAR’s desire to reduce its own costs, during autonomous choices it can choose an action sub-optimal from the User’s point of view. This, however, has to be balanced with gaining User’s trust. Otherwise the User will never allow the SPAR to make the choice autonomously. To achieve this balance, we fully formalise the problem by: a) modelling the User as a Multi-Armed Bandit solution algorithm; b) modelling the SPAR’s decision problem as an expected average cost Markov decision problem; c) we explicitly limit the SPAR to benevolent solutions, that guarantee not to increase the User’s costs.

In addition, this paper provides an approximation algorithm for the SPAR’s MDP problem, based on unscented filter method and a heuristic stabilisation meta policy – $\sigma$-heuristic. We experimentally show that the approximation performs correctly with respect to benevolence towards the User, and reduces SPAR’s costs where possible under this constraint.

As a future work we plan to investigate performance of $\sigma$-heuristic against models of human behaviour, as well attempt an analytical solution of SPAR’s EAC-MDP.

8. REFERENCES


Figure 4: Cost per step vs. SoftMax User.

Prioritized Shaping of Models for Solving DEC-POMDPs

Pradeep Varakantham†, William Yeoh†, Prasanna Velagapudi‡, Katia Sycara‡, Paul Scerri‡
†Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
‡School of Information Systems, Singapore Management University, Singapore 178902

ABSTRACT

An interesting class of multi-agent POMDP planning problems can be solved by having agents iteratively solve individual POMDPs, find interactions with other individual policies, shape their transition and reward functions to encourage good interactions and discourage bad ones then recompute a new policy. The D-TREMOR algorithm showed that this approach can allow distributed planning for hundreds of agents. However, the speed and quality of the planning process depends on exactly how the models are initialized and shaped with respect to possible interactions. In this paper, we first look at how the planning process is impacted depending on whether the agents initially assume interactions will or will not happen then look at how different prioritization policies, i.e., how the models are shaped when there is a negative interaction between agents, impact convergence. We prove the convergence behavior but empirically show that planning time and plan quality are only sometimes improved.

1. INTRODUCTION

Cooperative multi-agent and multi-robot teams in domains such as sensor networks and disaster rescue [5, 10] require that agents plan courses of action that achieve their joint objectives. In complex domains, where agents are faced with many options, uncertainty and risk, finding good plans can be computationally extremely difficult. For example, if a heterogeneous team of robots is attempting to rescue victims from a burning building, the robots must cooperatively plan to clear paths to the victims, rescue victims and avoid interfering with each other despite considerable uncertainty about their ability to move in different areas and about the state of the environment. An interesting class of multi-agent POMDP planning problems can be solved by having agents iteratively solve individual POMDPs, find interactions with other individual policies, shape their transition and reward functions to encourage good interactions and discourage bad ones then recompute a new policy. One such algorithm, Distributed - Team’s Re-shaping of Models for Rapid Execution (D-TREMOR), has been shown to efficiently compute POMDP policies for more than 100 agents, an order of magnitude scale up over most centralized, joint POMDP planners[9].

However, the speed and quality of the D-TREMOR planning process depends on exactly how the models are initialized and shaped with respect to possible interactions. In this paper, we look at two aspects of this: (a) the priority ordering of agents when they shape their models to improve interactions and (b) the prior beliefs of the agents with respect to whether an interaction will happen. Previous work with other types of algorithm has shown the promise of such directions[8, 2, 3]. For example, prioritization is at the heart of very fast path-planning algorithms for large robot teams[11]. However, to the best of our knowledge these ideas have not been applied to distributed POMDPs.

Prioritization has been successfully used as a multi-agent planning technique in both centralized and decentralized planners. The intuitive idea is to give order to the agents and make lower priority agents plan around the plans of the higher priority agents. In decentralized prioritized planning, the agents can plan simultaneously with conflicts in the plans resolved in favor of the higher priority agents. One prioritization heuristic shown to work well is prioritize agents that are more valuable to the team. We have applied this same concept to D-TREMOR, creating an algorithm called PD-TREMOR. Specifically, priorities are dynamically set based on the expected utility the agent computes for its local policy. Although these values change at each iteration, one agent’s priority is fixed ensuring convergence. We prove that using this prioritization scheme the algorithm is guaranteed to converge, unlike D-TREMOR.

Prioritized path planners have usually initially assumed that agents will not collide. This optimism results in good plans because the agents try the best options first only reverting to longer paths if higher priority agents interfere. However, in PD-TREMOR there are both positive and negative interactions making choosing the initial assumptions more difficult. If positive interactions such as one agent changing the environment to help another are initially assumed not to happen, performance can be poor, likewise if negative interactions, such as interference, are assumed to occur initial plans can be unnecessarily poor. Instead, we take advantage of a POMDPs ability to use probabilities and have experimented with intermediate values for prior beliefs about whether an interaction will occur. So an agent might initially assume that some interaction is going to occur with probability 0.25 and plan for that case. Notice that the iterations of the algorithm and the dynamic prioritization are responsible for ensuring convergence, the priors simply aim to start the search in a better position.

We performed a detailed empirical evaluation of PD-
TREMOR, using the domain presented in [9]. Surprisingly, the results were very mixed, with PD-TREMOR being outperformed by D-TREMOR in several cases. On small problem instances, PD-TREMOR consistently and significantly outperformed D-TREMOR. Moreover, for negative interactions, PD-TREMOR worked better, performing at least as well as D-TREMOR and often better. However, for positive interactions, where agents gain by interacting, the prioritization failed to help. Generally, it appears that when the density of interactions are low and there are positive interactions prioritized planning techniques do not work. This is surprising, since prioritization techniques that we applied have been successfully used with other algorithms. The reason for this is not yet clear. On statistically very similar problem instances, the difference between the algorithms varied greatly, hinting that some subtle effect is at play.

3. ILLUSTRATIVE PROBLEM DOMAIN

We employ an illustrative disaster rescue problem similar to the one introduced in [10]. Figure 1 shows the example. In this problem, a team of heterogeneous robots need to save victims trapped in a building with debris. We model the building as a discrete grid with narrow corridors connecting some neighboring grid cells and debris in some grid cells. There are two types of robots available: (a) rescue robots that save victims by providing them medical attention; and (b) cleaner robots that remove debris from building corridors to allow easy passage for rescue robots. Narrow corridors allow for only one robot to pass through; when multiple robots try to pass through, a collision (modeled with negative rewards) occurs. Cells containing debris allow rescue robots to pass through with low probability and cleaner robots to pass through with certainty. Furthermore, the debris is removed from the cell when a cleaner robot pass through it. All robots must reason about uncertainty in their actual positions and slippages (action failures) when moving to locations. The goal of the robots is to save as many victims as possible within the time available.

This problem has a rich environment of conflicting positive and negative interactions and situations where modeling uncertainty is critical to team performance, making it a challenging problem to test decision-making. However, the simplification of modeling collisions with negative rewards means that when these rewards are sufficiently large enough to impact policies, it is sometimes possible for policies that avoid risk to achieve higher values than policies that successfully rescue many victims, leading to unintuitive rankings of solutions.

3. BACKGROUND AND PRELIMINARIES

In this section, we briefly describe the DPCL model, the TREMOR algorithm, and the D-TREMOR algorithm.

3.1 DPCL

We employ the Distributed POMDPs with Coordination Locales (DPCL) model [9] to represent the problems of interest in this paper. DPCL is similar to the DEC-POMDP model in that they are both represented by the tuple of \((S, A, P, R, \Omega, \emptyset)\), where \(S, A, \Omega\) are the joint states, actions and observations, respectively, and \(P, R, \Omega\) are the joint transition, reward and observation functions, respectively. The primary difference between DPCLs and DEC-POMDPs is that the interactions between agents in DPCL are limited to coordination localities (CLs). CLs represent situations where the actions of one agent affect the local transition and reward functions of other agents. There are two kinds of CLs: positive and neutral CLs. Intuitively, positive CLs are CLs where the effects result in a positive gain in joint rewards. This includes CLs where the effect can be positive for one agent but negative for another agent as long as the net gain in joint rewards is positive. Using our disaster rescue problem, if the shortest route to a victim is blocked by a cell with debris, then that cell is a positive CL between a cleaner robot and a rescue robot; the cleaner robot can first pass through that cell so that the rescue robot can pass through the same cell with higher probability to save the victim. Conversely, negative CLs are CLs where the effects result in a negative gain in joint rewards. Using the same example problem, if two robots attempt to enter the same narrow corridor simultaneously, they would collide and incur a negative reward. Although there can also be neutral CLs, that is, CLs where the effects result in no gain in joint rewards, we ignore those in this paper.

Formally, a CL is defined as the tuple \((t, \{(s_i, a_i)\})\), where \(t\) is the decision epoch, \(s_i\) is the local state of agent \(i\) and \(a_i\) is the action taken by agent \(i\). The set of CLs is computed from the joint transition and reward functions. Figure 2(a) shows an example problem, which we will use to illustrate CLs. We will also use this example as a running example throughout this paper. The problem consists of a 3x3 grid with blocked cells, which are colored black, and unblocked cells, which are colored white. There are two agents, rescue robots \(R1\) and \(R2\), that are each assigned to rescue victims \(V1\) and \(V2\), respectively. The robots can either stay stationary or move deterministically to one of their neighboring unblocked cells. The robots collide with each other if they try to enter the same cell. The robots have only 4 time steps to rescue the victims and they receive a reward of 5 for rescuing victim \(V1\) and a reward of 10 for rescuing victim \(V2\).

Figures 2(b) and 2(c) show the model of robots \(R1\) and \(R2\), respectively, which they use to compute their policy to get to their respective goals. The policy of robot \(R1\) is the following trajectory \(A1, A2, B2, C2, C1\), and the policy of robot \(R2\) is \(A3, A2, B2, C2, C3\). Thus, they would collide in cells \(A2\) in time step \(t_1\), \(B2\) in \(t_2\) and \(C2\) in \(t_3\). Therefore, there are three negative CLs – \(\langle t_1, \{(A1, down), (A3, up)\}\rangle\), \(\langle t_2, \{(A2, right), (A2, right)\}\rangle\), and \(\langle t_3, \{(B2, right), (B2, right)\}\rangle\).

3.2 TREMOR and D-TREMOR
We now describe the Teams REshaping of MOdels for Rapid execution (TREMOR) algorithm [9] and its extension Distributed TREMOR (D-TREMOR) [12]. The goal in TREMOR is to find an optimal task allocation, and provide a policy for each agent to accomplish its tasks. TREMOR performs an approximate branch-and-bound search over the set of all task allocations using MDP-based heuristics. Algorithm 1 shows the pseudocode.

TREMOR first computes the policies for individual agents assuming no other agents exist in the environment (line 1). Given the policies, the probability of occurrence of a CL is determined by propagating beliefs for the individual POMDPs and only the active ones are considered for the next stage of the algorithm (line 5). A CL is active if it has a probability of occurrence that is greater than $\epsilon$. For each active CL, TREMOR evaluates the effect of the CL, that is, the change in the valuation of the agents’ current joint policy $\pi$, and uses these valuations along with the probability of occurrence of the CL to shape the POMDP models for the individual agents (lines 6-8). The algorithm then computes policies for the individual agents with the shaped models (line 10) and repeats these steps until convergence or for a maximum number of iterations (line 4).

TREMOR shapes the models in two steps: (a) it modifies the individual transition and reward functions in such a way that the joint policy evaluation is equal (or nearly equal) to the sum of individual policy evaluations; and (b) it introduces incentives or hindrances in the individual agent models based on whether a CL accrues positive or negative reward to the team members.

We use our running example to illustrate the shaping of an agent model. Recall that there are three negative CLs – $\{t_1, \{(A1, down), (A3, up)\}\}$, $\{t_2, \{(A2, right), (A2, right)\}\}$ and $\{t_3, \{(B2, right), (B2, right)\}\}$. TREMOR can shape the agent model of robot $R2$ to avoid the CLs, resulting in the shaped models in Figures 2(d), 2(e) and 2(f). As a result, the new policy of robot $R1$ is its initial policy and the new policy of robot $R2$ is to stay stationary since it cannot get to its victim within 4 time steps. Thus, the joint reward of the team is 5 (for rescuing victim $V1$).

By starting from individual POMDPs and incrementally modifying the model to accommodate most likely CLs, TREMOR was able to scale to problems that were not feasible with earlier approaches for DEC-POMDPs. However, the centralized detection and evaluation of CLs limits the scalability of TREMOR. Thus, researchers introduced Distributed TREMOR (D-TREMOR), an extension of TREMOR that detects and evaluates CLs in a distributed fashion. To ensure convergence, D-TREMOR employs two mechanisms – probabilistic shaping of agent models to resolve positive CLs and a prioritization scheme that determines which agent model to shape to resolve negative CLs. As a result of these enhancements, D-TREMOR is able to solve large-scale DPCL problems with hundreds of agents.

### Algorithm 1: TREMOR()

1. $\pi^* \leftarrow$ SOLVEINDIVIDUALPOMDPs($\{\mathcal{M}_i\}_{i \leq n}$)
2. $\pi \leftarrow \phi$
3. $\text{iter} \leftarrow 0$
4. \textbf{while} $\pi \neq \pi^*$ and $\text{iter} < \text{MaxIterations}$ \textbf{do}
5. \hspace{1em} $\text{Active CLs} \leftarrow \text{COMPUTEACTIVECLS}(\{\mathcal{M}_i\}_{i \leq n}, \text{all CLs})$
6. \hspace{2em} \text{for all } cl \in \text{Active CLs} \text{ do}
7. \hspace{3em} $\{\text{val}_i\}_{i \in \text{cl agents}} \leftarrow \text{EVALUATECL}(cl)$
8. \hspace{2em} $\{\mathcal{M}_i\} \leftarrow \text{SHAPEMODELS}(cl, \{\text{val}_i\}, \{\mathcal{M}_i\}_{i \in \text{cl agents}})$
9. \hspace{1em} $\pi^* \leftarrow \pi$
10. $\pi \leftarrow$ SOLVEINDIVIDUALPOMDPs($\{\mathcal{M}_i\}_{i \leq n}$)
11. $\text{iter} \leftarrow \text{iter} + 1$

### Table 1: Classification of Prioritization Schemes

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL-Independent</td>
<td>SI schemes</td>
<td>DI schemes</td>
</tr>
<tr>
<td>CL-Dependent</td>
<td>SD schemes</td>
<td>DD schemes</td>
</tr>
</tbody>
</table>

4. PRIORITIZED SHAPING

Although the prioritization scheme introduced in D-TREMOR helps in speeding up the algorithm, it is rather ad-hoc. The scheme is based on agent IDs, which are arbitrary. As a result, one can construct simple examples where the scheme can lead to arbitrarily bad results; we describe one such example later.

In this paper, we adopt a more systematic approach to prioritization by (1) defining the prioritization problem, (2) classifying possible prioritization schemes, and (3) introducing prioritization schemes for some of these classes. We define the prioritization problem as the problem of finding an ordering of agent priorities for each CL and (planning) iteration such that the agents together optimize their joint policy assuming that agents with lower priorities shape their models in favor of agents with higher priorities and agents compute optimal individual policies based only on their own agent models (shaped or otherwise).

We classify possible prioritization schemes along two axes – (a) CLs, where schemes are either CL-independent or CL-dependent; and (b) iterations, where schemes are either static or dynamic. The ordering of agent priorities are identical across iterations in static schemes and can differ across iterations in dynamic schemes. Similarly, the ordering of agent priorities are identical across CLs in CL-independent
schemes and can differ across CLs in CL-dependent schemes. Table 1 provides a tabular classification of these schemes. We first describe the general prioritized shaping algorithm and then describe the individual schemes in Sections 4.2 and 4.3.

4.1 PD-TREMOR

We implement the schemes over D-TREMOR and refer to the new extension as Prioritized D-TREMOR (PD-TREMOR). Algorithm 2 shows the pseudocode with the priority related functions – SetInitialPriority(), ShapeModel(), UpdatePriority() – implementing the different prioritization schemes. The overall algorithm has the same distributed structure as the D-TREMOR, with each "self" agent i running in parallel to other agents in the system. Each agent i now maintains a priority \(R_{i,cl}^{iter}\) for each CL cl and iteration iter. The key steps of PD-TREMOR are as follows:

1. Each agent starts by shaping its own model with the ShapeInitialModel() function assuming that each CL cl will occur with a probability \(prior_{cl}\) (line 1). In D-TREMOR, the agents use \(prior_{cl} = 0\) for all negative CLs and \(prior_{cl} = 1\) for all positive CLs to shape its model.

2. Each agent then computes its individual policy \(\pi_i\) assuming no other agents exist in the environment (line 2).

3. Each agent then sets its initial priorities with the SetInitialPriority() function (lines 4-5). If PD-TREMOR uses CL-independent schemes, then the priorities are identical across all CLs. If PD-TREMOR uses CL-dependent schemes, then the priorities can be different across CLs.

4. An agent then computes its active CLs (line 7) and for each active CL, it evaluates the effect of that CL (line 9), the probability of that CL occurring (line 10) and broadcasts that information together with its priorities to the other agents (line 11).

5. Upon receiving the CL and priority information of all other agents (line 12), each agent shapes its model according to those priorities with the ShapeModel() function (line 13). Intuitively, for each CL, low priority agents shape their models in favor of higher priority agents.

6. Like D-TREMOR, each agent then solves their individual POMDPs with their newly shaped models (line 14).

7. Lastly, if PD-TREMOR uses dynamic schemes, then each agent updates its priorities using information from its current iteration with the UpdatePriority() function (lines 16-17). If PD-TREMOR uses static schemes, then the function does nothing since the initial priorities should not change across iterations.

If PD-TREMOR uses CL-independent schemes, then the function fixes the priority of at least one agent for all CLs in each iteration. The priority of an agent at a CL is fixed if it is no longer allowed to change in future iterations. Thus, PD-TREMOR is guaranteed to converge in \(n\) iterations, where \(n\) is the number of agents.

8. The agents repeat these steps for a maximum number of iterations (line 6).

4.2 CL-Independent Schemes

We now describe CL-independent schemes. There are two subcategories of schemes that fall under this category:

- **Static CL-independent (SI) schemes** are schemes that find an identical ordering of agent priorities across CLs and iterations. Since the priorities do not change across iterations and lower priority agents shape their models in response to higher priority agents, the agent model of at least one agent is fixed at each iteration. Therefore, all schemes in this category are guaranteed to converge in \(O(n)\) (Theorem 1).

The agent-ID-based prioritization scheme introduced for D-TREMOR is one example that falls under this category. In that scheme, agents with smaller IDs have higher priority than agents with larger IDs across all CLs and iterations. Although this scheme has been shown to speed up D-TREMOR, it can lead to arbitrarily bad results in some cases. To illustrate one such case, we use our running example. If we use the agent-ID-based scheme, that is, robot R1 has a higher priority than robot R2 across all CLs and iterations, then the reward of the team is 5 (for rescuing victim V1). (We described the reasoning earlier in Section 3.2.) On the other hand, if we invert the ordering, that is, robot R2 has a higher priority than robot R1, then robot R1 will shape its model to avoid the CLs. (We do not illustrate the shaped models as they are symmetric to the shaped model of robot R2 in Figures 2(d), 2(e) and 2(f).) As a result, the new policy of robot R2 is its initial policy and the new policy of robot R1 is to stay stationary. Thus, the joint reward of the team is 10 (for rescuing victim V2), twice larger than the previous case.

Alternatively, instead of agent IDs, one can employ heuristics that are problem-dependent, such as an ordering of expected values of the individual policies of the agents in the first iteration or the expected values of the MDP policies of the agents in the first iteration. However, it is still easy to find examples where such schemes perform badly due to their static nature. Therefore, we introduce schemes that find an ordering of agent priorities that can differ across iterations below.

- **Dynamic CL-independent (DI) schemes** are schemes

---

**Algorithm 2 PD-TREMOR(Agent i)**

1: \(M_i \gets \text{ShapeInitialModel}(M_i, allC\{\text{prior}_{cl}\})\)
2: \(\pi_i \gets \text{SolveIndividualPOMDP}(M_i)\)
3: \(iter \gets 0\)
4: for all \(cl \in allC\) do
5: \(R_{i,cl}^{iter} \gets \text{SetInitialPriority}(M_i, cl, \pi_i)\)
6: while \(iter < MaxIterations\) do
7: \(\alpha \{\text{cl}, val_{i,cl}\} \gets \text{ComputeActiveCLs}(M_i, allC, \pi_i)\)
8: for all \(cl \in \alpha \{\text{cl}\}\) do
9: \(\text{val}_{i,cl} \gets \text{EvaluateCL}(M_i, cl, \pi_i)\)
10: \(\text{pr}_{i,cl} \gets \text{ComputePriorCL}(M_i, cl, \pi_i)\)
11: \(\text{CommunicateCL}(i, cl, \text{pr}_{i,cl}, \text{val}_{i,cl}, R_{i,cl}^{iter})\)
12: \(\text{recvCLs} \gets \text{ReceiveCLs}()\)
13: \(M_i \leftarrow \text{ShapeModel}(M_i, \text{recvCLs}, \{R_{i,cl}^{iter}\})\)
14: \(\{\pi_i, \text{val}_i\} \leftarrow \text{SolveIndividualPOMDP}(M_i)\)
15: \(iter \leftarrow iter + 1\)
16: for all \(cl \in allC\) do
17: \(R_{i,cl}^{iter} \leftarrow \text{UpdatePriority}(\text{val}_i, \text{val}_{i,cl})\)
that find an identical ordering of agent priorities across CLs but this ordering can differ across iterations. We introduce the ExpectedValueDependent scheme, which associates the priority of an agent with the expected value of their individual policies. The larger the expected value of an agent, the higher the priority of that agent. The intuition is that agents with small expected rewards should shape their models so that they can find individual policies with higher expected rewards. This scheme is dynamic across iterations since the individual policies can change across iterations. However, this scheme ensures that the priority of at least one agent is fixed at each iteration to ensure convergence. Thus, it takes at most \( n \) iterations to fix the agent models of all agents. Therefore, like SI schemes, all schemes in this category are also guaranteed to converge in \( O(n) \) (Theorem 1).

Using our running example, this scheme will determine that \( R2 \) have a higher priority over \( R1 \) since the expected value of \( R2 \)'s policy at the first iteration is 10 while the expected value of \( R1 \)'s policy is 5. Thus, this ordering is the same as the one described above, and the joint reward of the team is thus 10 (for rescuing victim \( V2 \)).

### 4.3 CL-Dependent Schemes

We now describe CL-dependent schemes. Like CL-independent schemes, there are also two subcategories of schemes that fall under this category:

- **Static CL-dependent (SD) schemes**, which find an identical ordering of agent priorities across iterations but this ordering can differ across CLs. One example SD scheme is a scheme that associates the priority of an agent for a CL with the gain in the expected value of that agent’s individual policy if it shapes its model to resolve the CL. The larger the gain of an agent, the lower the priority of that agent. The intuition is that agents with large gains should shape their models so that they can find individual policies with higher expected rewards. Since the ordering of agent priorities should be identical across iterations, the scheme uses the ordering computed for the first iteration in all iterations.

- **Dynamic CL-dependent (DD) schemes**, which find an ordering of agent priorities that can differ across CLs and iterations. One example DD scheme is a scheme like the one above, except that it associates the priorities at each iteration with gains in that iteration, instead of the gains in the first iteration only.

One can trivially see that all CL-independent schemes are also CL-dependent schemes but not vice versa. Thus, an optimal CL-dependent scheme is guaranteed to perform no worse than any CL-independent scheme. However, it is difficult to design good CL-dependent schemes as there are often groups of CLs whose priorities should depend on one another. We will use our running example to illustrate one such example. Recall that there are three negative CLs – \( \langle t1, A2 \rangle, \langle t2, B2 \rangle \) and \( \langle t3, C2 \rangle \). A robot will need to have a high priority for all three CLs for it to rescue its victim. Thus, the priorities for all three CLs should be coupled.

Additionally, bad CL-dependent schemes can perform significantly worse than bad CL-independent schemes. Using our running example again, instead of CL-independent schemes described earlier, imagine the following CL-dependent prioritization: robot \( R1 \) has a higher priority than robot \( R2 \) for the \( \langle t2, B2 \rangle \) CL and vice versa for the \( \langle t1, A2 \rangle \) and \( \langle t3, C2 \rangle \) CLs. Then, the robots will shape their models to avoid the CLs that they have lower priorities for, resulting in the shaped models in Figure 3. Unfortunately, the new policy of both robots with their shaped models is to stay stationary since they cannot get to their victims. Thus, the joint reward of the team is 0, which is lower than the joint reward of 10 with a CL-independent scheme.

Lastly, convergence is not guaranteed for PD-TREMOR with CL-dependent schemes; we performed preliminary experiments with some CL-dependent schemes and they failed to converge before the cut-off time. Thus, we have focused our work on CL-independent schemes for the rest of this paper.

### 4.4 Computation of Initial Policies

We now describe different heuristics to compute initial agent policies. In D-TREMOR, each agent shapes its initial model assuming that all positive CLs will occur and all negative CLs will not occur before computing its initial policy. This optimistic assumption allowed better exploration of the overall space of interactions and hence leads to better joint policies. This was specifically shown in the context of debris interactions. In this paper, we argue that it might not always be sufficient to make optimistic assumptions. More specifically, there is an entire range between pessimistic and optimistic assumptions that can potentially affect the final joint policy.

Thus, in this paper, we generalize the framework of D-TREMOR by associating an initial prior probability of occurrence \( prior_{cl} \) to each \( CL \). An agent sets \( prior_{cl} \) to 0 for all negative CLs and 1 to all positive CLs if it is optimistic and vice versa if it is pessimistic. Agents in PD-TREMOR then use these priors to shape its initial model with the \text{ShapeInitialModel()} function (line 1).

This prior probability of CLs can potentially be computed from the solution to a relaxed version of the problem, where...
there is full observability or no transition uncertainty. While we analyze the effect of the prior on the overall solution quality and convergence, we have not explored the computation of prior in this paper and is left to future work.

4.5 Convergence Bounds

We now show proof sketches on the bounds for convergence of PD-TREMOR using CL-independent schemes and a specific case of CL-dependent schemes.

**Theorem 1.** PD-TREMOR with a CL-independent prioritization scheme converges in at most \( n \) iterations, where \( n \) is the number of agents in the DPCL problem.

**Proof Sketch:** Within CL-dependent prioritization schemes, we have static and dynamic schemes. With a static scheme, PD-TREMOR does not impose a restriction on fixing an agent’s model, however, since there are static priority, at every iteration one agent’s model is fixed. Thus, it takes \( n \) iterations to converge over there. With dynamically changing priorities, PD-TREMOR ensures that the priority of the agent with the highest expected value is fixed (for all iterations from that point onwards). Due to this, at each iteration there is at least one agent which change its model and hence its policy. Hence, it will take \( n \) iterations for convergence in the worst case.

**Theorem 2.** If PD-TREMOR with a CL-dependent prioritization scheme fixed the model of at least one agent for at least one CL at each iteration, then the algorithm converges in \( O(nm) \), where \( n \) is the number of agents and \( m \) is the number of CLs in the DPCL problem.

**Proof Sketch:** If we look at the simplest case of one CL and a set of agents. It is easy to see that this is similar to the case proved in Proposition 1 and hence takes \( n \) iterations. The worst scenario for \( n \) agents and \( m \) CL is where, the CLs become active sequentially after the shaping for the previous CL is performed. That is to say, we have one active CL initially and once the shaping with respect to that CL is performed until convergence (which takes \( n \) iterations), a new CL becomes active due to the new policies and so on. In this case, it is obvious to see that there will be at most \( n \times m \) iterations for convergence.

5. EXPERIMENTS

A set of experiments were conducted to study the performance of PD-TREMOR under various interaction types and conditions. DPCLs representing the problem domain described in Section 2 were constructed to exemplify particular combinations of positive and negative interactions. Each problem instance was solved once by PD-TREMOR and 5 times with (non-prioritized) D-TREMOR. As D-TREMOR contained probabilistic shaping heuristics, these multiple runs were necessary to measure characteristic performance, which was unnecessary for PD-TREMOR’s deterministic prioritization heuristics.

<table>
<thead>
<tr>
<th>Map</th>
<th># Rescue</th>
<th># Cleaner</th>
<th># Width</th>
<th># Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>25</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>50</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>25</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>25</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2: Summary of the pilot problem instances.

Figure 4: Expected value of solution policies in pilot dataset.

5.1 Pilot Experiment

First, a small set of problem instances was generated in an attempt to initially observe the behavior of PD-TREMOR in an intuitive problem. Four tall, thin maps were generated with numbers of agents varying from 9 to 12, and a height varying from 7 cells to 10 cells. A combination of positive and negative interactions was introduced through the random placement of debris and narrow corridors along the map centerline. The details of these maps can be found in Table 5.

Figure 4 shows the expected value of the solution policies generated by D-TREMOR and PD-TREMOR. In these experiments, it was evident that PD-TREMOR outperformed D-TREMOR in average-case performance across most maps.

After demonstrating that PD-TREMOR had the potential to match and exceed D-TREMOR’s performance in a deterministic way, a set of larger experiments were constructed to characterize the behavior of the algorithm in larger systems. In this experiment, debris-only and collision-only datasets were constructed of 10 problem instances, each involving a square 9×9 grid map upon which agents, victims, debris, and narrow corridors were placed randomly. In the collision-only maps, 50 rescue agents and 50 victims were placed. In the debris-only maps, 25 rescue agents and 25 debris-clearing agents were placed, along with 25 pieces of debris and 25 victims.

5.2 Collision Interaction Dataset

The purpose of this dataset was to study the effects of prioritization on negative interactions between agents. In these problems, team performance could be substantially decreased if two agents entered the same narrow corridor at the same time. Thus, agents had to determine when particular corridors are occupied, and which agents should defer passage in order to allow others through.

Figure 5(a) shows the expected value of the solution policies generated by D-TREMOR and PD-TREMOR. As D-TREMOR is non-deterministic, its performance data is displayed as a boxplot over each of the 10 problem instances. While overall value was highly specific to individual map in-
stances, PD-TREMOR’s policies consistently matched and exceeded the value of average D-TREMOR policies on most maps. This suggests that dynamic prioritization offers competitive performance when resolving negative interactions.

An additional benefit of the dynamic prioritization can be seen in Figure 5(b), a plot of the total time taken for 10 iterations of each algorithm. Here, the solid line represents PD-TREMOR, and the dotted line represents the time taken by D-TREMOR. In every case, PD-TREMOR is able to complete the same number of iterations faster, meaning that a more efficient policy search is being performed.

5.3 Debris Interaction Dataset

The purpose of this dataset is to study the effects of prioritization on positive interactions between agents. In these problems, team performance can be substantially increased if cleaner agents remove debris from the paths of rescue agents. However, not all debris lies on the desired paths of the rescue agents, and clearing debris from the path of an agent only matters if the agent can subsequently reach a victim and attain a reward. Thus, agents must determine which debris should be cleared, and whether the expected gain of the debris clearing is worth the cost of moving a cleaner agent. A second dataset was created with purely positive interactions (debris that could be cleared to enable agent movement). As explained in Section 4, a prior belief in the likelihood of the interaction occurring needed to be selected for the initial computation of policies. Each problem instance was tested over a set of 5 priors: \{0.0, 0.25, 0.5, 0.75, 1.0\}.

As seen in Figure 5(c), the solution quality of PD-TREMOR in the positive-interaction case was matched or exceeded by the randomization heuristics in D-TREMOR in most cases. This was surprising, as the pilot results suggested that PD-TREMOR would outperform D-TREMOR given an appropriate selection of prior. PD-TREMOR still achieved rapid convergence, as predicted. Figure 6 shows the policy values quickly stabilizing for various priors on a randomly selected maps from the dataset.

In addition, the time taken by PD-TREMOR was found to be highly sensitive to the choice of prior. Figure 5(d) shows the average planning time for each prior value as well as for D-TREMOR. As the choice of prior reached 1.0, the amount of time the planner took increased greatly. This seems to follow the intuition that the choice of prior in a positive interaction affects the feasible states that will be explored by the planner, by essentially opening up a new set of states that were previously too low-valued to consider.

5.4 Density Experiments

The divergence of the debris interactions from the behavior observed in the pilot experiments suggested that there were other problem characteristics capable of affecting planning performance. One significant way in which the randomly-generated maps used in the debris dataset differed from the tall-thin maps used in the pilot experiment was in their respective densities of interacting cells. Thus, another experiment was designed to study the effects of interaction density on performance.

A density-varying dataset was constructed in which agents were placed on tall, thin grids, with rescue agents on one side and victims on the other. Debris and narrow corridors were inserted between the two sides, forcing agents to interact frequently in order to reach victims. Two conditions were created, with 10 problem instances for each condition. In the first condition, the height of the map was 25 cells, while in the second condition, the height of the map was 13 cells. In both cases, there were 13 rescue agents and 12 cleaner agents on each map, and correspondingly 13 victims and 12 debris cells. Both positive debris interactions and negative collision interactions were possible in these problems. However, the density of these interactions was varied by distributing a fixed set of robots, debris, and victims along a vertical column of varying height. This highlighted the effects of interaction density on prior selection, as well as demonstrating the performance of PD-TREMOR on problem instances with a mixture of interaction types.

The resulting policy values can be seen in Figures 7(a) and 7(b). In the high density map, PD-TREMOR meets or exceeds D-TREMOR average-case performance. However, in the lower-density problem instances, PD-TREMOR and D-TREMOR have similar performance, and in several cases, PD-TREMOR performs worse. This suggests that at least one of the major factors that impacts the efficacy of dynamic prioritization is the density of agent interactions.

However, PD-TREMOR takes more computation time than D-TREMOR in both cases, as seen in Figures 7(c) and 7(d), despite the disparity in performance gain. The reason for this is unclear, but it may be that the time taken by PD-TREMOR is decoupled from its performance, suggesting that there are further directions for improving performance without sacrificing computational simplicity.

6. RELATED WORK

The extensions from TREMOR and D-TREMOR have already been explained. In this section, we will describe about recent research on solving sparse interaction DEC-POMDPs and DEC-MDPs.

Witwicki et al. [13] have characterized weakly coupled interactions and used the characterization to provide bounds on solution quality. Becker et al. [1] introduced approaches for solving transition independent DEC-MDPs. ND-POMDPs [4] exploited interaction structure in transition and observation independent DEC-POMDPs. Oliehoek et al. [6] exploit static interaction structure in generic DEC-POMDPs. Seuken et al. [7] provide memory bounded dynamic programming (MBDP) approaches for solving general DEC-POMDPs. While MBDP and its variants solve considerably higher horizon problems, they have been primarily
limited to two agent problems.

The key difference of PD-TREMOR from that of the above techniques is due to prioritized and distributed planning, a trait that assists in scaling to problems with large numbers of agents while providing high quality solutions.

7. CONCLUSION

This paper presented PD-TREMOR, a distributed POMDP algorithm that uses prioritization to guarantee convergence to a solution. The empirical performance of the algorithm is sensitive to the prior belief that an agent has about whether some interaction will occur. Surprisingly, the prioritization did not always improve performance, if the density of interactions was low and/or there were mostly positive interactions, randomly resolving the interactions appears to work as well or better. Moreover, there is considerable variation in exactly how much the prioritization helps performance. Our ongoing work is looking to try to understand the details of the interactions that cause a technique that works well in so many algorithms to sometimes perform poorly for distributed POMDPs.

8. REFERENCES

Coordinated Multi-Agent Learning for Decentralized POMDPs

Chongjie Zhang
Computer Science Dept.
University of Massachusetts
Amherst, MA 01002, US
chongjie@cs.umass.edu

Victor Lesser
Computer Science Dept.
University of Massachusetts
Amherst, MA 01002, US
lesser@cs.umass.edu

ABSTRACT

In many multi-agent applications such as distributed sensor nets, a network of agents act collaboratively under uncertainty and local interactions. Networked Distributed POMDP (ND-POMDP) provides a framework to model such cooperative multi-agent decision making. Existing work on ND-POMDPs has focused on offline techniques that require accurate models, which are usually costly to obtain in practice. This paper presents a model-free, scalable learning approach that synthesizes multi-agent reinforcement learning (MARL) and distributed constraint optimization (DCOP). By exploiting structured interaction in ND-POMDPs, our approach distributes the learning of the joint policy and employs DCOP techniques to coordinate distributed learning to ensure the global learning performance. Our approach can learn a globally optimal policy for ND-POMDPs with a property called groupwise observability. Experimental results show that, with communication during learning and execution, our approach significantly outperforms the nearly-optimal non-communication policies computed offline.

INTRODUCTION

Decentralized partially observable MDP (DEC-POMDP) provides a powerful framework for modeling cooperative multi-agent decision making problems under uncertainty. Due to the intractability of optimally solving general DEC-POMDPs, research has focused on restricted versions of DEC-POMDP that are easier to solve yet rich enough to represent many practical applications. Networked Distributed POMDP (ND-POMDP) [1] is one such model that is inspired by a real-world sensor network coordination problem [2]. ND-POMDP assumes transition and observation independence and locality of interaction.

A rich portfolio of algorithms have been developed for solving ND-POMDPs [1, 3, 4]. One good feature of these techniques is that, although computing policies is centralized or requires extensive communication, executing computed policies does not require explicit communication. However, this feature may prevent agents from better coordination during execution when communication is allowed. In fact, in many practical applications, communications (at least between neighboring agents) are necessary for agents to perform tasks. For example, for target tracking in sensor networks, agents need to fuse their observations and actions to determine sensing results. The work [5] introduced communication in ND-POMDPs to periodically synchronize the belief state and extended existing algorithms to obtain policies with longer horizons. However, extensive communication is required for global synchronization, which is not scalable. In addition, all these algorithms for ND-POMDPs are offline techniques and require accurate models of the environment, which are usually costly to obtain in practice.

In this paper, we present a model-free, scalable learning approach to developing policies for ND-POMDPs. Our approach synthesizes multi-agent reinforcement learning (MARL) and distributed constraint optimization (DCOP). By exploiting locality of interactions in ND-POMDPs, our approach factors a global joint action-value function and distributes the learning of the joint policy, which potentially scales up the learning to large-scale ND-POMDPs. Using communication between neighboring agents, our approach employs DCOP techniques to coordinate distributed learning to ensure the global performance. Coordinated reinforcement learning based on coordination graphs [6] has been explored in [7, 8] for factored MDPs. In contrast to these previous work, in this paper, we explore coordinated multi-agent reinforcement learning in a principled way in ND-POMDPs and prove that our coordinated learning approach can learn the globally optimal policy for ND-POMDPs with a property, called groupwise observability. In addition, we also demonstrate that a max-sum algorithm [9] can be used for an approximate solution to our distributed coordination problem in learning, which requires limited communication overhead (typically scaling linearly with the number of agents) and computation. This DCOP algorithm can be readily implemented as an anytime algorithm to trade off solution quality and cost of computation and communication. Unlike the message-passing algorithm in [8], this algorithm can be directly used for coordinating interactions involving more than two agents. Our previous work [10, 11] presented a general supervisory framework for coordinating MARL, but did not provide a general coordination algorithm. In this paper, we demonstrate that DCOP algorithms can be used as general techniques for coordinating MARL in ND-POMDPs. Experimental results show that, even in ND-POMDPs without groupwise observability, our approach scales to larger domains and performs significantly better and with orders of magnitude time savings (in the offline mode) over the previous best offline algorithm. Note that, as our approach needs communication during execution, a direct comparison among approaches is not appropriate. However, the offline results do provide a way to evaluate our approach by providing a baseline (i.e., nearly-optimal performance without communication).

BACKGROUND

This section briefly introduces an illustrative problem in the sensor network domain, the ND-POMDP model, and basic learning
Figure 1: A 4-chain sensor configuration

approaches.

Illustrative Domain

This illustrative problem is motivated by a real-world challenge, where a network of agents (sensors) are used to track targets. Figure 1 shows a specific problem instance consisting of four sensors. Here, each sensor node can scan in one of four directions: North, South, East or West. To track a target and obtain the associated reward, two sensors with overlapping scanning areas must coordinate by scanning the same area simultaneously. For example, sensor1 needs to scan East and sensor2 needs to scan West simultaneously to track a target in location1. Thus, sensors have to act in a coordinated fashion. The movement of targets is unaffected by sensor agents. Sensors have imperfect observability of the target, so there can be false positive and negative observations. Sensors receive a reward on successfully tracking a target, and they incur a cost, when they either scan an area in an uncoordinated fashion or when the target is absent.

Networked Distributed POMDPs

Observe that sensors in this domain are mostly independent. Their state transitions, given the target location and the observations, are independent of the actions of the other agents. The only dependence arises from the fact that two agents must coordinate by scanning the same region to track a target. This dependence can be translated into a joint reward function. Such dependence is usually localized among a few agents (only two agents in this sensor network problem). The ND-POMDP model [1] was introduced to express such a type of interactions.

Definition 1. An ND-POMDP is defined by the tuple 
\( \langle I, S, A, \Omega, P, O, R, b \rangle \), where

- \( I = \{1, \ldots, n\} \) is a set of agent indices.
- \( S = \times_{i \in I} S_i \times S_u \). \( S_i \) refers to the local state of agent \( i \). \( S_u \) refers to a set of uncontrollable states that are independent of the actions of the agents. In the sensor network example, \( S_i \) is empty, while \( S_u \) corresponds to the set of locations where targets can be present.
- \( A = \times_{i \in I} A_i \), where \( A_i \) is the set of actions for agent \( i \). For the sensor network example, \( A_i = \{N, W, E, S, Off\} \).
- \( \Omega = \times_{i \in I} \Omega_i \) is the joint observation set.
- \( P \) is the joint transition function.
- \( O \) is the joint observation function.
- \( R \) is the reward function.

\[ b(b_u, b_1, \ldots, b_n) \] is the initial belief (or distribution) for joint state \( s = \langle s_u, s_1, \ldots, s_n \rangle \in S \) and \( b(s) = b(s_u) \cdot \prod_{i \in I} b_i(s_i) \), where \( b_u \) and \( b_i \) are the initial distribution over \( S_u \) and \( S_i \).

The goal for ND-POMDPs is to compute a joint policy \( \pi \) that maximizes the total expected reward of all agents over a finite horizon \( T \) starting from \( b \). Without communication, agents can only act based on its local observations. In this case, a joint policy \( \pi \) is defined by \( (\pi_1, \ldots, \pi_n) \), where \( \pi_i \) refers to the individual policy of agent \( i \) that maps its history of observations to an action \( a_i \in A_i \). If communication is allowed, a joint policy \( \pi \) can also be defined by one policy, called global policy, that maps from a history of joint observations to a joint action \( a \in A \). This is because agents can exchange their observations and select actions based on joint observations. Obviously, the optimal global policy inherently performs better than the optimal set of individual policies. In this paper, we assume agents can communicate (at least with their neighbors) during the entire time.

Basic Learning Approaches

To learn the joint policy, we need to define Q-function (or Q-value function). Let Q-function \( Q(h, a) \) represent the expected reward of doing joint action \( a \) with history \( h \) of joint observations and actions and behaving optimally from then on. The globally joint policy \( \pi \) can be derived from \( Q(h, a) \) by setting \( \pi(h) = \arg\max_{a \in A} Q(h, a) \).

In principle, we can directly estimate \( Q(h, a) \) by using standard single-agent Q-learning:

\[ Q(h', a') = (1 - \alpha)Q(h', a') + \alpha[r' + \gamma \max_{a} Q(h^{t+1}, a)] \] (1)

where \( \alpha \in (0, 1) \) is the learning rate, \( r' \) is the immediate reward of doing \( a' \) for observation history \( h' \), \( \gamma \in [0, 1] \) is the discount factor, which is usually set to 1 for a finite horizon. We call this approach globally joint learning. Although this approach leads to an optimal policy, it is practically intractable, because the policy space is exponential in the number of agents and the agents might not have access to the needed information (i.e., observations, actions, and rewards of all other agents) for learning and selecting actions.

At the other extreme, we can have the independent learning approach [12] in which agents ignore the actions and rewards of the other agents, and concurrently learn their own action-value functions solely based on their local observations and rewards. To provide local rewards in ND-POMDPs, we can split the reward component \( R_l \) evenly among agents in group \( l \). This approach is distributed, results in big storage and computational savings in the policy space, and does not require communication during learning and execution. However, this approach lacks coordination and
might lead to oscillations or converge to local optimal policies. For example, in Figure 1, if location1, location2, and location3 always have targets with sensing reward 50, 60, and 50, respectively, then, by using independent learning approach, sensor2 and sensor3 will learn to always sense location2, which is locally optimal with average expected reward 60. However, the optimal policy is that sensor1 and sensor2 always sense location1 and sensor3 and sensor4 always sense location3, whose global expected reward is 100. Therefore, some form of coordination is needed in order to learn the globally optimal policy.

**COORDINATED MULTI-AGENT REINFORCEMENT LEARNING**

As discussed in the previous section, directly learning the globally joint policy in a centralized way is infeasible from a practical perspective, while independent learning is a distributed, scalable approach, but may yield poor global performance. In this section, we present a coordinated multi-agent learning approach for ND-POMDPs that attempts to achieve both scalability and optimality (or near-optimality). This approach distributes the learning by exploiting structured interactions in ND-POMDPs and coordinates distributed learning to ensure the global performance.

Our approach optimizes a decomposable Q-function \( \hat{Q}(\tilde{h}, a) \) that is used to approximate the global Q-function \( Q(\hat{h}, a) \). This Q-function \( Q(\tilde{h}, a) \) is defined as a sum of smaller local Q-functions based on hyperlinks in the interaction hypergraph of ND-POMDPs, that is,

\[
\hat{Q}(\tilde{h}, a) = \sum_{l \in E} Q_l(\tilde{h}_l, a_l),
\]

where \( Q_l(\tilde{h}_l, a_l) \) is the expected reward for agents on hyperlink \( l \) by doing joint action \( a_l^* \) at joint history \( \tilde{h}_l \) and behaving globally optimally from then on in respect to maximizing \( Q(\tilde{h}, a) \). We will show in the next subsection that this approximation becomes exact for ND-POMDPs with a property called *groupwise observability*, which will lead to the theoretical result of optimality for our approach. In fact, this approximation is reasonable for general ND-POMDPs. This is because the global reward in ND-POMDPs is the sum of local rewards of groups defined on hyperlinks in the interaction hypergraph, and, as a result, \( Q(\tilde{h}, a) \) and \( Q(\tilde{h}, a) \) are strongly positively correlated. Therefore, maximizing \( \hat{Q}(\tilde{h}, a) \) can potentially optimize \( Q(\tilde{h}, a) \). Our experimental results will verify this hypothesis on ND-POMDPs without the groupwise observability property.

Q-learning is used to learn the optimal \( \hat{Q}(\tilde{h}, a) \). With the decomposition in (2), the global Q-learning update rule in (1) can be rewritten as

\[
\sum_{l \in E} Q_l(\tilde{h}_l, a_l^*) = (1 - \alpha) \sum_{l \in E} Q_l(\tilde{h}_l, a_l) + \alpha [\sum_{l \in E} r_l + \gamma \max_a Q(\tilde{h}^{t+1}, a)]
\]

Note that the discounted future reward, \( \max_a \hat{Q}(\tilde{h}^{t+1}, a) \), can not be directly written as the sum of local discounted future rewards, because it depends on the joint action that maximizes the global reward. Fortunately, we can accomplish this by defining the joint action \( a^* = \arg\max_a Q(\tilde{h}^{t+1}, a) \) and \( \max_a \hat{Q}(\tilde{h}^{t+1}, a) = \hat{Q}(\tilde{h}^{t+1}, a^*) = \sum_{l \in E} Q_l(\tilde{h}_l^{t+1}, a_l^*) \). We are now able to decompose all the terms in (3) and write the update rule for each group:

\[
Q_l(\tilde{h}_l, a_l) = (1 - \alpha) Q_l(\tilde{h}_l, a_l) + \alpha [r_l + \gamma \max_a \hat{Q}_l(\tilde{h}_l^{t+1}, a_l^*)]
\]

Similar to Sparse Cooperative Q-Learning [8], update rule in (4) is based on local reward and Q-function, except for \( a_l^* \). Note that the local contribution \( Q_l(\tilde{h}_l^{t+1}, a_l^*) \) of group \( l \) to the global action value might be lower than \( \max_a Q_l(\tilde{h}_l^{t+1}, a_l) \), the maximizing value of its local Q-function, because it is unaware of the dependencies among groups. We will use distributed constraint optimization (DCOP) techniques to compute \( a_l^* \), which will be discussed later. Update rule in (4) is different from coordinated reinforcement learning approach in [7], where local Q-function update depends on the global reward signal and the global Q-value, which are not usually specifically tailored to local behaviors, thus resulting in slower learning convergence.

Using update rule in (4), our approach distributes the learning of the global function \( Q \) among groups. Our approach assumes that each group has a delegate agent (which can be chosen arbitrarily from a group) that learns \( Q_l \) on behalf of the group. The basic learning process is as follows. During each learning cycle \( t \), after executing actions \( a_l^* \), agents in group \( l \) receive and transmit their observations to the delegate agent of their group and the delegate agent receives its group reward signal \( r_l^* \). Using its observed history \( \tilde{h}_l^{t+1} \), the delegate agent then computes the next best action \( a_l^+ \) for \( \tilde{h}_l^{t+1} \) by using a DCOP technique and updates its Q-function \( Q_l \) using rule (4). Finally, it distributes the next actions to its group members, which will be \( a_l^+ \) or some exploration actions.

The learned global Q-function is distributedly represented by local Q-functions of delegate agents. As a result, during execution, agents’ action selections are computed online in a distributed manner by a DCOP algorithm from local Q-funcions. Note that local Q-function \( Q_l(\tilde{h}_l, a_l^*) \) is defined on the observation history of group \( l \), which scales exponentially with the horizon. To deal with a large horizon, one approach is to use a fixed-size window of observations, as we did in our experiments. Other more sophisticated approaches (i.e., utilize suffix memory [13]) for dealing with partial observability can also be used with our approach.

In next two subsections, we will formally analyze the optimality of our approach and discuss how to compute joint action selections for learning or execution.

**Optimality Analysis**

In this section, we first define a property for ND-POMDPs, called *groupwise observability*, and then prove that our approach can learn an optimal policy for ND-POMDPs with this property.

**DEFINITION 2.** An ND-POMDP is said to have groupwise observability if, for all \( l \in E \), the set of observations \( \omega_l = \langle \omega_{l1}, \ldots, \omega_{lk} \rangle \) made by agents on hyperlink \( l \) together fully determine the current uncontrolled state, that is, if \( \forall i \in \omega_l \exists u_i : Pr(s_u|\omega_l) = 1 \).

Note that this property does not imply that agents can observe their local states or states of other agents. It does imply that, for each agent \( i \in l \), \( P_l(s_i'|s_i, a_i, \omega_l) = P_l(s_i'|s_i, a_i, \omega_l) \) and \( O_l(\omega_l|s_i, a_i, \omega_l) = O_l(\omega_l|s_i, a_i, \omega_l) \), which means, given joint observation \( \omega_l \), observation and transition of agent \( i \) on \( l \) are completely independent of observations and actions of other agents, and, as a result, its local belief update only depends on its local action and observation. This further implies that, in ND-POMDPs with groupwise observability, the local belief of agent \( i \) on \( l \) can be fully determined by its initial local state and the history of joint observations and actions of agents on \( l \).

The theoretical result of optimality of our approach is as follows.

**THEOREM 1.** For ND-POMDPs with groupwise observability, under basic assumption of Q-learning and by using update rule (4), \( Q_l(\tilde{h}_l, a_l) \) will converge to the optimal \( Q_l^*(\tilde{h}_l, a_l) \), for all \( l \in E \).
and the policy \( \pi^*(\tilde{h}) = \arg\max_a \sum_{t \in E} Q^*_t(\tilde{h}_t, a_t) \) is globally optimal.

The proof for this theorem can be conducted by showing that Q-function \( Q \) defined in Equation (2) is exactly the same as the objective function \( Q \) of ND-POMDPs. This is because, if the approximation of \( Q \) is exact, then our coordinated learning approach described above is essentially a distributed version of update rule (1) that uses Q-learning, which leads to the global optimal \( Q^*(\tilde{h}, a) \). The exactness of this approximation for ND-POMDPs with groupwise observability will be shown by Proposition 2.

Our proof first defines a Q-function with state variables, then shows it is decomposable, and finally uses this result to prove the approximation of \( Q \) to \( Q \) is exact for ND-POMDPs with groupwise observability. To simplify the equations, we introduce some abbreviations:

\[
\begin{align*}
    p^t_i &\equiv P_t(s^t_i | s^t_{-i}, a^t_i) \cdot O_t(\omega^t_i | s^t_i, a^t_i) \\
p^t_u &\equiv P_u(s^t_u | s^t_t) \\
r^t_i &\equiv R_t(s_t, s_u, a_t) \\
Q^t &\equiv Q^t(s^t_i, \tilde{h}^t_i, a^t_i) \\
Q^t_i &\equiv \max_a Q^t(s^t_i, \tilde{h}^t_i, a) \\
\end{align*}
\]

The global Q-function \( Q(s^t_i, \tilde{h}^t_i, a^t_i) \) with state will satisfy the Bellman equation:

\[
Q(s^t_i, \tilde{h}^t_i, a^t_i) = R(s^t_i, a^t_i) + \gamma \sum_{s^t_{+1}, \omega^t_{+1}} p^t_{u} p^t_{i} \ldots p^t_{n} Q^{t+1},
\]

where \( \tilde{h}^t_{i+1} \) is \( \tilde{h}^t_i \) appended by \( (a^t_i, \omega^t_{+1}) \).

Let \( b^t \) be the belief state at time \( t \). As \( b^t \) is fully determined by the initial belief \( b \) and history \( \tilde{h}^t_i \) of joint observations and actions, we have

\[
\begin{align*}
    Q(\tilde{h}^t_i, a^t_i) &= \sum_{s \in S} b^t(s) Q(s^t_i, \tilde{h}^t_i, a^t_i). \tag{5}
\end{align*}
\]

Similarly, we define a Q-function for each hyperlink \( l \):

\[
\begin{align*}
    Q^t_i(s^t_l, \tilde{h}^t_l, a^t_l) &= r^t_l + \gamma \sum_{s^t_{+1}, \omega^t_{+1}} p^t_u p^t_l \ldots p^t_h Q^{t+1}_l,
\end{align*}
\]

where \( \tilde{h}^{t+1}_l \) is \( \tilde{h}^t_l \) appended by \( (a^t_l, \omega^t_{+1}) \) and \( Q^{t+1}_l \) denotes \( Q_i(s^t_{+1}, \tilde{h}^{t+1}_l, a^t) \), where \( a^t_l \) is the globally optimal joint action taken by agents on \( l \) in the next global state and history of joint observations and actions of all agents.

For ND-POMDPs with groupwise observability, as \( b^t_i(s_u) \) is fully determined by history \( \tilde{h}^t_i \) of joint observations and actions, and, for \( i \in l \), \( b^t_i(s_u) \) is fully determined by the initial belief \( b_i(s_i) \) and history \( \tilde{h}^t_i \); we then have

\[
\begin{align*}
    Q(\tilde{h}^t_l, a^t_l) &= \sum_{s_u, s_i} b^t_i(s_u, s_i) Q_i(s_i, s_u, \tilde{h}^t_l, a^t_l). \tag{6}
\end{align*}
\]

PROPOSITION 1. In ND-POMDPs, the global function \( Q^t(s^t_i, \tilde{h}^t_i, a^t_i) \) for any finite horizon \( T \) is decomposable, that is,

\[
Q^t(s^t_i, \tilde{h}^t_i, a^t_i) = \sum_{l \in E} Q^t_i(s^t_l, s^t_{-i}, \tilde{h}^t_l, a^t_l). \tag{7}
\]

PROOF. Proof is by mathematical induction. Proposition holds for \( t = T - 1 \) because \( r^t = \sum_{l \in E} r^t_l \) and there is no future reward. Assume it holds for \( t \) where \( 1 \leq t \leq T - 1 \), that is,

\[
Q^{t-1} = R(s^{t-1}, a^{t-1}) + \gamma \sum_{s^t_{+1}, \omega^t_{+1}} p^t_u \ldots p^t_n Q^t.
\]

Now let us show that proposition holds for \( t - 1 \).

\[
\begin{align*}
    Q^{t-1} &= R(s^{t-1}, a^{t-1}) + \gamma \sum_{s^t_{+1}, \omega^t_{+1}} p^t_u \ldots p^t_n Q^t \\
    &= \sum_{l \in E} \left[ r^{t-1}_l + \gamma \sum_{s^t_{+1}, \omega^t_{+1}} p^{t-1}_u \ldots p^{t-1}_n \right] Q^t_l \\
    &= \sum_{l \in E} Q^{t-1}_l \\
    &= \sum_{l \in E} Q^t_l
\end{align*}
\]

Based on Proposition 1, Equation 5 and 6, we can show an exact decomposition of the Q-function without state.

PROPOSITION 2. In ND-POMDPs with groupwise observability, the global Q-value function \( Q^t(\tilde{h}^t_i, a^t_i) \) for any finite horizon \( T \) is decomposable, that is,

\[
Q^t(\tilde{h}^t_i, a^t_i) = \sum_{l \in E} Q^t_l(\tilde{h}^t_l, a^t_l). \tag{8}
\]

PROOF. \( Q(\tilde{h}^t_i, a^t_i) = \sum_{s_u, s_{+1}, \ldots, s_n} b^t_i(s_u) b^t_l(s_l) \ldots b^t_h(s_h) \).

\[
= \sum_{l \in E} Q^t_l(s_i, s_u, \tilde{h}^t_l, a^t_l) \\
= \sum_{l \in E} \sum_{s^l, s^t} b^t_l(s^l, s^t) Q^t_l(s^t, s^l, \tilde{h}^t_l, a^t_l) \\
= \sum_{l \in E} Q^t_l(\tilde{h}^t_l, a^t_l).
\]

This proposition completes the proof of Theorem 1. Note that the work [1] also showed that the value function of a given joint policy can be decomposed and the work [14] generalized its result to factored Dec-POMDPs. However, unlike our Q-value function decomposition, decomposition components of their value function are defined on a joint policy, so their results are not directly applicable to the proof of Theorem 1.

**Optimal Joint Action Selection**

Our learning approach requires computing the joint action that maximizes the global Q-value function for updating local Q-functions or for acting during execution. We can formulate this problem as a DCOP, which is defined by a set of discrete variables \( a = \{a_1, \ldots, a_n\} \), where \( a_i \in A_i \) is controlled by agent \( i \) and represents its action choice, and a set of functions \( Q = \{Q_l | l \in E\} \), where \( Q_l \) is the Q-value function for hyperlink \( l \). Note that history \( \tilde{h} \) is fixed for every computation, so we will ignore it in the following discussion and denote \( Q_l(\tilde{h}, a) \) by \( Q_l(a) \). The goal is to find the joint action \( a^* \), such that the global Q-value function, the sum of all Q-functions, is maximized, that is, \( a^* = \arg\max_a \sum_{l \in E} Q_l(a) \). We can represent this DCOP as a factor graph by creating a node for each variable and for each function and connecting a function node to a variable node if the corresponding function is dependent upon that variable. The resulting graph is bipartite.
A variable elimination algorithm [6] can be used to compute an optimal solution for this DCOP, but it requires extensive communication and computation (scaling exponentially with the induced width of the agent interaction graph). In this paper, we investigate the max-sum algorithm [9] for an approximate solution, which requires much less communication and computation and can be readily implemented as an anytime algorithm to trade off the quality and efficiency of computing joint actions. Unlike the max-plus algorithm in [8], this algorithm can be directly used for coordinating interactions involving more than two agents.

The max-sum algorithm operates directly on the factor graph, and does so by specifying the messages that should be passed from variable to function nodes, and from function nodes to variable nodes, which are defined as follows:

- **Message from variable node** $i$ **to function node** $l$:
  $$q_{i \rightarrow l}(a_i) = \sum_{g \in \mathcal{F}_i \setminus l} r_{g \rightarrow l}(a_i) + c_{i l}$$

  where $\mathcal{F}_i$ is a vector of function indexes, indicating which function nodes are connected to variable node $i$, and $c_{i l}$ is a normalizing constant to prevent the messages from increasing endlessly in cyclic graphs.

- **Message from function node** $l$ **to variable node** $i$:
  $$r_{l \rightarrow i}(a_i) = \max_{a_i \setminus a_i} [Q_l(a_i) + \sum_{g \in V_l \setminus i} q_{g \rightarrow l}(a_g)]$$

  where $V_l$ is a vector of variable indexes, indicating which variable nodes are connected to function node $l$ and $a_i \setminus a_i = \{a_i : g \in V_l \setminus i\}$.

Here variable node $i$ is agent $i$ who needs to select its action and function node $l$ is the delegate agent of hyperlink $l$ that hosts the Q-value function $Q_l$. If the factor graph is cycle-free, the algorithm is guaranteed to converge to the optimal global solution such that each agent $i$ can find its optimal action $a^*_i$ by locally calculating $a^*_i = \arg\max_{a_i} z_l(a_i)$, where $z_l(a_i) = \sum_{g \in F_l} r_{g \rightarrow l}(a_i)$. Otherwise, there is no guarantee of convergence. However, extensive empirical results show that, even in this case, the algorithm frequently provides good solutions. Before convergence, the value $z_l(a_i)$ of agent $i$ calculated from incoming messages is actually an approximation of the exact value of action $a_i$ given other agents act optimally. Therefore, the max-sum algorithm can be implemented as an anytime algorithm by controlling the number of rounds of passing messages, which will trade off the quality and efficiency (or communication cost) of the action selection. In addition, the max-sum algorithm is essentially distributed. Its messages are small (linearly scaling with the maximum number of actions of agents), the number of messages typically varies linearly with the number of agents and hyperlinks, and its computational complexity scales exponentially with the maximum size of hyperlinks (which typically is much less than the total number of agents).

**EXPERIMENTS**

To evaluate our coordinated learning (CL) approach in general ND-POMDPs, we experimented it in the illustrative sensor network domain, which does not have the groupwise observability property. We compared CL with the independent learning (IL) approach (described in the Background Section) and CBDP [4], one of the most efficient algorithms for ND-POMDPs. We conducted experiments with configurations shown in Figure 2. The first three configurations are introduced in [3], but we changed their initial beliefs to an uniform distribution over ten states to increase problem difficulty. The 25-grid sensor network has two targets with the same sensing rewards as 15-3D, but has a larger state space and longer target paths.

Since both CL and IL are model-free, we develop a simulator for ND-POMDPs to learn and evaluate policies. The evaluation process is as follows: for each ND-POMDP, we use CBDP to solve it and get its joint policy, then run both learning approaches in a simulator for that ND-POMDP, whose learning time is set to some ratio of CBDP’s computation time, and, finally evaluate learned policies and CBDP’s policy in the simulator. The solution quality for each horizon is indicated by the expected global reward for that horizon. Solution quality is computed over 10000 simulation runs. Results are then averaged over 10 experiments and the deviation is computed, which is very small (under 5) and not shown properly in the following figures. The learning rate $\alpha$ is set to 0.001 and discount factor $\gamma = 1$. Both learning approaches learned policies that map fixed-windows of observations (with size $\leq 4$) to an action even for scenarios with horizon greater than 5. To trade off the speed and solution quality, we restricted the max-sum algorithm passing messages at most 4 rounds for each joint action computation (except for experiments of controlling communication).

Figure 3 (a) shows the solution quality of CL and IL with different learning time on the configuration 15-3D with horizon $T = 10$. The configuration 15-3D is the most complex problem instance for CBDP. The $x$ axis represents the ratio of learning time to CBDP’s computation time, which is plotted with a logarithmic scale. The performance of both CL and IL generally increases with more training time. We observe that CL can learn policies, whose performance surpasses that of CBDP’s policy, with learning time two orders of magnitude less than CBDP’s computation time. However, IL performs much worse than CL and CBDP. One reason is that, as we have discussed, IL can only converge to local optima, which is far away from the global optimal solution on the configuration 15-3D. This result actually illustrates the importance of the coordination during learning and execution. Another reason is that IL (and CL) uses fixed-window policy that maps up to 4 observations to an action, while CBDP’s policies with horizon $T = 10$ maps up to 9 observations to an action. We did observe that IL could perform comparably or better than CBDP on smaller problems with small horizon (e.g., one the domain 11-Helix with 5 horizon).

Figure 3 (b) shows the solution quality over a range of horizons on the configuration 15-3D. We can see that the solution quality of CL linearly increases with the horizon size, whose increase rate is greater than CBDP. This indicates that CL can potentially scale better than CBDP with the horizon size. Figure 3 (c) shows the solution quality on other configurations, where 15-Mod is the modified
REFERENCES


ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation under Award No. IIS-116078. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.


