Transition-Independent Decentralized Markov Decision Processes

Raphen Becker, Shlomo Zilberstein, Victor Lesser, Claudia V. Goldman
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{raphen,shlomo,lesser,clag}@cs.umass.edu

ABSTRACT

There has been substantial progress with formal models for sequential decision making by individual agents using the Markov decision process (MDP). However, similar treatment of multi-agent systems is lacking. A recent complexity result, showing that solving decentralized MDPs is NEXP-hard, provides a partial explanation. To overcome this complexity barrier, we identify a general class of transition-independent decentralized MDPs that is widely applicable. The class consists of independent collaborating agents that are tied together through a global reward function that depends upon both of their histories. We present a novel algorithm for solving this class of problems and examine its properties. The result is the first effective technique to solve optimally a class of decentralized MDPs. This lays the foundation for further work in this area on both exact and approximate solutions.

Categories and Subject Descriptors

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

General Terms

Algorithms, Theory

Keywords

decentralized MDP, decision-theoretic planning

1. INTRODUCTION

There has been a growing interest in recent years in formal models to control collaborative multi-agent systems. Some of these efforts have focused on extensions of the Markov decision process (MDP) to multiple agents, following substantial progress with the application of such models to problems involving single agents. Examples of these attempts include the Multi-agent Markov Decision Process (MMDP) proposed by Boutilier [2], the Partially Observable Identical Payoff Stochastic Game (POIPSG) proposed by Peshkin et al. [10], the multi-agent decision process proposed by Xuan and Lesser [14], the Communicative Multiagent Team Decision Problem (COM-MTDP) proposed by Pynadath and Tambe [11], the Decentralized Markov Decision Process (DEC-POMDP and DEC-MDP) proposed by Bernstein et al. [1], and the DEC-POMDP with Communication (Dec_POMDP_Com) proposed by Goldman and Zilberstein [5].

The MMDP model is based on full observability of the global state by each agent. Similar assumptions have been made in recent work on multi-agent reinforcement learning [4]. However, we are interested in situations in which each agent has a different partial view of the global state and together they can uniquely determine it. This observation model is characteristic of a DEC-MDP and the Xuan-Lesser framework. The other models differ in that they assume a more general form of observations where there is joint partial observability.

A recent complexity study of decentralized control shows that solving such problems is extremely difficult. The complexity of both DEC-POMDP and DEC-MDP is NEXP-hard, even when only two agents are involved [1]. This is in contrast to the best known bounds for MDPs (P-hard) and POMDPs (PSPACE-hard) [9, 7]. The few recent studies of decentralized control problems (with or without communication between the agents) confirm that solving even simple problem instances is extremely hard [11, 13].

One way to overcome this complexity barrier is to exploit the structure of the domain offered by some special classes of DEC-MDPs. We study one such class of problems in which two agents operate independently but are tied together through a reward structure that depends on both of their execution histories. The model is motivated by the problem of controlling the operation of multiple space exploration rovers, such as the ones used by NASA to explore the surface of Mars [12]. Periodically, such rovers are in communication with a ground control center. During that time, the rovers transmit the scientific data they have collected and receive a new mission for the next period. The mission involves visiting several sites at which the rovers could take pictures, conduct experiments, and collect data. Each rover must operate autonomously (including no communication between them) until communication with the control center is feasible. Because the rovers have limited resources (computing power, electricity, memory, and time) and because there is uncertainty about the amount of resources that are consumed at each site, a rover may not be able to complete all of its objectives. In previous work, we have shown how to model and solve the single rover control problem by creating a corresponding MDP [15].

When two rovers are deployed, each with its own mission, there is an important interaction between the activities they perform. Two activities may be *complementary* (e.g., taking pictures of two sides of a rock), or they may be *redundant* (e.g., taking two spectrometer readings of the same rock). Both complementary and redundant activities present a problem: the global utility function is no longer additive over the two agents. When experiments provide redundant information, there is little additional value to completing both so the global value is subadditive. When experiments are complementary, completing just one may have little value so the global value is superadditive. The problem is to find a pair of policies, one for each rover, that maximizes the value of the information received by ground control.

A simplified version of our problem has been studied by Ooi and Wornell [8], under the assumption that all agents share state information every K time steps. A dynamic programming algorithm has been developed to derive optimal policies for this case. A downside of this approach is that the state space for the dynamic programming algorithm grows *doubly exponentially* with K. The only known *tractable* algorithms for these types of problems rely on additional simplifying assumptions. One such algorithm was developed by Hsu and Marcus [6] and works under the assumption that the agents share state information every time step (although it can take one time step for the information to propagate). Approximation algorithms have also been developed for these problems, although they can at best give guarantees of local optimality. For instance, Peshkin et al. [10] studied algorithms that perform gradient descent in a space of parameterized policies.

The rest of the paper is organized as follows. Section 2 provides a formal description of the class of problems we consider. Section 3 presents the coverage set algorithm, which optimally solves this class of problems, and proves that the algorithm is both complete and optimal. Section 4 illustrates how the algorithm performs on a simple scenario modeled after the planetary rover. We conclude with a summary of the contributions of this work.

2. FORMAL PROBLEM DESCRIPTION

In this section we formalize the 2-agent control problem as a transition-independent, cooperative, decentralized decision problem. The domain involves two agents operating in a decentralized manner, choosing actions based upon their own local and incomplete view of the world. The agents are cooperative in the sense that there is one value function for the system as a whole that is being maximized[1].

Definition 1. *A factored, 2-agent, DEC-MDP is defined by a tuple $< S, A, P, R >$, where*

- *$S = S_1 \times S_2$ is a finite set of states, with a distinguished initial state (s_0^1, s_0^2). S_i indicates the set of states of agent i.*
- *$A = A_1 \times A_2$ is a finite set of actions. A_i indicates the action taken by agent i.*
- *P is a transition function. $P((s_1', s_2')|(s_1, s_2), (a_1, a_2))$ is the probability of the outcome state (s_1', s_2') when the action pair (a_1, a_2) is taken in state (s_1, s_2).*
- *R is a reward function. $R((s_1, s_2), (a_1, a_2), (s_1', s_2'))$ is the reward obtained from taking actions (a_1, a_2) from state (s_1, s_2) and transitioning to state (s_1', s_2').*

We assume that agent i observes s_i, hence the two agents have joint observability. We call components of the factored DEC-MDP that apply to just one agent *local*, for example s_i and a_i are local states and local actions for agent i. A policy for each agent, denoted by π_i, is a mapping from local states to local actions. $\pi_i(s_i)$ denotes the action taken by agent i in state s_i. A **joint policy** is a pair of policies, one for each agent.

Definition 2. *A factored, 2-agent DEC-MDP is said to be* **transition independent** *if there exist P_1 and P_2 such that*

$$P(s_1'|(s_1, s_2), (a_1, a_2), s_2') = P_1(s_1'|s_1, a_1)$$
$$P(s_2'|(s_1, s_2), (a_1, a_2), s_1') = P_2(s_2'|s_2, a_2)$$

That is, the new local state of each agent depends only on its previous local state and the action taken by that agent.

Definition 3. *A factored, 2-agent DEC-MDP is said to be* **reward independent** *if there exist R_1 and R_2 such that*

$$R((s_1, s_2), (a_1, a_2), (s_1', s_2')) = R_1(s_1, a_1, s_1') + R_2(s_2, a_2, s_2')$$

That is, the overall reward is composed of the sum of two local reward functions, each of which depends only on the local state and action of one of the agents.

Obviously, if a DEC-MDP is both transition independent and reward independent, then it can be formulated and solved as two separate MDPs. However, if it satisfies just one of the independence properties it remains a non-trivial class of DEC-MDPs. We are interested in the general class that exhibits transition independence without reward independence. The Mars rovers application is an example of such a domain. The local states capture the positions of the rovers and the available resources. The actions involve various data collecting actions at the current site or the decision to skip to the next site. The overall reward, however, depends on the value of the data collected by the two rovers in a non-additive way.

2.1 Joint Reward Structure

We now introduce further structure into the global reward function. To define it, we need to introduce the notion of an occurrence of an event during the execution of a local policy.

Definition 4. *A* **history**, *$\Phi = [s_0, a_0, s_1, a_1, ...]$ is a sequence that records all of the local states and actions for one agent, starting with the local initial state for that agent.*

Definition 5. *A* **primitive event**, *$e = (s_i, a_i, s_i')$ is a triplet that includes a state, an action, and an outcome state. An* **event** *$E = \{e_1, e_2, ..., e_m\}$ is a set of primitive events.*

[1]In this paper, we assume that the agents cannot communicate during execution episodes so they must derive an optimal joint policy that involves no exchange of information. We are also studying decentralized MDPs in which the agents can communicate at some cost[5, 13].

Definition 6. *A primitive event $e = (s_i, a_i, s'_i)$* **occurs** *in history Φ, denoted $\Phi \models e$ if the triplet (s_i, a_i, s'_i) appears as a subsequence of Φ. An event $E = \{e_1, e_2, ..., e_m\}$* **occurs** *in history Φ, denoted $\Phi \models E$ iff*

$$\exists e \in E \ : \ \Phi \models e.$$

Events are used to capture the fact that an agent accomplished some task. In some cases a single local state may be sufficient to signify the completion of a task. But because of the uncertainty in the domain and because tasks could be accomplished in many different ways, we generally need a set of primitive events to capture the completion of a task. To illustrate this in the rover example, we will define the state to be $< t, l >$ where t is the time left in the day and l is the location of the current data collection site. The actions are *skip* and *collect*. The event *took picture of site 4* would be described by the following set of primitive events:

$$\{(< t, l >, a, < t', l' >) | < t, l > = < *, 4 >,$$
$$a = collect, < t', l' > = < *, * >\}.$$

Definition 7. *A primitive event is said to be* **proper** *if it can occur at most once in any possible history of a given MDP. That is:*

$$\forall \Phi = \Phi_1 e \Phi_2 \ : \ \neg(\Phi_1 \models e) \wedge \neg(\Phi_2 \models e)$$

Definition 8. *An event $E = \{e_1, e_2, ..., e_n\}$ is said to be* **proper** *if it consists of mutually exclusive proper primitive events with respect to some given MDP. That is:*

$$\forall \Phi \ \neg \exists i \neq j \ : \ (e_i \in E \wedge e_j \in E \wedge \Phi \models e_i \wedge \Phi \models e_j)$$

We limit the discussion in this paper to proper events. They are sufficiently expressive for the rover domain and for the other applications we consider, while simplifying the discussion. Later we show how some non-proper events can be modeled in this framework.

Definition 9. *Let Φ_1 and Φ_2 be histories for two agents. A* **joint reward structure**

$$\rho = [(E_1^1, E_1^2, c_1), ..., (E_n^1, E_n^2, c_n)],$$

specifies the reward (or penalty) c_k that is added to the global value function if $\Phi_1 \models E_k^1$ and $\Phi_2 \models E_k^2$.

In this paper we focus on factored DEC-MDPs that are transition-independent and whose global reward function R is composed of two local reward functions R_1 and R_2 for each agent plus a joint reward structure ρ. This allows us to define two *underlying* MDPs, $< S_i, A_i, P_i, R_i >$ even though the problem is not reward independent.

Given a local policy, π_i, the probability that a primitive event $e = (s_i, a_i, s'_i)$ will occur during any execution of π_i, denoted $P(e|\pi_i)$, can be expressed as:

$$P(e|\pi_i) = \sum_{t=0}^{\infty} P_t(s_i|\pi_i) P(a_i|s_i, \pi_i) P_i(s'_i|s_i, a_i) \qquad (1)$$

where $P_t(s_i|\pi_i)$ is the probability of being in state s_i at time step t. $P_t(s_i|\pi_i)$ can be easily computed for a given MDP from its transition model; $P(a_i|s_i, \pi_i)$ is simply 1 if $\pi_i(s_i) = a_i$ and 0 otherwise; and $P_i(s'_i|s_i, a_i)$ is simply the transition probability. Similarly, the probability that a proper event $E = \{e_1, e_2, ..., e_m\}$ will occur is:

$$P(E|\pi_i) = \sum_{e \in E} P(e|\pi_i). \qquad (2)$$

We can sum the probabilities over the primitive events in E because they are mutually exclusive.

Definition 10. *Given a joint policy (π_1, π_2) and a joint reward structure ρ, the* **joint value** *is:*

$$JV(\rho|\pi_1, \pi_2) = \sum_{i=1}^{|\rho|} P(E_i^1|\pi_1) P(E_i^2|\pi_2) c_i$$

Definition 11. *The* **global value function** *of a transition-independent decentralized MDP with respect to a joint policy (π_1, π_2) is:*

$$GV(\pi_1, \pi_2) = V_{\pi_1}(s_0^1) + V_{\pi_2}(s_0^2) + JV(\rho|\pi_1, \pi_2)$$

where $V_{\pi_i}(s_0^i)$ is the standard value of the underlying MDP for agent i at s_0^i given policy π_i.

While $V_\pi(s)$ is generally interpreted to be the value of state s given policy π, we will sometimes refer to it as the value of π because we are only interested in the value of the initial state given π. The goal is to find a joint policy that maximizes the global value function.

Definition 12. *An* **optimal joint policy***, denoted $(\pi_1, \pi_2)^*$, is a pair of policies that maximize the global value function, that is:*

$$(\pi_1, \pi_2)^* = \text{argmax}_{\pi'_1, \pi'_2} GV(\pi'_1, \pi'_2).$$

To summarize, a problem in our transition-independent decentralized MDP framework is defined by two underlying MDPs, $< S_1, A_1, P_1, R_1 >$ and $< S_2, A_2, P_2, R_2 >$, and a joint reward structure ρ.

2.2 Expressiveness of the Model

Transition-independent DEC-MDPs with a joint reward structure may seem to represent a small set of domains, but it turns out that the class of problems we address is quite general. Many problems that do not seem to be in this class can actually be represented by adding extra information to the global state. In particular, some problems that do not naturally adhere to the mutual exclusion among primitive events can be handled in this manner. The mutual exclusion property guarantees that *at most one* primitive event within an event set can occur. We discuss below some cases violating this assumption and how they can be treated within the framework we developed.

At least one primitive event – Suppose that multiple primitive events within an event set can occur and that an additional reward is added when *at least one* of them does occur. In this case the state can be augmented with one bit, initially set to 0. When a primitive event in the set occurs, the bit is set to 1. If we redefine each primitive event to include the bit switching from 0 to 1, then the event set is now proper because the bit is toggled from 0 to 1 only on the first primitive event encountered.

All primitive events – Suppose that an event is composed of n primitive events, *all* of which must occur to trigger the extra reward. In this case, each local state must be augmented with n bits (one per primitive event), which start at 0 and are toggled to 1 when the corresponding primitive event occurs (ordering constraints among primitive events could reduce the number of bits necessary). The new event set occurs when all of the bits are 1.

Counting occurrences – Suppose that an event is based on a primitive event (or another event set) repeating at least n times. Here, the local state can be augmented with $\log n$ bits to be used as a counter. The extra reward is triggered when the desired number of occurrences is reached, at which point the counting stops.

Temporal constraints – So far we have focused on global reward structures that do not impose any temporal constraints on the agents. Some temporal constraints, like *facilitates* [3], can also be represented in this framework if time is enumerated as part of the local states. For example, suppose that event E^1 in agent 1 facilitates/hinders event E^2 in agent 2, that is, the occurrence of E^1 *before* E^2 leads to an extra reward c. To properly define ρ, we need to create new events E_i^1 and E_i^2 for all $1 \leq i \leq maxTime$, and $c_i = c$. E_i^1 represents E^1 occurring at time i and E_i^2 represents E^2 occurring after i. If both E_i^1 and E_i^2 occur, then reward c is gained because E^1 happened before E^2. A more compact representation of temporal constraints remains the subject of future research.

To summarize, there is a wide range of practical problems that can be represented within our framework. Non-temporal constraints tend to have a more natural, compact representation, but some temporal constraints can also be captured.

3. COVERAGE SET ALGORITHM

This section presents a novel algorithm to find optimal joint policies for transition-independent decentralized MDPs. This is one of the first algorithms to tractably and optimally solve a significant subclass of DEC-MDPs. Most other work on distributed problems have used approximate solutions, such as heuristic policy search and gradient descent (e.g., [10]), or assumed complete communication at every step or when the optimal action is ambiguous (e.g., [2, 13]). The former do not converge on the optimal solution and the latter are not practical when communication is not possible or very expensive.

Throughout the discussion we use i to refer to one agent and j to refer to the other agent.

The algorithm is divided up into three major parts:

1. Create augmented MDPs. An augmented MDP represents one agent's underlying MDP with an augmented reward function.

2. Find the optimal coverage set for the augmented MDPs, which is the set of all optimal policies for one agent that correspond to *any* possible policy of the other agent. As we show below, this set can be represented compactly.

3. Find for each policy in the optimal coverage set the corresponding best policy for the other agent. Return the best among this set of joint policies, which is the optimal joint policy.

Pseudo-code for the coverage set algorithm is shown in Figure 1. The main function, CSA, takes a transition-independent DEC-MDP (as described in Section 2) as input, and returns an optimal joint policy. The remaining functions are described in detail below.

```
function CSA($MDP_1$, $MDP_2$, $\rho$)
    returns the optimal joint policy
    inputs:  $MDP_1$, underlying MDP for agent 1
             $MDP_2$, underlying MDP for agent 2
             $\rho$, joint reward structure

    $optset \leftarrow$ COVERAGE-SET($MDP_1, \rho$)
    $value \leftarrow -\infty$
    $jointpolicy \leftarrow \{\}$
    /* find best joint optimal policy */
    for each $policy_1$ in $optset$
        $policy_2 \leftarrow$ SOLVE(AUGMENT($MDP_2, policy_1, \rho$))
        $v \leftarrow$ GV($\{policy_1, policy_2\}, MDP_1, MDP_2, \rho$)
        if ($v > value$)
            then $value \leftarrow v$
                 $jointpolicy \leftarrow \{policy_1, policy_2\}$
    return $jointpolicy$

function COVERAGE-SET($MDP$, $\rho$)
    returns set of all optimal policies with respect to $\rho$
    inputs:  $MDP$, arbitrary MDP
             $\rho$, joint reward structure

    $planes \leftarrow \{\}$ /* planes are equivalent to policies */
    $points \leftarrow \{\}$
    /* initialize boundaries of parameter space */
    for $n \leftarrow 1$ to $|\rho|$
        $boundaries \leftarrow boundaries \cup \{x_n = 0, x_n = 1\}$
    /* loop until no new optimal policies found */
    do
        $newplanes \leftarrow \{\}$
        $points \leftarrow$ INTERSECT($planes \cup boundaries$)
        /* get optimal plane at each point */
        for each $point$ in $points$
            $plane \leftarrow$ SOLVE(AUGMENT($MDP, point, \rho$))
            if $plane$ not in $planes$
                then $newplanes \leftarrow newplanes \cup \{plane\}$
        $planes \leftarrow planes \cup newplanes$
    while $|newplanes| > 0$
    return $planes$
```

Figure 1: Coverage Set Algorithm

3.1 Creating Augmented MDPs

The first part of the algorithm is to create the augmented MDPs, which are essentially the underlying MDPs for each agent with an augmented reward function. The new reward is calculated from the original reward, the joint reward structure and the policy of the other agent. The influence of the other agent's policy on the augmented MDP can be captured by a vector of probabilities, which is a point in the following parameter space.

Definition 13. *The* **parameter space** *is an* $|\rho|$ *dimensional space where each dimension has a range of* $[0, 1]$. *Each policy* π_j *has a corresponding point in the parameter space,* $\bar{x}_{\pi_j}$*, which measures the probabilities that each one of the events in* ρ *will occur when agent* j *follows policy* π_j*:*

$$\bar{x}_{\pi_j} = [P(E_1^j|\pi_j), P(E_2^j|\pi_j), ..., P(E_{|\rho|}^j|\pi_j)].$$

Given a point in the parameter space, $\bar{x}_{\pi_j}$, agent i can define a decision problem that accurately represents the global value instead of its local value. It can do this because both the joint reward structure and agent j's policy are fixed. This new decision problem is defined as an augmented MDP.

DEFINITION 14. *An* **augmented MDP**, *$MDP_i^{\bar{x}_{\pi_j}}$, is defined as $< S_i, A_i, P_i, R'_i, \bar{x}_{\pi_j}, \rho >$, where $\bar{x}_{\pi_j}$ is a point in the parameter space computed from the policy for agent j, ρ is the joint reward structure and R'_i is:*

$$R'_i(e) = R_i(e) + \sum_{k=1}^{|\rho|} \delta_k P(E_k^j|\pi_j)c_k,$$

$$where\ \delta_k = \begin{cases} 1 & e \in E_k^i \\ 0 & otherwise \end{cases}$$

Note that $e = (s, a, s')$ so $R(e)$ is the same as $R(s, a, s')$.

An intuitive way to think about the augmented MDPs is in terms of a credit assignment problem. The system receives an extra reward if an event occurs for both agents. However, instead of giving that reward to the system, the reward could be divided up between the agents. An augmented MDP for agent i represents giving all of the credit (extra expected reward) to agent i.

THEOREM 1. *The value of a policy π_i over $MDP_i^{\bar{x}_{\pi_j}}$ is:*

$$V_{\pi_i}^{\bar{x}_{\pi_j}}(s_0^i) = V_{\pi_i}(s_0^i) + JV(\rho|\pi_i, \pi_j).$$

PROOF. The value of an MDP given a policy can be calculated by summing over all time steps t and all events e, the probability of seeing e after exactly t steps, times the reward gained from e:

$$\begin{aligned} V_{\pi_i}^{\bar{x}_{\pi_j}}(s_0^i) &= \sum_{t=0}^{\infty}\sum_{e} P_t(e|\pi_i)R'(e) \\ &= \sum_{t=0}^{\infty}\sum_{e} P_t(e|\pi_i)\left(R_i(e) + \sum_{k=1}^{|\rho|}\delta_k P(E_k^j|\pi_j)c_k\right) \\ &= \sum_{t=0}^{\infty}\sum_{e} P_t(e|\pi_i)R_i(e) + \\ &\quad \sum_{t=0}^{\infty}\sum_{e}\sum_{k=1}^{|\rho|}\delta_k P_t(e|\pi_i)P(E_k^j|\pi_j)c_k \\ &= V_{\pi_i}(s_0^i) + \sum_{k=1}^{|\rho|} P(E_k^j|\pi_j)c_k \sum_{t=0}^{\infty}\sum_{e\in E_k^i} P_t(e|\pi_i) \\ &= V_{\pi_i}(s_0^i) + \sum_{k=1}^{|\rho|} P(E_k^j|\pi_j)P(E_k^i|\pi_i)c_k \\ &= V_{\pi_i}(s_0^i) + JV(\rho|\pi_i, \pi_j). \end{aligned}$$ □

The function AUGMENT in Figure 1 takes an MDP, a policy and a joint reward structure and returns an augmented MDP according to DEFINITION 14.

Since the joint value function has been neatly folded into the value of an augmented MDP, the global value function can be rewritten as:

$$GV(\pi_1, \pi_2) = V_{\pi_1}^{\bar{x}_{\pi_2}}(s_0^1) + V_{\pi_2}(s_0^2) = V_{\pi_1}(s_0^1) + V_{\pi_2}^{\bar{x}_{\pi_1}}(s_0^2) \quad (3)$$

From this it is easy to show that an optimal joint policy is a Nash equilibrium.

PROPOSITION 1. *An optimal joint policy $(\pi_1, \pi_2)^*$ is a Nash equilibrium over the augmented MDPs:*

$$\begin{aligned} V_{\pi_1}^{\bar{x}_{\pi_2}} &= \max_{\pi'_1} V_{\pi'_1}^{\bar{x}_{\pi_2}}(s_0^1) \\ V_{\pi_2}^{\bar{x}_{\pi_1}} &= \max_{\pi'_2} V_{\pi'_2}^{\bar{x}_{\pi_1}}(s_0^2). \end{aligned}$$

PROOF. Assume $\exists \pi'_1 \neq \pi_1 : V_{\pi'_1}^{\bar{x}_{\pi_2}}(s_0^1) > V_{\pi_1}^{\bar{x}_{\pi_2}}(s_0^1)$. From Equation 3:

$$\begin{aligned} GV(\pi'_1, \pi_2) &= V_{\pi'_1}^{\bar{x}_{\pi_2}}(s_0^1) + V_{\pi_2}(s_0^2) \\ GV(\pi_1, \pi_2) &= V_{\pi_1}^{\bar{x}_{\pi_2}}(s_0^1) + V_{\pi_2}(s_0^2) \end{aligned}$$

Therefore, $GV(\pi'_1, \pi_2) > GV(\pi_1, \pi_2)$. This contradicts $(\pi_1, \pi_2)^*$. By symmetry, we can show the same for π_2. Therefore, the optimal joint policy is a Nash equilibrium over augmented MDPs. □

This naturally suggests an iterative hill-climbing algorithm where the policy for one agent is fixed and the optimal policy for the other agent's augmented MDP is computed. Then the new policy is fixed and a new optimal policy for the first agent is computed. Repeating this process will converge upon a locally optimal solution, and with random restarts it becomes an attractive approximation algorithm.

The problem is that it provides no guarantees. No matter how long it is run, there is always the possibility that it will not find the optimal joint policy. We circumvent this problem by providing an algorithm that finds *all* of the local maxima. In the average problem this is feasible because the number of local maxima is orders of magnitude fewer than the number of policies (though certainly a problem could be crafted in which this is not the case).

3.2 Finding the Optimal Coverage Set

An augmented MDP is defined over a point in the parameter space, which is a continuous space. This means that for both agents, there are an infinite number of augmented MDPs, however, only a finite number of them are potentially meaningful: the ones where the point in parameter space corresponds to a policy of the other agent. Generally, most of the augmented MDPs have the same optimal policies, so the set of all optimal policies for all of the augmented MDPs for one agent is quite small. This set is what we call the optimal coverage set.

DEFINITION 15. *The* **optimal coverage set**, *O_i, is the set of optimal policies for $MDP_i^{\bar{x}}$ given any point in parameter space, $\bar{x}$:*

$$O_i = \{\pi_i \mid \exists \bar{x}, \pi_i = \text{argmax}_{\pi'_i} V_{\pi'_i}^{\bar{x}}(s_0^i)\}.$$

Another way to look at the optimal coverage set is to examine the geometric representation of a policy over the parameter space. The value of a policy π_i, given in THEOREM 1, is a linear equation. If $|\bar{x}_{\pi_j}| = 1$, then the value function is a line in two dimensions. When $|\bar{x}_{\pi_j}| = 2$, the value function is a plane in three dimensions. In general, $|\bar{x}_{\pi_j}| = n$ and the value function is a hyperplane in $n + 1$ dimensions.

The optimal coverage set, then, is the set of hyperplanes that are highest in the $n + 1$ dimension for all points in the parameter space (first n dimensions). Figure 2 gives an example of planes over a 2-dimensional parameter space.

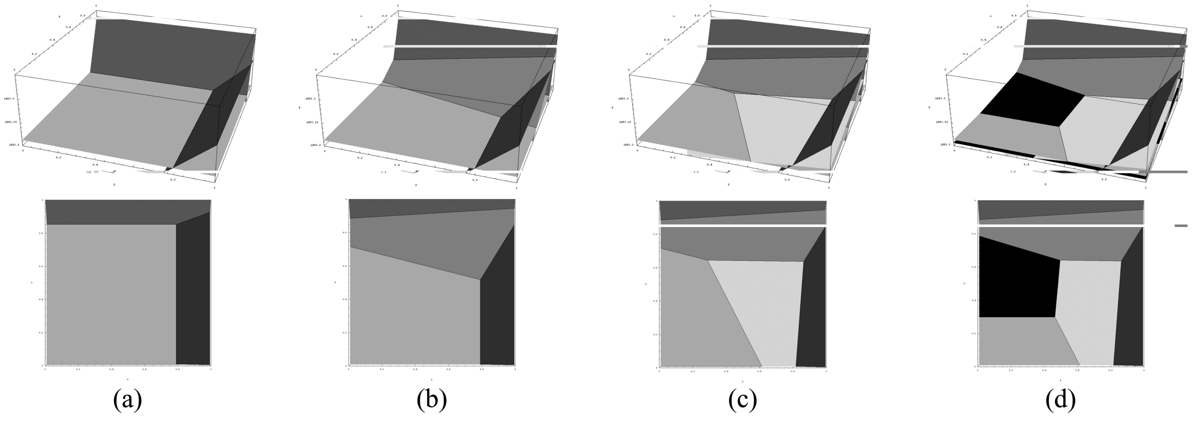

Figure 2: Intersecting Planes. (a) The first iteration checks the corners of the parameter space: $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, which yields three planes. In the second iteration four new useful points are found. The top three all have the same optimal plane, which is added in (b). The fourth point yields the plane added in (c). The next iteration produces eight new useful points, two of which result in the sixth plane, added in (d). The next iteration finds no new optimal planes and terminates, returning the set of six policies.

THEOREM 2. *If two points $\bar{x}$ and $\bar{y}$ in n-dimensional parameter space have the same corresponding optimal policy π_i, then all points on $f(\alpha) = \bar{x} + \alpha(\bar{y} - \bar{x})$, $0 \leq \alpha \leq 1$, the line segment between $\bar{x}$ and $\bar{y}$, have optimal policy π_i.*

PROOF.
Let $\pi = \text{argmax}_\pi V_\pi^{\bar{x}}(s_0) = \text{argmax}_\pi V_\pi^{\bar{y}}(s_0)$,
$\bar{z} = f(\alpha_0)$, $0 < \alpha_0 < 1$, and
$\pi' = \text{argmax}_{\pi''} V_{\pi''}^{\bar{z}}(s_0)$.
Assume $V_\pi^{\bar{z}}(s_0) < V_{\pi'}^{\bar{z}}(s_0)$. We know $V_\pi^{\bar{x}}(s_0) \geq V_{\pi'}^{\bar{x}}(s_0)$, and because $V(\cdot)$ and $f(\cdot)$ are linear functions, we can compute their value at $f(1) = \bar{y}$ by computing the unit slope.

$$V_\pi^{\bar{y}}(s_0) = \frac{V_\pi^{\bar{z}}(s_0) - V_\pi^{\bar{x}}(s_0)}{\alpha_0} + V_\pi^{\bar{x}}(s_0)$$

$$V_{\pi'}^{\bar{y}}(s_0) = \frac{V_{\pi'}^{\bar{z}}(s_0) - V_{\pi'}^{\bar{x}}(s_0)}{\alpha_0} + V_{\pi'}^{\bar{x}}(s_0)$$

$$V_\pi^{\bar{y}}(s_0) < V_{\pi'}^{\bar{y}}(s_0)$$

This contradicts that π is optimal at $\bar{y}$, therefore

$$V_\pi^{\bar{z}}(s_0) = V_{\pi'}^{\bar{z}}(s_0). \qquad \square$$

A bounded polyhedron in n dimensions is composed of a set of faces, which are bounded polyhedra in $n-1$ dimensions. The corners of a bounded polyhedron are the points (polyhedra in 0 dimensions) that the polyhedron recursively reduces to.

THEOREM 3. *Given a bounded polyhedron in n dimensions whose corners all have the same corresponding optimal policy π_i, any point on the surface or in the interior of that polyhedron also has optimal policy π_i.*

PROOF. By induction on the number of dimensions
Base case: n=1. A polyhedron in 1 dimension is a line segment with corners being the endpoints. From THEOREM 2, all points on the line have optimal policy π_i.
Inductive case: Assume true for $n-1$, show true for n. From the inductive assumption, all points in the faces of the polyhedron have optimal policy π_i. For any point within the polyhedron, a line passing through that point intersects exactly two faces. From THEOREM 2, this point has optimal policy π_i. $\square$

The part of the algorithm discussed in this section is handled by the function COVERAGE-SET in Figure 1. It takes an MDP and a joint reward structure and returns the optimal coverage set, based on THEOREM 3. To illustrate how this works, we will step through a small example.

Consider an instance of the Mars rover problem with just two elements in the joint reward structure: (E_1^1, E_1^2, c_1) and (E_2^1, E_2^2, c_2). The function CSA calls COVERAGE-SET on MDP_1 and ρ. The first thing that COVERAGE-SET does is to create the boundaries of the parameter space. These are the hyperplanes that enclose the parameter space. Since each dimension is a probability, it can range from 0 to 1, so in this case there are 4 boundary lines: $x_1 = 0$, $x_1 = 1$, $x_2 = 0$, $x_2 = 1$. The algorithm then loops until no new planes are found.

In each loop, INTERSECTer368(called)-367(on)-367(the)-368(set)-36-2
boundary and policy hyperplanes. INTERSECT takes a set of hyperplanes and returns a set of points that represent the intersections of those hyperplanes. The simple implementation would just return every intersection point, however many of those points are not useful–those that lie outside the parameter space or lie below a known optimal plane. For example, Figure 2(d) has six policy planes and the four boundaries of the parameter space. The total number of points is approximately 84, but only the 14 visible points are necessary to divide up the parameter space into the set of polygons.

After computing the set of points, the augmented MDP for each of those points is created and the optimal policy for each of those augmented MDPs
which can use standard dynamic programming algorithms.

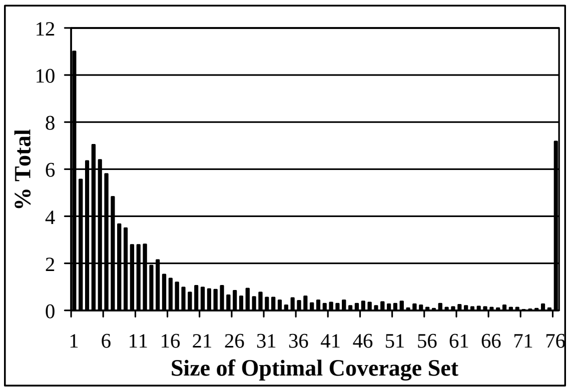

Figure 3: Distribution over the size of the optimal coverage set. The rightmost column is ≥ 76.

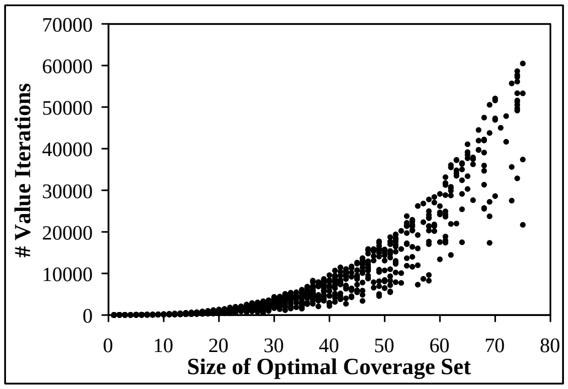

Figure 4: Number of value iterations by the size of the optimal coverage set.

to the coverage set. If a complete iteration does not find any new planes, then the loop terminates and the current coverage set is returned.

3.3 Selecting the Optimal Joint Policy

Given an optimal coverage set for agent i (considering the joint reward ρ), finding the optimal joint policy is straightforward. From PROPOSITION 1 and DEFINITION 15 we know that one of the policies in the optimal coverage set is a part of an optimal joint policy. Therefore, finding the optimal joint policy reduces to policy search through the optimal coverage set. For each policy in agent i's optimal coverage set, create the corresponding augmented MDP for agent j and find its optimal policy. The optimal joint policy is the pair with the highest global value.

The function GV returns the global value as defined in DEFINITION 11.

THEOREM 4. *The coverage set algorithm always returns the optimal value.*

PROOF. To prove that the coverage set algorithm always returns the optimal value, we show that the algorithm terminates, it finds the optimal coverage set, and then it returns the optimal joint policy.

Termination – Three of the four loops in this algorithm iterate over the elements in finite, unmodified sets. The fourth loop is the *do ... while* $|$ *newplanes* $| > 0$. In every iteration, policies for the MDP are added to *newplanes* only if they have not been added in a previous iteration. Since the set of possible policies is finite, eventually there will be no policies to add and the loop will terminate.

Optimal coverage set is found – All the planes/policies in the returned set are derived by solving the corresponding MDP using dynamic programming and are therefore optimal. All the relevant point intersections between the hyperplanes are found. This set of points divides the parameter space into a set of polyhedra. From THEOREM 3 if no new optimal policies are found from those points, then the set of optimal policies is the optimal coverage set.

The optimal joint policy is returned – The set of joint policies created by taking an optimal coverage set and finding the corresponding optimal policy for the other agent includes all Nash equilibria. The algorithm returns the best of those equilibria, which from PROPOSITION 1 is the optimal joint policy. □

4. EXPERIMENTAL RESULTS

We have implemented a version of this algorithm that works on problems with a 2-dimensional parameter space, and we have tested it on a number of randomly generated problems. The test instances were modeled after the Mars rover example used throughout this paper. There are two rovers, each with an ordered set of sites to visit and collect data. The state for each rover is composed of their current task, and the current time left in the day. The action for each state is to *skip* the current site or to *collect* data at the current site. If the rover chooses *skip*, then it moves on to the next site without spending any time. There is a distribution over time to collect data, but it always succeeds unless the time in the day runs out.

The problem instances generated had a total time of 15 hours and 6 sites for a total of 90 states for each rover. The reward for skipping a task was 0, and for collecting at a site was randomly chosen from a uniform distribution between 0.1 and 1.0. The time to collect was a Gaussian distribution with a mean between 4.0 and 6.0 and a variance 40% of the mean. Two of the sites (3 and 4) had a reward that was superadditive in the global value. c_1 and c_2 were randomly chosen from a uniform distribution between 0.05 and 0.5.

Data was collected from 4208 random problems. About 11% of the underlying MDPs were trivial, that is, the optimal coverage set only included one policy. At the other extreme, about 7% of the underlying MDPs were hard, and had a large number of policies in the optimal coverage set. In our tests, we considered an optimal coverage set of size ≥ 76 to be hard. The average size (not including the hard problems) was 13.3. The full distribution can be seen in Figure 3.

Figure 4 demonstrates how this algorithm scales with the size of the optimal coverage set. The average number of value iterations was 1963. The number of reachable states in this problem ranged from 70 to 75. To solve this using brute force policy search would result in at least 2^{70} policy evaluations. While policy evaluation is usually cheaper than value iteration, it is still clearly infeasible for these problems.

To collect this data, we used a brute force intersection algorithm that found all of the intersection points and weeded out only those that were out of bounds. It did not check to see whether the points were below another already known plane. We also cached the results of each run of value iteration and never ran it twice on the same point. We are working on a more efficient implementation of the function INTERSECT that will not need to intersect all planes to find the interesting points. This will lead to both a faster INTERSECT function, and fewer runs of value iteration.

5. CONCLUSION

The framework of decentralized MDPs has been proposed to model cooperative multi-agent systems in which agents receive only partial information about the world. Computing the optimal solution to the general class is NEXP-hard, and the only known algorithm is brute force policy search. We have identified an interesting subset of problems called transition-independent DEC-MDPs, and designed and implemented an algorithm that returns the optimal joint policy for these problems.

Besides being the first optimal algorithm to solve any nontrivial class of DEC-MDPs, the new algorithm can help establish a baseline for evaluating approximate solutions. Using the exact algorithm, other experiments have shown that a simple hill-climbing algorithm with random restarts performs quite well. In many cases, it finds the optimal joint policy very quickly, which we would not have known without the coverage set algorithm to identify it.

The new algorithm performed well on randomly generated problems within a simple experimental testbed. A more comprehensive evaluation is the focus of future work. This will include a formal analysis of the algorithm's running time, and testing the algorithm with more complex problem instances. This algorithm is also naturally an anytime algorithm, because the COVERAGE-SET function could terminate at any time and return a subset of the optimal coverage set. We will explore this more in future work. We also plan to explore the range of problems that can be modeled within this framework. For example, one problem that seems to fit the framework involves a pair of agents acting with some shared resources. If together they use more than the total available amount of one of the resources, they incur a penalty representing the cost of acquiring additional units of that resource.

Finally, the optimal coverage set is an efficient representation of the changes in the environment that would cause an agent to adopt a different policy. This information could be extremely useful in deciding when and what to communicate or negotiate with the other agents. In future work, we will explore ways to use this representation in order to develop communication protocols that are sensitive to the cost of communication.

6. ACKNOWLEDGMENTS

We thank Daniel Bernstein for useful feedback on earlier versions of this paper. This work was supported in part by NASA under grant NCC 2-1311 and by the National Science Foundation under grant IIS-0219606. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of NASA or NSF.

7. REFERENCES

[1] D.S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819-840, November 2002.

[2] C. Boutilier. Sequential optimality and coordination in multiagent systems. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 478–485, Stockholm, Sweden, 1999.

[3] K. Decker, V. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behaviour.*, Volume 2, pp. 215-234. January, 1993.

[4] M. Ghavamzadeh and S. Mahadevan. A multiagent reinforcement learning algorithm by dynamically merging Markov decision processes. *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, 2002.

[5] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. To appear in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, Melbourne, Australia, 2003.

[6] K. Hsu and S.I. Marcus. Decentralized control of finite state Markov processes. *IEEE Transactions on Automatic Control*, 27(2):426–431, 1982.

[7] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, 47(4):681–720, 2000.

[8] J.M. Ooi and G.W. Wornell. Decentralized control of a multiple access broadcast channel: Performance bounds. *Proceedings of the 35th Conference on Decision and Control*, 293–298, 1996.

[9] C.H. Papadimitriou and J. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[10] L. Peshkin, K.-E. Kim, N. Meuleau, and L.P. Kaelbling. Learning to cooperate via policy search. *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence*, 489–496, 2000.

[11] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 389–423, 2002.

[12] R. Washington, K. Golden, J. Bresina, D.E. Smith, C. Anderson, and T. Smith. Autonomous rovers for Mars exploration. *Proceedings of the IEEE Aerospace Conference*, 1999.

[13] P. Xuan and V. Lesser. Multi-agent polices: From centralized ones to decentralized ones. *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, 2002.

[14] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 616-623, Montreal, Canada, 2001.

[15] S. Zilberstein, R. Washington, D. S. Bernstein, and A. I. Mouaddib. Decision-Theoretic Control of Planetary Rovers. In M. Beetz et al. (Eds.): Plan-Based control of Robotic Agents, *LNAI*, No. 2466, 270–289, 2002.