Producing efficient error-bounded solutions for transition independent decentralized MDPs

Jilles Steeve Dibangoye, Christopher Amato, Arnaud Doniec, François Charpillet

Producing Efficient Error-bounded Solutions for Transition Independent Decentralized MDPs

Jilles S. Dibangoye
INRIA
Loria, Campus Scientifique
Vandœuvre-lès-Nancy, France
jilles.dibangoye@inria.fr

Christopher Amato
CSAIL
MIT
Cambridge, MA, USA
camato@csail.mit.edu

Arnaud Doniec
Université Lille Nord de France
Mines Douai, Département IA
F-59500 Douai, France
arnaud.doniec@mines-douai.fr

François Charpillet
INRIA
Loria, Campus Scientifique
Vandœuvre-lès-Nancy, France
francois.charpillet@loria.fr

ABSTRACT

There has been substantial progress on algorithms for single-agent sequential decision making using partially observable Markov decision processes (POMDPs). A number of efficient algorithms for solving POMDPs share two desirable properties: *error-bounds* and *fast convergence rates*. Despite significant efforts, no algorithms for solving decentralized POMDPs benefit from these properties, leading to either poor solution quality or limited scalability. This paper presents the first approach for solving transition independent decentralized Markov decision processes (Dec-MDPs), that inherits these properties. Two related algorithms illustrate this approach. The first recasts the original problem as a deterministic and completely observable Markov decision process. In this form, the original problem is solved by combining heuristic search with constraint optimization to quickly converge into a near-optimal policy. This algorithm also provides the foundation for the first algorithm for solving infinite-horizon transition independent decentralized MDPs. We demonstrate that both methods outperform state-of-the-art algorithms by multiple orders of magnitude, and for infinite-horizon decentralized MDPs, the algorithm is able to construct more concise policies by searching cyclic policy graphs.

Categories and Subject Descriptors

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

Keywords

Planning under uncertainty, cooperative multiagent systems, decentralized POMDPs

1. INTRODUCTION

Multi-agent planning and coordination problems are common in many real-world domains. The decentralized POMDP (Dec-POMDP) provides a general mathematical model for cooperative multi-agent decision-making under uncertainty, but solving it optimally is intractable [7]. As a result, while several optimal approaches have been developed for general decentralized POMDPs, they do not scale to more than two agents or even moderately sized problems [3, 4, 6, 8, 12, 25, 26]. Approximate algorithms for general decentralized POMDPs scale better, but without the theoretical guarantees of optimal or near-optimal algorithms [2, 9, 11, 19, 20, 22, 27, 29, 30]. Therefore, in general, we are constrained to either solving small toy problems near-optimally, or solving larger problems but possibly producing poor solution quality.

One alternative is to consider more tractable subclasses of decentralized POMDPs. Transition independent decentralized MDPs represent a general subclass, where agents possess perfect observability of a local state (but not those of the other agents) and agents can only interact with one another through reward functions. The standard approach to solving this problem is to recast it as a set of augmented MDPs, which incorporate the joint reward into local reward functions for each agent [5]. This approach relies on local value functions that map states and the complete sets of policy candidates of the other agents to expected values, requiring a large amount of time and memory. A bilinear programming (BLP) method [21] has also been proposed for this problem, but it requires incorporating time into the state and relies on sparse reward interaction (rewards that are additive in many states) for much of its scalability. A recent algorithm by Dibangoye et al. uses a different approach that represents a policy explicitly as a sequence of mappings from states to actions called a *Markovian policy* [10]. This algorithm represents a value function as a mapping from state probability distributions to expected values, but fails to generalize expected values over unvisited state probability distributions, which results in slow convergence rates.

In this paper, we consider an alternative approach to solving transition independent decentralized MDPs by recasting them as continuous state MDPs where states and actions are state probability distributions and decision rules, respectively. We show that this formulation possesses a piecewise linear convex value function, permitting ideas for solving POMDPs to be utilized. We then present two related algorithms that possess both solution quality *error bounds* and *fast convergence rates* similar to those found in solving POMDPs. The first algorithm improves Dibangoye et

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

al.'s approach by exploiting the piecewise linear and convex value function to allow expected values to generalize over unvisited state probability distributions. It provides the foundation for a related infinite-horizon algorithm, which can produce an ϵ-optimal solution by representing the policy as a cyclic Markovian policy graph. While both approaches outperform state-of-the-art algorithms by multiple orders of magnitude, the second algorithm is also able to produce significantly more concise policies.

The remainder of this paper is organized as follows. First, we describe the decentralized MDP framework and discuss related work. We then present theoretical results, showing that the value function for the finite-horizon case is piecewise linear and convex over the state probability distributions. Next, we describe improvements over Dibangoye et al.'s algorithm, which include: *first*, the value function is now represented as a piecewise linear and convex function; *secondly*, after each trial of the algorithm, we maintain a concise value function by means of pruning. We further introduce a related algorithm for solving the infinite-horizon case within any ϵ of the optimal solution. Finally, we present an empirical evaluation of these algorithms with respect to 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 14 agents.

2. BACKGROUND ON DEC-MDPS

In a decentralized MDP, the actions of a given agent do not affect the transitions of the other agents. In this case we say that the problem is *transition independent*. The agents interact with one another only through a common reward function. After taking an action, each agent receives a local observation, which here fully determines it current local state. Despite this *local full observability* property, each agent's local observation is insufficient to optimize the selection of its next decision. This is mainly because agents may not have access to the local observations of the other agents. However, if all agents shared their local observations, the true state of the world would be known. It is the presence of this *joint full observability* property that differentiates decentralized MDPs from decentralized POMDPs. These characteristics appear in many real-world applications including: navigation problems, *e.g.,* Mars exploration rovers [5, 21]; network sensors, *e.g.,* distributed sensor net surveillance [18]; and smart grids, *e.g.,* distributed smart-gird management.

2.1 The Dec-MDP Model

DEFINITION 2.1 (THE DECENTRALIZED MDP). *A N-agent decentralized MDP $(S, A, p, r, \eta_0, \beta)$ consists of:*

- *A finite set $S = Z^1 \times \cdots \times Z^N$ of states $s = (z^1, \ldots, z^N)$, where Z^i denotes the set of local observations z^i of agent $i = 1, 2, \ldots, N$.*
- *A finite set $A = A^1 \times \cdots \times A^N$ of joint actions $a = (a^1, \ldots, a^N)$, where A^i denote the set of local actions a^i of agent $i = 1, 2, \ldots, N$.*
- *A transition function $p\colon S \times A \times S \mapsto [0, 1]$, which denotes the probability $p(\bar{s}|s, a)$ of transiting from state $s = (z^1, \ldots, z^N)$ to state $\bar{s} = (\bar{z}^1, \ldots, \bar{z}^N)$ when taking joint action $a = (a^1, \ldots, a^N)$.*
- *A reward function $r\colon S \times A \mapsto \mathbb{R}$, 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 η_0; and β, the discount factor.*

As noted above, decentralized MDPs are distinguished by the state being *jointly fully observable*. This property ensures that the global state would be known if all agents shared their local observations at each time step (i.e., there is no external uncertainty in the problem) and follows trivially from the definition of states as observations for each agent.

2.2 Additional Assumptions

Throughout this paper, we are interested in decentralized MDPs that exhibit two main properties.

The first is the transition independence assumption where the local observation of each agent depends only on its previous local observation and local action taken by that agent.

ASSUMPTION 2.2. *An N-agent decentralized MDP is said to be **transition independent** if there exists, for all agent i, a local transition function $p^i\colon Z^i \times A^i \times Z^i \mapsto [0, 1]$, such that*

$$p(\bar{s}|s, a) = \prod_{i=1}^{N} p^i(\bar{z}^i|z^i, a^i),$$

where $s = \langle z^1, z^2, \ldots, z^N\rangle$ and $a = \langle a^1, a^2, \ldots, a^N\rangle$.

We also implicitly assume observation independence, in which the observation function of each agent does not depend on the dynamics of the other agents. This hold since we assume the transition and observation functions are the same in the Dec-MDP model (as states are made up of local observations of the agents).

Transition independent decentralized MDPs have both properties. When the agents operate over a bounded number of time-steps (typically referred to as the problem horizon) T, the model is referred to as a *finite-horizon* case. When the agents operate over an unbounded number of time-steps, the model is referred to as the *infinite-horizon* case.

2.3 Additional Definitions and Notations

A decentralized MDP is solved by finding a joint policy represented as a sequence of rules for selecting joint actions at each time step τ, called *decentralized decision rules* σ_τ. Note that the policies are decentralized in the sense that action choices depend only on local information.

Decentralized decision rule σ_τ is a N-tuple of decision rules $\langle \sigma_\tau^1, \ldots, \sigma_\tau^N\rangle$, one individual decision rule for each agent. Each individual decision rule σ_τ^i is a mapping from *local information* about the states of the process at time step τ to local actions. We further distinguish between *history-dependent* policies and *Markovian* policies.

History-dependent decision rule σ_τ^i is a mapping from local action-observation histories $h_\tau^i = \langle a_0^i, z_1^i, \ldots, a_{\tau-1}^i, z_\tau^i\rangle$ to local actions $\sigma_\tau^i(h_\tau^i) = a_\tau^i$. A policy that consists of history-dependent decision rules defines a history-dependent policy. Standard approaches for solving decentralized MDPs (and Dec-POMDPs) search in the space of history-dependent policies.

Markovian decision rule σ_τ^i, however, maps local observations z_τ^i to local actions $\sigma_\tau^i(z_\tau^i) = a_\tau^i$. A policy that consist of Markovian decision rules defines a Markovian policy. Clearly, the space of Markovian policies is exponentially smaller than that of history-dependent policies. Under transition independent assumptions, Goldman et al. [13] and more recently Dibangoye et al. [10] demonstrated that there always exists a Markovian policy that achieves performance at least as good as any history-dependent policy for

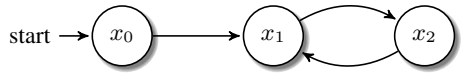

Figure 1: A policy represented as a Markovian policy graph.

transition independent Dec-MDPs. For this reason, algorithms that search in the Markovian policy space will be more efficient than those that search in the history-independent policy space.

Markovian policy graphs are introduced in this paper to represent Markovian policies for both finite and infinite-horizon cases. More formally, a Markovian policy graph is defined with the tuple $(X, \Sigma, \nu, \mu, x_0)$, where X is the set of nodes in the graph; Σ denotes the set of decision rules; $\nu\colon X \mapsto \Sigma$ describes the mapping that determines the decision rule $\nu(x)$ to be executed when the graph is in node x; mapping $\mu\colon X \mapsto X$ is the deterministic transition function, providing the next graph node for each node; and x_0 denotes the initial node. This graph allows for a concise representation of a nonstationary policy. An example of a joint policy represented as a Markovian policy graph is illustrated in Figure 1. Note that decentralized decision rules for each node (x) are not shown. Similar graphs (in the form of finite-state controllers) have been used to represent policies for general Dec-POMDPs [6], but in our case, actions (decision rules) and transitions are deterministic and transitions do not depend on the observation seen.

The occupancy state is also a useful concept. The occupancy state η_τ, where $\forall \bar{s}\colon \eta_\tau(\bar{s}) \stackrel{\text{def}}{=} P(s_\tau = \bar{s} \mid \eta_0, \sigma_0, \ldots, \sigma_{\tau-1})$, represents a probability distribution over states given an initial state distribution η_0, and a joint policy prior to step τ. The occupancy state can be updated at each step to incorporate the latest decentralized decision rule. That is, $\eta_\tau(\bar{s}) = \sum_{s \in S}\ p(\bar{s}|s, \sigma_{\tau-1}(s)) \cdot \eta_{\tau-1}(s)$. We also denote $\triangle$ the space of occupancy states, that is the standard $|S|$-dimensional simplex. Dibangoye et al. [10] demonstrated that the occupancy state is a sufficient statistic for a given system under the control of a sequence of decentralized decision rules $\langle \sigma_0, \ldots, \sigma_{\tau-1} \rangle$.

Also, note the difference between occupancy states and *belief states*. Formally, a belief state b_τ, where $b_\tau(s) \stackrel{\text{def}}{=} P(s|h_\tau, \eta_0)$, denotes the probability that the system is in state s if the system's history is h_τ starting in state probability distribution η_0. The total probability property provides, $\eta_\tau(s) = \sum_{h_\tau} P(s|h_\tau, \eta_0) \cdot P(h_\tau|\eta_0)$. In words, this equation states that at the τ-th time step all belief states are summarized in a single occupancy state.

3. RELATED WORK

Most exact algorithms for decentralized MDPs, including those assuming transition independence, search directly in the policy space [2, 4, 5, 6, 8, 9, 11, 17, 19, 20, 21, 22, 25, 26, 29, 30]. Given some explicit representation of policies, these algorithms build a joint policy that optimizes a performance objective (or comes acceptably close to doing so). We assume the objective is to maximize the expected total discounted reward (where $\beta \in (0, 1]$ is a discount factor). Exact algorithms that search in the policy space must, nonetheless, be able to compute a value from the joint policy they iteratively improve.

We discuss two alternative representations of the *value function* under transition independence assumptions in the finite-horizon case. One possibility is to view a value function as a mapping from joint policies to expected values. Given a decentralized Markovian policy $\pi \equiv \langle \sigma_0, \sigma_1, \ldots, \sigma_{T-1} \rangle$, this representation of the value function stores the expected value of taking that policy starting in occupancy state η_0: $\upsilon_\beta(\pi) \stackrel{\text{def}}{=} E(\sum_{\tau=0}^{T-1}\ \beta^\tau \cdot \rho(\eta_\tau, \sigma_\tau) \mid \eta_0, \pi)$, where

Algorithm 1: Markovian Policy Search

```
mps begin
    Initialize the bounds ῠ_β and ῡ_β.
    while ῡ_{β,0}(η_0) − ῠ_{β,0}(η_0) > ε do
        mp-trial(η_0)

mps-trial(η_τ) begin
    if ῡ_{β,τ}(η_τ) − ῠ_{β,τ}(η_τ) > ε then
        Select σ_greedy according to ῡ_β and η.
        Update upper bound value function.
        Call mp-trial (χ[η_τ, σ_greedy]).
        Update lower bound value function.
```

the expected immediate reward for taking decentralized Markovian decision rule σ_τ in occupancy state η_τ is given by $\rho(\eta_\tau, \sigma_\tau) = \sum_{s \in S} \eta_\tau(s) \cdot r(s, \sigma_\tau[s])$. Algorithms that use this representation must explicitly enumerate joint policies, as the value function does not generalize over unvisited joint policies. While these algorithms can be optimal, they must keep track of the complete set of Markovian policy candidates for each agent, requiring a large amount of time and memory [5, 21].

A second possibility is to represent a value function as a sequence $\langle \upsilon_{\beta,0}, \upsilon_{\beta,1}, \ldots, \upsilon_{\beta,T-1} \rangle$ of mappings $\upsilon_{\beta,\tau}\colon \triangle \mapsto \mathbb{R}$ from occupancy states to expected values, one for each time step $\tau = 0, 1, \ldots, T-1$. This is similar to the approach used by many POMDP solution methods to represent the value over the belief states [16]. Given this representation of the value function, a corresponding joint policy $\pi = \langle \sigma_0, \sigma_1, \ldots, \sigma_{T-1} \rangle$ is extracted using one-step lookahead: $\sigma_\tau = \arg\max_\sigma \rho(\eta_\tau, \sigma) + \beta \cdot \upsilon_{\beta,\tau+1}(\chi[\eta_\tau, \sigma])$, where $\chi[\eta_\tau, \sigma_\tau]$ denotes the occupancy state that results from taking decentralized Markovian decision rule σ_τ in occupancy state η_τ. Algorithms that use the value function represented as a mapping from occupancy states to expected values must traverse the space of occupancy states to update the value function following the *one-step-backup* for each visited occupancy state: for all $\tau = 0, 1, \ldots, T-1$,

$$\upsilon_{\beta,\tau}(\eta_\tau) = \max_\sigma \rho(\eta_\tau, \sigma_\tau) + \beta \cdot \upsilon_{\beta,\tau+1}(\chi[\eta_\tau, \sigma_\tau]), \qquad (1)$$

and $\upsilon_{\beta,T}(\eta_T) = 0$. In words, (eqn. 1) means that the value at occupancy state η_τ is given by the immediate reward for taking the best decentralized Markovian decision rule for η_τ plus the discounted expected value of the resulting occupancy state $\chi[\eta_\tau, \sigma_\tau]$. Algorithms that iteratively perform the one-step backup converge to the optimal value function in the limit. However, the one-step backup requires the explicit enumeration of all decentralized Markovian decision rules, which is computationally intensive and significantly limit the scalability.

Dibangoye et al. [10] recently introduced the first algorithm that solves a transition independent decentralized MDP over planning-horizon T by searching in the value function space, namely the *Markovian policy search* described in Algorithm 1. This algorithm searches to a finite depth T and finds a solution in the form of a tree that grows with the depth of the search. The search tree can be represented as *a decision tree* in which the nodes of the tree correspond to occupancy states and the root of the tree is the initial occupancy state η_0. Arcs are labeled by the choice of a decentralized Markovian decision rule. The value of a node is the maximum among the values of decentralized Markovian decision rules that follow each occupancy state. Upper and lower bounds are computed for occupancy states on the fringe of the search tree and backed up through the tree to the starting occupancy state at its root. Thus, expanding the search tree improves the bounds at the interior nodes of the

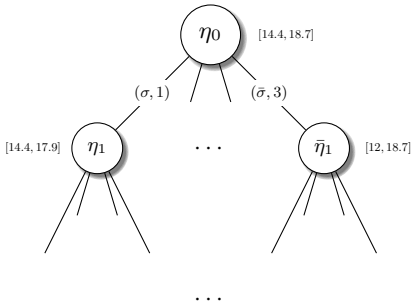

Figure 2: A decision-tree constructed by the Markovian policy search algorithm [10]. The state occupancy distributions are represented by circular nodes. Decentralized Markovian decision rules σ and rewards $\rho(\eta, \sigma)$ label the outgoing arcs from the nodes. The values inside the brackets represent the lower and upper bounds.

tree. The error bound $(\bar{\upsilon}_{\beta,\tau}(\eta_\tau) - \underline{\upsilon}_{\beta,\tau}(\eta_\tau))$ can be made arbitrarily small by expanding the search tree far enough and an ϵ-optimal value function for the occupancy state at the root of the tree can be found after a finite search. An example on how such a decision tree is constructed and evaluated is presented in Figure 2.

For lower and upper bound functions, Dibangoye et al. [10] use mappings from occupancy states to expected values. To describe how lower and upper bound functions are improved, recall that every node of the search tree corresponds to a occupancy state. Therefore, expanding a node (one arc at a time), and backing up its upper bound, is equivalent to performing a one-step backup for the corresponding occupancy state as in Equation (1). To circumvent the explicit enumeration of all decentralized Markovian decision rules, Dibangoye et al. implement the one-step backup using constraint optimization. In contrast, the lower bound is updated as follows. Beginning at the root of the search tree and selecting the decentralized Markovian decision rule that maximizes the upper bound of the current node, each decentralized Markovian decision rule that label arcs of the search tree that has been visited during the trial are identified. For each of these decentralized Markovian decision rules σ_τ, in backwards order from the leaf node of the search tree, the lower bound at the corresponding occupancy state η_τ is updated as follows $\underline{\upsilon}_{\beta,\tau}(\eta_\tau) = \max\{\underline{\upsilon}_{\beta,\tau}(\eta_\tau), \rho(\eta_\tau, \sigma_\tau) + \beta \cdot \underline{\upsilon}_{\beta,\tau+1}(\chi[\eta_\tau, \sigma_\tau])\}$. The algorithm terminates when the error bound at the initial occupancy state is small enough. It is guaranteed to terminate after a finite number of trials with a ϵ-optimal joint policy.

Dibangoye et al. [10] represent lower and upper bound functions in a way that does not allow them to generalize over unvisited occupancy states. As a consequence, to improve lower or upper bound value at a given occupancy state, the algorithm needs to explicitly visit it and perform the updates for that specific occupancy state. However the number of occupancy states that must be backed up to guarantee ϵ-optimality is often intractable. The key for improving the convergence rate of the Markovian policy search algorithm, is to leverage the piecewise linear and convex structure of the exact value function of transition independent decentralized MDPs.

4. THE FINITE-HORIZON CASE

This section discusses a new algorithm for solving transition independent Dec-MDPs over planning-horizon T by re-casting the model as a continuous-state MDP. This allows algorithms to be developed that take advantage of the *piecewise linear and convex* structure of the value function to improve efficiency.

DEFINITION 4.1. *Given a transition independent decentralized Markov decision process (S, A, p, r), the corresponding Markov decision process $(\triangle, \Sigma, \chi, \rho)$ is given by:*

- *A continuous-state space $\triangle$ is the space of occupancy states.*
- *An action set Σ is the space of decentralized Markovian decision rules.*
- *A deterministic transition function $\chi\colon \triangle \times \Sigma \mapsto \triangle$, which defines the next occupancy state $\chi[\eta, \sigma] = \bar{\eta}$ when taking decentralized decision rule σ in occupancy state η, where $\bar{\eta}(\bar{s}) = \sum_{s \in S}\ p(\bar{s}|s, \sigma[s]) \cdot \eta(s)$.*
- *A reward function $\rho\colon \triangle \times \Sigma \mapsto \mathbb{R}$, where quantity $\rho(\eta, \sigma)$ denotes the expected immediate reward when executing decentralized decision rule σ in occupancy state η.*

By recasting a transition independent decentralized MDP as a completely observable MDP with a continous, $|S|$-dimensional state space that consists of all possible occupancy states, the problem can be solved using POMDP theory and algorithms. To better understand this relationship, notice that standard approach to solving a POMDP is to recast it as a completely observable MDP with a state space that consists of all possible belief states. In this form, Smallwood and Sondik [23] demonstrate that the value function defined for all possible state probability distributions can be represented as a piecewise linear and convex function. Transition independent decentralized MDPs inherit this property.

Let $\upsilon^*_\beta\colon \triangle \times \mathbb{N} \mapsto \mathbb{R}$ be the exact value function over planning-horizon T, such that value $\upsilon^*_\beta(\eta, \tau) = \upsilon_{\beta,\tau}(\eta)$ depends on the current occupancy state η and horizon τ, where $\upsilon_{\beta,\tau}(\eta, \tau) = 0$ for $\tau \geq T$. We are now ready to present our main theoretical result.

THEOREM 4.2. *The value function $\upsilon^*_\beta\colon \triangle \times \mathbb{N} \mapsto \mathbb{R}$ solution of the system of Equations (1) is piecewise linear and convex.*

PROOF. We proceed by induction. The $(T-1)$-th value function $\upsilon_{\beta,T-1}$ is piecewise linear and convex since $\rho(\cdot, \sigma)\colon \triangle \mapsto \mathbb{R}$ is a linear combination of rewards $r(s, \sigma[s])$ for any decentralized Markovian decision rule σ. Suppose $\upsilon_{\beta,\tau+1}\colon \triangle \mapsto \mathbb{R}$ is piecewise linear and convex. In demonstrating this property for $\upsilon_{\beta,\tau}\colon \triangle \mapsto \mathbb{R}$ defined for any occupancy state η_τ by $\upsilon_{\beta,\tau}(\eta_\tau) = \max_\sigma \rho(\eta_\tau, \sigma) + \beta \cdot \upsilon_{\beta,\tau+1}(\chi[\eta_\tau, \sigma])$, we first note that the transition function $\chi(\cdot, \sigma)\colon \triangle \mapsto \triangle$ is convex because it is a linear combination of transition probabilities. The $\max$ operator preserves piecewise linearity and convexity, and the sum of piecewise linear and convex functions is piecewise linear and convex. □

A piecewise linear and convex value function υ^*_β can be represented by a sequence of finite sets of $|S|$-dimensional vectors of real numbers $\langle \Lambda_0, \Lambda_1, \cdots, \Lambda_{T-1} \rangle$, such that the value of each occupancy state η_τ is given by:

$$\upsilon_{\beta,\tau}(\eta_\tau) \quad = \quad \max_{\upsilon \in \Lambda_\tau} \sum_{s \in S} \eta_\tau(s) \cdot \upsilon(s). \tag{2}$$

Algorithms for solving transition independent decentralized MDPs by searching in the value function space, can represent the value function in this way, allowing it to generalize over unvisited occupancy states. To illustrate this, we consider the Markovian policy search algorithm discussed earlier and show how to represent upper and lower bound functions $\bar{\upsilon}_{\beta,\tau}$ and $\underline{\upsilon}_{\beta,\tau}$ to exploit the piecewise linearity and convexity property of the exact τ-th value function $\upsilon_{\beta,\tau}$. Similar representations have been previously used by algorithms for solving POMDPs [15, 16, 24].

4.1 The Upper-Bound Function

We use a point set representation, $\Psi_\tau = \{(\eta_\tau^k, \phi_\tau^k) | k = 0, 1, \cdots\}$, for the upper bound function $\bar{v}_\beta \colon \triangle \times \mathbb{N} \mapsto \mathbb{R}$. The upper-bound value $\bar{v}_{\beta,\tau}(\eta_\tau) = \phi_\tau$ at any unvisited occupancy state η_τ is thus the projection of η_τ onto the convex hull formed by finite set of occupancy states and their corresponding values in Ψ_τ. This demonstrates the ability to generalize the upper-bound function $\bar{v}_\beta$ over unvisited occupancy states. Updates are performed by adding new points into point set after the application of a one-step backup. To initialize upper-bound function $\bar{v}_{\beta,\tau}$, we assume full observability and calculate the τ-th optimal value function of the underlying MDP. This provides the initial upper-bound values at corners of simplex $\triangle$, which form the initial point set Ψ_τ.

For algorithms that represent the upper-bound function in this way, it is important to achieve a fast upper-bound evaluation at any occupancy state. However the convex hull projection relies heavily on linear programming and is computationally intensive. One possible alternative, the so called *sawtooth projection*, uses approximate linear programming and leverages the special structure of the occupancy state simplex to improve efficiency [15]. Relative to the convex hull projection with the same point set, the sawtooth projection provides a weaker bound but much faster function evaluations. To better understand the complexity of the sawtooth projection, let $|\Psi_\tau|$ be the number of points in the point set that represents value function $\bar{v}_{\beta,\tau}$. The complexity $\mathcal{O}(|\Psi_\tau||S|)$ of each evaluation increases with increasing points in Ψ_τ. Given that evaluations occur multiple times during a trial, the importance of pruning away unnecessary points is clear. In practice, many points (η_τ, ϕ_τ) in point set Ψ_τ may be completely *dominated* by a combination of the other points. Those points can be pruned away without affecting the upper-bound accuracy. A point (η_τ, ϕ_τ) is said to be dominated if the remaining points in conjunction with the corner points satisfy inequality $\bar{v}_{\beta,\tau}(\eta_\tau) \leq \phi_\tau$.

4.2 The Lower-Bound Function

For the lower bound function $\underline{v}_\beta \colon \triangle \times \mathbb{N} \mapsto \mathbb{R}$, we use piecewise linear and convex functions $\langle \underline{v}_{\beta,0}, \underline{v}_{\beta,1}, \ldots, \underline{v}_{\beta,T-1} \rangle$. Each piecewise linear and convex value function $\underline{v}_{\beta,\tau} \colon \triangle \mapsto \mathbb{R}$ corresponds to a finite set of $|S|$-dimensional vectors of real numbers $\underline{\Lambda}_\tau$. Because each function $\underline{v}_{\beta,\tau}$ is represented as a piecewise linear and convex function, it generalizes over unvisited occupancy states. Updates are performed by adding new vectors into sets $\underline{\Lambda}_0, \underline{\Lambda}_1, \ldots$, and $\underline{\Lambda}_{T-1}$. To describe how to construct vector v_τ that is added into set $\underline{\Lambda}_\tau$, let's assume the algorithm starts in occupancy state η_τ, and selects σ_τ with respect to upper bound function, and then moves to occupancy state $\chi[\eta_\tau, \sigma_\tau]$. Recall that the lower bound function is updated backwards. Thus, we start by selecting vector $v_{\tau+1}$ in set $\underline{\Lambda}_{\tau+1}$ that is maximal at occupancy state $\chi[\eta_\tau, \sigma_\tau]$. We then compute vector v_τ as follows: $v_\tau(s) = r(s, \sigma[s]) + \beta \sum_{\bar{s}} p(\bar{s}|s, \sigma[s]) \cdot v_{\tau+1}(\bar{s})$. As a consequence, we add a new vector into set $\underline{\Lambda}_\tau$ after each trial. This may slow down lower bound function evaluations and updates.

Many vectors in sets $\underline{\Lambda}_0, \underline{\Lambda}_1, \ldots$, and $\underline{\Lambda}_{T-1}$ may be unnecessary to maximize the lower bound value at the starting occupancy state. These vectors can be pruned away without affecting the ability to find the optimal value function. To perform this pruning mechanism, we rely on a correspondence between vectors and one-step policy choices. Each vector in $\underline{\Lambda}_\tau$ corresponds to the choice of a decentralized Markovian decision rule, and the choice of a successor vector in $\underline{\Lambda}_{\tau+1}$. To describe the correspondence between vectors and one-step policy choices, we introduce the following notation. For each vector v_τ^x in $\underline{\Lambda}_\tau$, let $\nu(x)$ denote the choice of a decentralized Markovian decision rule, and let $\mu(x)$ denote the successor vector in $\underline{\Lambda}_{\tau+1}$. Given this correspondence between vectors and one-step policy choices, it is clear that any vector that is not reachable from the vector that maximizes the lower bound value of the starting occupancy state can be pruned away without affecting the value of the starting occupancy state. This correspondence also proves that an optimal joint polity for a finite-horizon transition independent decentralized MDP can be represented by an *acyclic Markovian policy graph* in which each node corresponds to a vector in nonstationary value function $\underline{v}_\beta \colon \triangle \times \mathbb{N} \mapsto \mathbb{R}$.

5. THE INFINITE-HORIZON CASE

Although the Markovian policy search algorithm is designed to solve finite-horizon decentralized MDPs with transition independent assumptions, it can also be used to solve infinite-horizon problems. The optimal value function for the infinite-horizon case is not necessarily piecewise linear, although it is convex. However it can be approximated arbitrarily closely by a piecewise linear and convex function. Building upon this insight, we use the Markovian policy search algorithm to calculate a value function $v_\beta^\epsilon \colon \triangle \times \mathbb{N} \mapsto \mathbb{R}$ over a finite planning-horizon T which achieves a *total β-discounted rewards* within ϵ of an optimal value function over an infinite-horizon. To this end, we chose T such that the regret $\mathbf{regret}(v_\beta^\epsilon)$ of operating only over $T = \left\lceil \log_\beta \left(\frac{(1-\beta)\epsilon}{||r||_\infty} \right) \right\rceil$ steps instead of an infinite number of steps is upper-bounded by ϵ:

$$\mathbf{regret}(v_\beta^\epsilon) \quad = \quad \sum_{\tau=T}^{\infty} \beta^\tau \cdot ||r||_\infty, \tag{3}$$

where $||r||_\infty = \max_{s,a} r(s,a) - \min_{s,a} r(s,a)$. As β gets closer to 1, planning-horizon T increases. This results in arbitrarily large sequence $\langle \underline{v}_{\beta,0}, \underline{v}_{\beta,1}, \ldots, \underline{v}_{\beta,T-1} \rangle$ of value functions.

Although the *improved* Markovian policy search algorithm can apply in infinite-horizon cases and converges more quickly than the *original* Markovian policy search algorithm, both are limited to solving finite-horizon problems with reasonable planning-horizon T. The shared bottleneck is that they need to find a (possibly large) sequence of optimal value functions $\langle v_{\beta,0}, v_{\beta,1}, \ldots, v_{\beta,T-1} \rangle$, one for each horizon. The fastest algorithm for solving finite-horizon decentralized MDPs with transition independent assumptions is still prohibitively slow for infinite-horizon cases when regret ϵ is significantly small as the planning-horizon increases with decreasing regret. In this section, we introduce a new approach, called *Markovian policy graph search*, for solving infinite-horizon decentralized MDPs with transition independent assumptions that is closely related to the Markovian policy search algorithm described in the previous section but differs in an important respect; it does not search for an ϵ-optimal nonstationary value function $v_\beta^\epsilon \colon \triangle \times \mathbb{N} \mapsto \mathbb{R}$ using a (possibly large) sequence of T value functions $v_{\beta,\tau} \colon \triangle \mapsto \mathbb{R}$. Instead, it optimizes a stationary value function $v_\beta^\epsilon \colon \triangle \mapsto \mathbb{R}$ to maximize a long-term expected cumulative discounted reward.

For both lower and upper bound functions, $\underline{v}_\beta^\epsilon \colon \triangle \mapsto \mathbb{R}$ and $\bar{v}_\beta^\epsilon \colon \triangle \mapsto \mathbb{R}$, we use representations that have been discussed earlier to exploit the piecewise linear and convex property of ϵ-optimal value function $v_\beta^\epsilon \colon \triangle \mapsto \mathbb{R}$, namely vector set $\underline{\Lambda}$ and point set Ψ, respectively. We also discussed how to update both lower and upper bound functions, we do not add to that discussion in the Markovian policy graph search algorithm, but we note again the one-to-one relationship between nodes in the policy graph and vectors in the lower bound. The words vector and node can be use interchangeably in this context. After each trial, we prune the upper bound function, much as described earlier. For the lower bound function, however, the pruning mechanism we described earlier may maintain unnecessary vectors (and thus unnecessary nodes) in set $\underline{\Lambda}$. To describe our pruning mechanism, let $\underline{\Lambda}'$ be the set of vector candi-

Algorithm 2: Markovian Policy Graph Search

mpgs **begin**
- Initialize the bounds $\underline{\upsilon}^{\epsilon}_{\beta}$ and $\bar{\upsilon}^{\epsilon}_{\beta}$.
- **while** $\bar{\upsilon}^{\epsilon}_{\beta}(\eta_0) - \underline{\upsilon}^{\epsilon}_{\beta}(\eta_0) > \epsilon$ **do**
  - mpgs-trial$(\eta_0, 0)$
  - Prune lower bound $\underline{\upsilon}^{\epsilon}_{\beta}$ and corresponding policy graph
  - Prune upper bound $\bar{\upsilon}^{\epsilon}_{\beta}$
  - Evaluate lower bound $\underline{\upsilon}^{\epsilon}_{\beta}$ and corresponding policy graph

mpgs-trial(η, τ) **begin**
- **if** $\bar{\upsilon}^{\epsilon}_{\beta}(\eta) - \underline{\upsilon}^{\epsilon}_{\beta}(\eta) > \epsilon\beta^{-\tau}$ **then**
  - Select σ_{greedy} according to $\bar{\upsilon}^{\epsilon}_{\beta}$ and η (adding a new node to the policy graph).
  - Update upper bound value function $\bar{\upsilon}^{\epsilon}_{\beta}$.
  - Call mpg-trial $(\chi[\eta, \sigma_{\text{greedy}}], \tau + 1)$.
  - Update lower bound $\underline{\upsilon}^{\epsilon}_{\beta}$ according to σ_{greedy} and η.

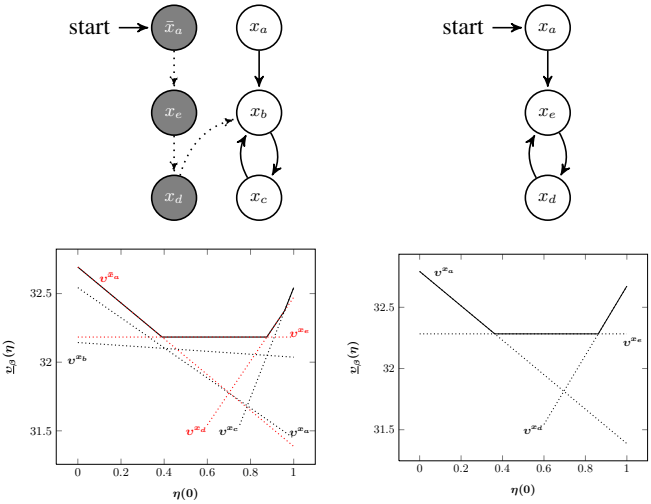

Figure 3: Example of how the lower bound function $\underline{\Lambda}$ can be improved. Vectors $\upsilon^{\bar{x}_a}$, υ^{x_d} and υ^{x_e} (and their corresponding nodes) in the left panel represent to a path through the search tree, beginning from the starting occupancy state and transitioning to υ^{x_b}. Vector υ^{x_d} becomes a new vector in $\underline{\Lambda}$. Vector υ^{x_e} pointwise dominates vector υ^{x_b} and therefore the decentralized Markovian decision rule in the policy graph and the successor vector of υ^{x_b} can be changed accordingly. Vector $\upsilon^{\bar{x}_a}$ is a duplicate of vector υ^{x_a} and causes no change. Finally, vector υ^{x_c} and associated node are pruned. The improved lower bound function and explicit Markovian policy graph are shown in the right panel.

dates, and $\underline{\Lambda}$ be the set of old vectors. We now demonstrate how a simple comparison of vectors in $\underline{\Lambda}'$ and $\underline{\Lambda}$ provides the basis for our pruning mechanism.

Following similar pruning rules as policy iteration for solving infinite-horizon POMDPs [14], we can make use of the graph structure and stationary properties of the infinite-horizon policy to improve the pruning of our lower bound (and thus the policy graph). Note that vectors υ' in $\underline{\Lambda}'$ can be duplicates of vectors υ in $\underline{\Lambda}$, that is, they can have the same decentralized Markovian decision rule and successor vector (in which case they will be pointwise equal). If they are not duplicates, they indicate how the lower bound function $\underline{\Lambda}$ can be improved – either by changing vector υ (that is, changing its corresponding decentralized Markovian decision rule $\nu(\upsilon) = \nu(\upsilon')$ and/or successor vector $\mu(\upsilon) = \mu(\upsilon')$) or by adding vector υ'. There may also be vectors that are not reachable from the vector that maximizes the starting occupancy state, those vectors can be pruned from $\underline{\Lambda}$ without affecting the lower bound value at the starting occupancy state. Thus, the lower bound function $\underline{\Lambda}$ is improved using a combination of three transformations: changing vectors, adding vectors, and pruning vectors. When a vector is changed, the values of other vectors may also change (as the graph structure has been changed). In this case, the lower bound evaluation is invoked to recompute the vector values. This is achieved by solving the following system of equations:

$$\upsilon^{x}(s) \quad = \quad r(s, \nu(x)[s]) + \beta \sum_{\bar{s} \in S} p(\bar{s}|s, \nu(x)[s]) \cdot \upsilon^{\mu(x)}(s),$$

for all vectors υ^{x} in $\underline{\Lambda}$. Markovian policy graph search is summarized in Algorithm 2 and an example on how a lower bound is improved one is presented in (fig. 3).

6. EMPIRICAL EVALUATIONS

Experiments were run on standard finite and infinite-horizon transition independent Dec-MDPs. For both cases we ran the proposed Markovian Policy Search (MPS) algorithm with the two extensions described in Section 4 and 5, referred to as Improved Markovian Policy Search (IMPS) and Markovian Policy Graph Search (MPGS), respectively. These algorithms were run 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, as suggested by Dibangoye et al. [10].

For the finite-horizon case, we compare the IMPS algorithm to the current state-of-the-art algorithm for finite-horizon transition independent Dec-MDPs, namely the Markovian Policy Search (MPS) algorithm. We do not compare against other algorithms here because Dibangoye et al. [10] demonstrate that MPS outperforms other algorithms that apply in this case (including GMAA*-ICE [25], IPG [3], and BLP [21]) by an order of magnitude in all tested benchmarks. For the infinite-horizon case, we compare to state-of-the-art algorithms for solving infinite-horizon Dec-MDPs and Dec-POMDPs: including optimal methods policy iteration (PI) [6] and incremental policy generation (IPG) [3], as well as approximate methods for computing finite-state controllers using nonlinear programming [1] and a combination of heuristic search and EM (Peri, PeriEM, EM) [20]. Note that, while PI and IPG are optimal in theory, they do not produce optimal solutions on our benchmark problems due to resources being exhausted before convergence can occur. Because the infinite-horizon approaches were developed for general Dec-POMDPs, their results are unknown on many of our domains. Note that we do not compare with ND-POMDP methods that consider locality of interaction [18]. This is mainly because our benchmarks allow all agents to interact with all teammates at all times, therefore there is no reason to expect the optimal ND-POMDP method (GOA [18]) to outperform algorithms presented in this section.

The results for finite-horizon case are in Table 1. We report the running for each algorithm to compute lower and upper bounds within ϵ of the optimal value function along with the the values for these bounds and the difference between them. In all benchmarks, the IMPS algorithm is significantly faster than the state-of-the-art MPS algorithm. In the Meeting in a 3x3 grid, for example, the IMPS algorithm is 10 times faster than the MPS algorithm. We expect that computational speedup will also allow IMPS to scale up to significantly larger problems.

For the infinite-horizon case, the results can be seen in Table 2

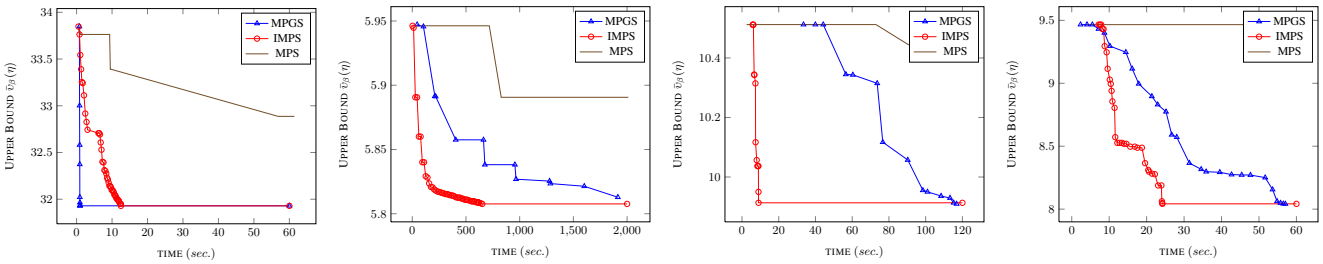

Figure 4: The convergence rates for 4 problems: Recycling robot (left), Meeting in 3x3 grid (center-left), ISR (center-right), and Pentagon (right).

Algorithm	Time	$\underline{v}^\epsilon_\beta(\eta_0)$	$\bar{v}^\epsilon_\beta(\eta_0)$	Error
Recycling robots ($\|S\| = 4,\ \|A^i\| = 3,\ \|Z^i\| = 2$)				
MPS	0.438s	308.78	308.78	0
IMPS	0s	308.78	308.78	0
Meeting in a 2x2 grid ($\|S\| = 16,\ \|A^i\| = 5,\ \|Z^i\| = 4$)				
MPS	346.9s	98.80	98.81	0.009
IMPS	80.7s	98.80	98.80	0
Meeting in a 3x3 grid ($\|S\| = 81,\ \|A^i\| = 5,\ \|Z^i\| = 9$)				
MPS	2212.7s	94.25	94.31	0.06
IMPS	208s	94.26	94.35	0.093
Meeting in a 8x8 grid ($\|S\| = 4096,\ \|A^i\| = 5,\ \|Z^i\| = 64$)				
MPS	214.73s	93.68	93.68	0
IMPS	25.4s	93.68	93.68	0
MIT($\|S\| = 8100,\ \|A^i\| = 4,\ \|Z^i\| = 90$)				
MPS	1236.11s	154.93	154.93	0
IMPS	242.04s	154.93	154.93	0
ISR ($\|S\| = 8100,\ \|A^i\| = 4,\ \|Z^i\| = 90$)				
MPS	1294.28s	182.54	182.58	0.03
IMPS	335.56s	182.54	182.54	0
PENTAGON ($\|S\| = 9801,\ \|A^i\| = 4,\ \|Z^i\| = 99$)				
MPS	1033.70s	76.39	76.48	0.09
IMPS	198.5s	76.39	76.39	0

Table 1: Results for finite-horizon decentralized MDP domains with transition independent assumptions over planning-horizon $T = 100$ with $\beta = 1.0$ and $\epsilon = 0.1$.

Algorithm	$\|\underline{\Lambda}\|$	Time	$\underline{v}^\epsilon_\beta(\eta_0)$
Recycling robots ($\|S\| = 4,\ \|A^i\| = 3,\ \|Z^i\| = 2$)			
MPGS	2	0s	31.929
MPS	109	0s	31.928
IMPS	109	0s	31.928
Mealy NLP	1	0s	31.928
Peri	6×30	77s	31.84
PeriEM	6×10	272s	31.80
EM	2	13s	31.50
IPG	4759	5918s	28.10
PI	15552	869s	27.20
Meeting in a 3x3 grid ($\|S\| = 81,\ \|A^i\| = 5,\ \|Z^i\| = 9$)			
MPGS	2	45s	5.818
MPS	88	2s	5.802
IMPS	88	2s	5.802
Peri	20×70	9714s	4.64
Meeting in a 8x8 grid ($\|S\| = 4096,\ \|A^i\| = 5,\ \|Z^i\| = 64$)			
MPGS	10	1s	5.133
MPS	67	4s	5.127
IMPS	67	4s	5.127
ISR ($\|S\| = 8100,\ \|A^i\| = 4,\ \|Z^i\| = 90$)			
MPS	102	45s	9.907
IMPS	102	6s	9.907
MPGS	66	39s	9.907
PENTAGON ($\|S\| = 9801,\ \|A^i\| = 4,\ \|Z^i\| = 99$)			
MPS	102	34s	8.031
IMPS	102	7s	8.031
MPGS	18	2s	8.031

Table 2: Results for infinite-horizon transition independent decentralized MDPs with $\beta = 0.9$. Results for Mealy NLP, EM, PeriEM, PI and IPG were likely computed on different platforms, an therefore time comparisons may be approximate at best.

and Figure 4. Table 2 describes how much time and memory (in terms of the number of nodes in the solution) it takes for each algorithm to compute a solution. For our approaches, we provide results for producing a lower bound function that is within ϵ of the best known value. This provides a basis for comparison to algorithms that do not have any theoretical guarantees. For all tested benchmarks, (I)MPS and MPGS outperform other algorithms. The MPGS algorithm produces the most concise lower bound function, but as the size of the lower bound function increases it also adds a little overhead as illustrated in the PENTAGON and ISR problems. The IMPS and MPS algorithms, however, produce good lower bound functions quite fast for all benchmarks although they require more memory to represent the lower bound function. Algorithms that quickly find a good lower bound function can nonetheless have slow convergence rates to guarantee they have found an ϵ-optimal solution.

To illustrate this point, Figure 4 describes the time MPS, IMPS and MPGS require to find an upper bound function that is within $\epsilon = 0.1$ of the optimal value function. The MPS algorithm ran out of time for all four benchmarks, and returned an upper bound function that is relatively far from the optimal value function. In contrast, the upper bound functions of IMPS and MPGS quickly converge to an ϵ-optimal value function. MPGS performs less updates and trials than the IMPS algorithm, but it often requires more computational effort at each update as illustrated for the Meeting in a 3x3 grid problem, resulting in a slower convergence rate. IMPS performs more updates and trials but at a lower price, resulting in a faster convergence rate over many problems including: Meeting in a 3x3 grid, ISR and PENTAGON.

We continue the evaluation of the proposed algorithms on randomly generated instances with multiple agents, based on the recycling robot problem, see Dibangoye et al. for a description of these scenarios. We calculate ϵ-optimal lower bound functions for 100 instances per group of N agents where N go from 5 to 14 agents, and reported the average computational time as shown in Figure 5. While MPS, IMPS and MPGS were able to compute a near-optimal value function for all groups of agents, the MPS algorithm takes many days for the larger instances. The IMPS and MPGS algorithms, however, find a near-optimal value function much faster. For example, while the IMPS and MPGS algorithms take less than a day (on average) to solve each instance of problems with 15 agents, the MPS algorithm requires almost three days (on average). Problems with up to 11 agents can also be solved in much less time.

Overall, there are many reasons that explain our results. IMPS and MPGS quickly converge to a near-optimal value function mainly because they leverage the piecewise linearity and convexity struc-

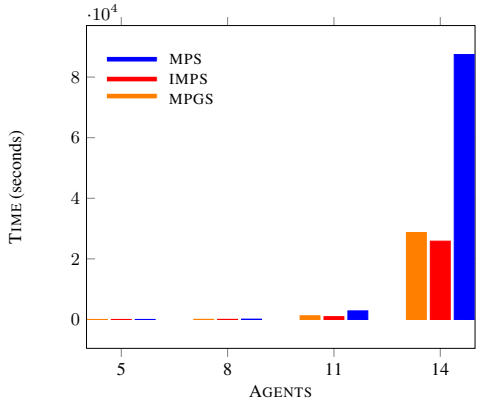

Figure 5: The MPS, IMPS and MPGS algorithm performance for increasing number of agents with $\beta = 0.9$ and $\epsilon = 0.1$.

ture of the value function. We note that IMPS often converges faster than MPGS although they both perform updates of lower and upper bounds. Because MPS optimizes a nonstationary value function, it significantly reduces the size of lower and upper bound functions at each time step. The size of upper and lower bound function at each time step increases by one in the worst case after each trial, resulting in much faster updates. MPGS, however, optimizes a stationary value function. Therefore, its upper and lower bound functions increase by one after each update. Since there may be many updates per trial, in the worst case the size of the lower and upper bound functions increase linearly with the depth of the search tree, resulting in more computationally demanding updates.

7. CONCLUSION AND DISCUSSION

In this paper, we present the first approach to solving transition independent decentralized MDPs that provides error-bounds and fast convergence rates. To accomplish this, we describe how transition independent decentralized MDPs can be transformed into continuous state MDPs with piecewise linear convex value functions. We notice that by recasting these problems as MDPs with states that consists of Dec-MDP state probability distributions, powerful approaches from POMDP literature can be applied.

While POMDP approaches can now be applied, these algorithms suffer from a major drawback. They rely on the one-step backup that requires the explicit enumeration of all possible actions. Algorithms that perform the exhaustive enumeration for solving transition independent decentralized MDPs quickly run out of time and memory, because they need to explicitly enumerate all possible decentralized decision rules. Our proposed algorithms, however, circumvent the exhaustive enumeration by combining constraint optimization and heuristic search.

In the future, we plan to explore extending IMPS and MPGS to other classes of problems and larger teams of agents. For instance, we may be able to produce near-optimal solution to more general classes of Dec-MDPs or approximate results for Dec-POMDPs by recasting them as continuous-state MDPs. The improved scalability of IMPS and MPGS is encouraging, and we plan to pursue methods to increase this even further. In particular, we think our approach could help increase the number of agents that interact in conjunction with other assumptions in the model such as locality of interaction [18, 28] and sparse joint reward matrices (as in bilinear programming approaches [21]).

8. ACKNOWLEDGMENTS

Research supported in part by AFOSR MURI project #FA9550-091-0538.

9. REFERENCES

[1] C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems*, 21(3):293–320, 2010.

[2] C. Amato, B. Bonet, and S. Zilberstein. Finite-state controllers based on mealy machines for centralized and decentralized POMDPs. In *AAAI*, 2010.

[3] C. Amato, J. S. Dibangoye, and S. Zilberstein. Incremental policy generation for finite-horizon DEC-POMDPs. In *ICAPS*, pages 2–9, Thessaloniki, Greece, 2009.

[4] R. Aras and A. Dutech. An investigation into mathematical programming for finite horizon decentralized POMDPs. *JAIR*, 37:329–396, 2010.

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

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

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

[8] A. Boularias and B. Chaib-draa. Exact dynamic programming for decentralized POMDPs with lossless policy compression. In *ICAPS*, pages 20–27, Sydney, Australia, 2008.

[9] G. Corona and F. Charpillet. Distribution over beliefs for memory bounded Dec-POMDP planning. In *UAI*, pages 135–142, 2010.

[10] J. S. Dibangoye, C. Amato, and A. Doniec. Scaling up decentralized MDPs through heuristic search. In *UAI*, 2012.

[11] J. S. Dibangoye, A.-I. Mouaddib, and B. Chai-draa. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *AAMAS*, pages 569–576, Budapest, Hungary, 2009.

[12] J. S. Dibangoye, A.-I. Mouaddib, and B. Chaib-draa. Toward error-bounded algorithms for infinite-horizon DEC-POMDPs. In *AAMAS*, pages 947–954, 2011.

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

[14] E. A. Hansen. Solving POMDPs by searching in policy space. In *UAI*, pages 211–219, 1998.

[15] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *J. Artif. Intell. Res. (JAIR)*, 13:33–94, 2000.

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

[17] A. Kumar, S. Zilberstein, and M. Toussaint. Scalable multiagent planning using probabilistic inference. In *IJCAI*, pages 2140–2146, 2011.

[18] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *AAAI*, pages 133–139, 2005.

[19] J. Pajarinen and J. Peltonen. Efficient planning for factored infinite-horizon DEC-POMDPs. In *IJCAI*, pages 325–331, 2011.

[20] J. Pajarinen and J. Peltonen. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *NIPS*, pages 2636–2644, 2011.

[21] M. Petrik and S. Zilberstein. A bilinear programming approach for multiagent planning. *JAIR*, 35:235–274, 2009.

[22] S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[23] R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

[24] T. Smith and R. G. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*, pages 542–547, 2005.

[25] M. T. J. Spaan, F. A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *IJCAI*, pages 2027–2032, 2011.

[26] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized pomdps. In *UAI*, pages 576–590, 2005.

[27] P. Velagapudi, P. Varakantham, K. Sycara, and P. Scerri. Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents. In *AAMAS*, pages 955–962, 2011.

[28] S. J. Witwicki and E. H. Durfee. Towards a unifying characterization for quantifying weak coupling in Dec-POMDPs. In *AAMAS*, pages 29–36, 2011.

[29] F. Wu, S. Zilberstein, and X. Chen. Point-based policy generation for decentralized POMDPs. In *AAMAS*, pages 1307–1314, 2010.

[30] F. Wu, S. Zilberstein, and X. Chen. Trial-based dynamic programming for multi-agent planning. In *AAAI*, 2010.