Producing efficient error-bounded solutions for transition independent decentralized MDPs
Jilles Steeve Dibangoye, Christopher Amato, Arnaud Doniec, François Charpillet

To cite this version:
Jilles Steeve Dibangoye, Christopher Amato, Arnaud Doniec, François Charpillet. Producing efficient error-bounded solutions for transition independent decentralized MDPs. International conference on Autonomous Agents and Multi-Agent Systems, May 2013, Saint Paul, MN, United States. pp.539-546. hal-00918066

HAL Id: hal-00918066
https://hal.science/hal-00918066
Submitted on 12 Dec 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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
1.2.11 [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, 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, 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
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 infinite-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 its 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 grid 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 probability** $p: S \times A \times S \to [0, 1]$, which denotes the probability $p(s'|s, a)$ of transitioning from state $s = (z^1, \ldots, z^N)$ to state $s' = (z'^1, \ldots, z'^N)$ when taking joint action $a = (a^1, \ldots, a^N)$.

- **A reward function** $r: S \times A \to \mathbb{R}$, 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** $\eta_0$, and $\beta$, 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 $p^i: Z^i \times A^i \times Z^i \to [0, 1]$, such that

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

where $s = (z^1, z^2, \ldots, z^N)$ 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 $\tau$, called decentralized decision rules $\sigma_\tau$. Note that the policies are decentralized in the sense that action choices depend only on local information.

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

**History-dependent decision rule** $\sigma^i_\tau$ is a mapping from local action-observation histories $h^i_\tau = \langle a^i_0, z^i_1, \ldots, a^i_{\tau-1}, z^i_\tau \rangle$ to local actions $\sigma^i_\tau(h^i_\tau) = a^i_\tau$. 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** $\sigma^i_\tau$, however, maps local observations $z^i_\tau$ to local actions $\sigma^i_\tau(z^i_\tau) = a^i_\tau$. 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. [19] demonstrated that there always exists a Markovian policy that achieves performance at least as good as any history-dependent policy for
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; \(\Sigma\) denotes the set of decision rules; \(\nu: X \rightarrow \Sigma\) describes the mapping that determines the decision rule \(\nu(x)\) to be executed when the graph is in node \(x\); mapping \(\mu: X \rightarrow X\) 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 the transitions do not depend on the observation seen.

The **occupancy state** is also a useful concept. The occupancy state \(\eta_\tau\), where \(\forall s: \eta_\tau(s) \equiv P(s_\tau = s | \eta_0, \sigma_0, \ldots, \sigma_{\tau-1})\), represents a probability distribution over states given an initial state distribution \(\eta_0\) and a joint policy prior to step \(\tau\). The occupancy state can be updated at each step to incorporate the latest decentralized decision rule. That is, \(\eta_\tau(s) = \sum_{s \in S} P(s_{\tau-1}(s) | \eta_\tau(s)) \cdot \eta_{\tau-1}(s)\). We also denote \(\Delta\) 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 \((\sigma_0, \ldots, \sigma_{\tau-1})\).

Also, note the difference between occupancy states and belief states. Formally, a belief state \(b_\tau\), where \(b_\tau(s) \equiv P(s|h_\tau, \eta_0)\), denotes the probability that the system is in state \(s\) if the system’s history is \(h_\tau\), starting in state probability distribution \(\eta_0\). The total probability property provides, \(\eta_\tau(s) = \sum_{s \in S} P(s|h_\tau, \eta_0) \cdot P(h_\tau|\eta_0)\). In words, this equation states that at the \(\tau\)-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, 10, 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 \(\pi \equiv (\sigma_0, \sigma_1, \ldots, \sigma_{T-1})\), this representation of the value function stores the expected value of taking that policy starting in occupancy state \(\eta_0\): \(v_{\beta, \tau}(\pi) \equiv E(\sum_{\tau=0}^{T-1} \beta^\tau \cdot \rho(\eta_\tau, \sigma_\tau) | \eta_0, \pi)\), where

**Algorithm 1: Markovian Policy Search**

```markdown
mps begin
Initialize the bounds \(\bar{\gamma}_\beta\) and \(\bar{\delta}_\beta\).
while \(\bar{\gamma}_\beta, \bar{\delta}_\beta(\eta_0) - \bar{\delta}_\beta(\eta_0) > \epsilon\) do
  \(\text{mp-trial}(\eta_0)\)
mps-trial(\(\eta_\tau\)) begin
if \(\bar{\gamma}_\beta, \bar{\delta}_\beta(\eta_\tau) - \bar{\delta}_\beta, \bar{\delta}_\beta(\eta_\tau) > \epsilon\) then
  Select \(\sigma_{\text{greedy}}\) according to \(v_{\beta, \tau}\) and \(\eta_\tau\).
  Update upper bound value function.
Call mp-trial\((\chi[\eta_\tau, \sigma_{\text{greedy}}])\).
Update lower bound value function.
```

the expected immediate reward for taking decentralized Markovian decision rule \(\sigma_\tau\) in occupancy state \(\eta_\tau\) 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 \((v_{\beta, 0}, v_{\beta, 1}, \ldots, v_{\beta, T-1})\) of mappings \(v_{\beta, \tau}\), \(\Delta \mapsto \mathbb{R}\) 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 [10]. Given this representation of the value function, a corresponding joint policy \(\pi = (\sigma_0, \sigma_1, \ldots, \sigma_{T-1})\) is extracted using one-step lookahead: \(\sigma_{\tau} = \arg \max_{\sigma_{\tau}} \rho(\eta_\tau, \sigma_{\tau}) + \beta \cdot v_{\beta, \tau+1}(\chi[\eta_{\tau+1}, \sigma_{\tau+1}])\), where \(\chi[\eta_{\tau+1}, \sigma_{\tau+1}]\) denotes the occupancy state that results from taking decentralized Markovian decision rule \(\sigma_{\tau+1}\) in occupancy state \(\eta_{\tau+1}\). 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 \(\tau = 0, 1, \ldots, T - 1\),

\[
\begin{align*}
  v_{\beta, \tau}(\eta_{\tau+1}) &= \max_{\sigma_{\tau}} \rho(\eta_{\tau+1}, \sigma_{\tau}) + \beta \cdot v_{\beta, \tau+1}(\chi[\eta_{\tau+1}, \sigma_{\tau+1}]), \\
  v_{\beta, 0}(\eta_{\tau+1}) &= 0.
\end{align*}
\]

In words, (eqn. [1]) means that the value at occupancy state \(\eta_{\tau+1}\) is given by the immediate reward for taking the best decentralized Markovian decision rule for \(\eta_{\tau+1}\) plus the discounted expected value of the resulting occupancy state \(\chi[\eta_{\tau+1}, \sigma_{\tau+1}]\). 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 \(\eta_0\). 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
tree. The error bound \((\bar{v}_{\beta,r}(\eta_r) - v_{\beta,r}(\eta_r))\) can be made arbitrarily small by expanding the search tree far enough and an \(\epsilon\)-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 this relationship, notice that standard approach to solving a POMDP is to cast 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 \(v_{\beta,\pi}^\dagger: \Delta \times \Sigma \rightarrow \mathbb{R}\) be the exact value function over planning-horizon \(T\), such that value \(v_{\beta,\pi}^\dagger(\eta, \tau) = v_{\beta,\pi}^\dagger(\eta)\) depends on the current occupancy state \(\eta\) and horizon \(\tau\), where \(v_{\beta,\pi}^\dagger(\eta, \tau) = 0\) for \(\tau \geq T\). We are now ready to present our main theoretical result.

**Theorem 4.2.** The value function \(v_{\beta,\pi}^\dagger: \Delta \times \Sigma \rightarrow \mathbb{R}\) solution of the system of Equations (7) is piecewise linear and convex.

**Proof.** We proceed by induction. The \((T - 1)\)-th value function \(v_{\beta,\pi,T-1}\) is piecewise linear and convex since \(v_{\beta,\pi,T-1}\) is a linear combination of rewards \(r(\cdot, \sigma)\) for any decentralized Markovian decision rule \(\pi\). Suppose \(v_{\beta,\pi,T}\): \(\Delta \rightarrow \mathbb{R}\) is piecewise linear and convex. In demonstrating this property for \(v_{\beta,\pi}\): \(\Delta \rightarrow \mathbb{R}\) defined for any occupancy state \(\eta_T\) by \(v_{\beta,\pi}(\eta_T) = \max_{\pi_R} \phi(\eta_T, \sigma) + \beta \cdot v_{\beta,\pi,T+1}(\chi(\eta_T, \sigma))\), we first note that the transition function \(\chi(\cdot, \cdot): \Delta \rightarrow \Delta\) 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 \(v_{\beta,\pi}^\dagger\) can be represented by a sequence of finite sets of \(|\Sigma|\)-dimensional vectors of real numbers \(\langle \Lambda_0, \Lambda_1, \cdots, \Lambda_{T-1} \rangle\), such that the value of each occupancy state \(\eta_T\) is given by:

\[
\begin{align*}
\bar{v}_{\beta,r}(\eta_T) &= \max_{\pi_R} \sum_{s \in S} \eta_T(s) \cdot v(s). 
\end{align*}
\]

(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 \(\bar{v}_{\beta,r}\) and \(\underline{v}_{\beta,r}\) to exploit the piecewise linearity and convexity property of the exact \(\tau\)-th value function \(v_{\beta,r}\). 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, \( \Psi = \{ (\eta^k, \phi^k) | k = 0, 1, \ldots \} \), for the upper bound function \( \bar{v}_{\xi} \). \( \Delta \times \mathbb{N} \rightarrow \mathbb{R} \). The upper-bound value \( \bar{v}_{\xi}(\tau) = \phi_{\tau} \) at any unvisited occupancy state \( \eta \) is thus the projection of \( \eta \) onto the convex hull formed by fixed set of occupancy states and their corresponding values in \( \Psi \). This demonstrates the ability to generalize the upper-bound function \( \bar{v}_{\xi} \) 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 \( \bar{v}_{\bar{\beta},r} \), we assume full observability and calculate the \( r \)-th optimal value function of the underlying MDP. This provides the initial upper-bound values at corners of simplex \( \Delta \), which form the initial point set \( \Psi \).

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 \(|\Psi|\) be the number of points in the point set that represents value function \( \bar{v}_{\bar{\beta},r} \). The complexity \( \mathcal{O}(|\Psi||S|) \) of each evaluation increases with increasing points in \( \Psi \). Note that evaluations occur multiple times during a trial, the importance of pruning away unnecessary points is clear. In practice, many points \( (\eta, \phi) \) in point set \( \Psi \) 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 \( (\eta, \phi) \) is said to be dominated if the remaining points in conjunction with the corner points satisfy inequality \( \bar{v}_{\bar{\beta},r}(\tau) \leq \phi_{\tau} \).

4.2 The Lower-Bound Function

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

Many vectors in sets \( \Lambda_{0}, \Lambda_{1}, \ldots, \Lambda_{T-1} \) 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 \( \Lambda_{\tau} \) corresponds to the choice of a decentralized Markovian decision rule, and the choice of a successor vector in \( \Delta_{\tau+1} \). To describe the correspondence between vectors and one-step policy choices, we introduce the following notation. For each vector \( v_{\tau} \in \Lambda_{\tau} \), let \( \tau(x) \) denote the choice of a decentralized Markovian decision rule, and let \( \mu(x) \) denote the successor vector in \( \Delta_{\tau+1} \). 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. Thus, we prove that an optimal joint policy 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 \( \underline{v}_{\xi} \).

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 \( v_{\xi} \) over a finite planning-horizon \( T \) which achieves a total ε-discounted rewards within \( \epsilon \) of an optimal value function over an infinite-horizon. To this end, we chose \( T \) such that the regret \( \Delta(\epsilon) \) of operating only over \( T = \log_{\beta} \left( \frac{1}{1-\beta} \right) \) steps instead of an infinite number of steps is upper-bounded by \( \epsilon \):
Algorithm 2: Markovian Policy Graph Search

```plaintext
mpgs begin
  Initialize the bounds \( \varphi_0^L \) and \( \varphi_0^U \).
  while \( \varphi_{s_0}^L(\eta_0) - \varphi_{s_0}^U(\eta_0) > \epsilon \) do
    mpgs-trial(\( \eta_0 \))
    Prune lower bound \( \varphi_{s_0}^L \) and corresponding policy graph
    Prune upper bound \( \varphi_{s_0}^U \)
    Evaluate lower bound \( \varphi_{s_0}^L \) and corresponding policy graph
  end
mpgs-trial(\( \eta, \tau \)) begin
  if \( \varphi_{\tau}^L(\eta) - \varphi_{\tau}^U(\eta) > \beta^{-\tau} \) then
    Select \( \sigma_{\text{greedy}} \) according to \( \varphi_{\tau}^L \) and \( \eta \) (adding a new node to the policy graph).
    Update upper bound value function \( \varphi_{\tau}^U \).
    Call mpgs-trial(\( \chi(\eta, \sigma_{\text{greedy}}), \tau + 1 \)).
    Update lower bound \( \varphi_{\tau}^L \) according to \( \sigma_{\text{greedy}} \) and \( \eta \).
end
```

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

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 Sections 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 2.4 GHz Dual-Core Intel and 2 GB 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 (IP) 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 [19]) 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 \( \epsilon \) 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.
contrast, the upper bound functions of IMPS and MPGS quickly
out of time for all four benchmarks, and returned an upper bound
\( \epsilon \) and MPGS require to find an upper bound function that is within
less have slow convergence rates to guarantee they have found an
upper bound function. Although 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. Algorithm to compute a solution. For our approaches, we provide
terms of the number of nodes in the solution) it takes for each al-
gorithm to compute a solution. For our approaches, we provide
terms of the number of nodes in the solution) it takes for each al-

and Figure 4 describes how much time and memory (in
terms of the number of nodes in the solution) it takes for each al-
gorithm to compute a solution. For our approaches, we provide
results for producing a lower bound function that is within \( \epsilon \) of the best known value. This provides a basis for comparison to algo-
rithms that do not have any theoretical guarantees. For all tested
benchmarks, IMPS 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 prob-
lems. 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. Algo-
rithms that quickly find a good lower bound function can nonethe-
less have slow convergence rates to guarantee they have found an
\( \epsilon \)-optimal solution.

To illustrate this point. Figure 4 describes the time, 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 \( \epsilon \)-optimal value function. MPGS performs less up-
dates 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

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

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>( \psi^*_\beta(\eta_0) )</th>
<th>( \psi^*_\beta(\eta_0) )</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recycling robots</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS</td>
<td>0.438s</td>
<td>308.78</td>
<td>308.78</td>
<td>0</td>
</tr>
<tr>
<td>IMPS</td>
<td>0s</td>
<td>308.78</td>
<td>308.78</td>
<td>0</td>
</tr>
<tr>
<td>Meeting in a 2x2 grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS</td>
<td>346.5s</td>
<td>98.80</td>
<td>98.81</td>
<td>0.099</td>
</tr>
<tr>
<td>IMPS</td>
<td>80.7s</td>
<td>98.80</td>
<td>98.80</td>
<td>0</td>
</tr>
<tr>
<td>Meeting in a 3x3 grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS</td>
<td>2212.7s</td>
<td>94.25</td>
<td>94.31</td>
<td>0.06</td>
</tr>
<tr>
<td>IMPS</td>
<td>208s</td>
<td>94.26</td>
<td>94.35</td>
<td>0.093</td>
</tr>
<tr>
<td>Meeting in a 8x8 grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS</td>
<td>214.7s</td>
<td>93.68</td>
<td>93.68</td>
<td>0</td>
</tr>
<tr>
<td>IMPS</td>
<td>25.4s</td>
<td>93.68</td>
<td>93.68</td>
<td>0</td>
</tr>
<tr>
<td>MIT (</td>
<td>S</td>
<td>= 8100,</td>
<td>A</td>
<td>= 4,</td>
</tr>
<tr>
<td>MPS</td>
<td>1329.1s</td>
<td>154.94</td>
<td>154.93</td>
<td>0</td>
</tr>
<tr>
<td>IMPS</td>
<td>242.0s</td>
<td>154.93</td>
<td>154.93</td>
<td>0</td>
</tr>
<tr>
<td>ISR (</td>
<td>S</td>
<td>= 8100,</td>
<td>A</td>
<td>= 4,</td>
</tr>
<tr>
<td>MPS</td>
<td>1294.28s</td>
<td>182.54</td>
<td>182.58</td>
<td>0.03</td>
</tr>
<tr>
<td>IMPS</td>
<td>335.56s</td>
<td>182.54</td>
<td>182.54</td>
<td>0</td>
</tr>
<tr>
<td>PENTAGON (</td>
<td>S</td>
<td>= 9801,</td>
<td>A</td>
<td>= 4,</td>
</tr>
<tr>
<td>MPS</td>
<td>1033.70s</td>
<td>76.39</td>
<td>76.48</td>
<td>0.09</td>
</tr>
<tr>
<td>IMPS</td>
<td>198.3s</td>
<td>76.39</td>
<td>76.39</td>
<td>0</td>
</tr>
</tbody>
</table>

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.

| Algorithm | \(|\mathcal{A}|\) | Time | \( \psi^*_\beta(\eta_0) \) |
|-----------|----------------|------|--------------------------|
| MPGS      | 2             | 0s   | 31.929                   |
| MPS       | 109           | 0s   | 31.928                   |
| IMPS      | 109           | 0s   | 31.928                   |
| Mealy NLP | 1             | 0s   | 31.928                   |
| Peri      | 6\times10    | 77s  | 31.84                    |
| PeriEM    | 6\times10    | 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| = 5, |Z'| = 9) |  |                          |                          |       |
| MPS       | 2             | 45s  | 5.818                     |
| MPGS      | 88           | 2s   | 5.802                    |
| MPS       | 88           | 2s   | 5.802                    |
| Peri      | 20\times70   | 9714s| 4.64                     |
| Meeting in a 8x8 grid (|S| = 4096, |A| = 5, |Z'| = 64) |  |                          |                          |       |
| MPS       | 10           | 1s   | 5.133                    |
| MPGS      | 67           | 4s   | 5.127                    |
| MPS       | 67           | 4s   | 5.127                    |
| Peri      | 20\times70   | 9714s| 4.64                     |
| ISR (|S| = 8100, |A| = 4, |Z'| = 90) |  |                          |                          |       |
| MPS       | 102          | 45s  | 9.907                    |
| IMPS      | 102          | 6s   | 9.907                    |
| MPGS      | 66           | 39s  | 9.907                    |
| PENTAGON (|S| = 9801, |A| = 4, |Z'| = 99) |  |                          |                          |       |
| MPS       | 102          | 24s  | 8.031                    |
| IMPS      | 102          | 7s   | 8.031                    |
| MPGS      | 18           | 2s   | 8.031                    |

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 ran-
domly generated instances with multiple agents, based on the recy-
cling robot problem, see Dibangoye et al. for a description of these
scenarios. We calculate \( \epsilon \)-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 MPGS algorithm
takes many days for the larger instances. The IMPS and MPS al-

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-
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 MPGS 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 functions 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-

9. REFERENCES