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 up by a global reward function that depends on 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.

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 Communicative Multiagent Team Decision Problem (COM-MTDP) proposed by Pynadath and Tambe [11], and the decentralized Markov decision process (DEC-POMDP and DEC-MDP) proposed by Bernstein, Givan, Immerman and Zilberstein [1]. 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 [3]. We are interested in situations in which each agent might have a different partial view of the global state. This is captured by the DEC-POMDP model, in which a single global Markov process is controlled by multiple agents, each of which receives partial information about the global state after each action is taken. A DEC-MDP is a DEC-POMDP with the restriction that at each time step the agents' observations together uniquely determine the global state.

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, 6]. 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, 14].

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 by some 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 [13]. Periodically, such rovers are in communication with a ground control center. During that time, the rovers transmit the scientific data they had 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 its mission entirely. 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 important interaction between the activities they perform. Two activities may be *complementary* (e.g., taking pictures of the 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

*Student author

Submitted to the Second International joint Conference on Autonomous Agents and Multi-Agent Systems.

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.

We develop in this paper a general framework to handle such problems. Pynadath and Tambe [11] studied the performance of some existing teamwork theories from a decision-theoretic perspective, similar to the one we develop here. However, our technique exploits the structure of the problem to find the optimal solution more efficiently.

A simplified version of our problem has been studied by Ooi and Wornell [7], 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 even more assumptions. One such algorithm was developed by Hsu and Marcus [5] 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 rovers 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').

[1]In this paper, we assume that the agents cannot communicate between execution episodes, so they need to 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[4].

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*

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

That is, the new local state of each agent depends only on the 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 Structures

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 the local states and actions for one of the agents, starting with the local initial state of that agent.*

DEFINITION 5. *A* **primitive event**, $e = (s, a, s')$ *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.*

DEFINITION 6. *A primitive event* $e = (s, a, s')$ **occurs** *in history* Φ, *denoted* $\Phi \models e$ *if the triplet* (s, a, s') *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:

$$\{(< t, l >, a, < t', l' >) | < t, l >=< *, 4 >, \\ a = collect, < t', l' >=< *, 5 >\}..$$

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 of two transition-independent MDPs. 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 $\Phi_2 \models E_k^2$.

We limit the discussion in this paper to 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 refer to the *underlying MDP*, $< S_i, A_i, P_i, R_i >$ even though the problem is not reward independent.

Given a local policy, π, the probability that a primitive event $e = (s, a, s')$ will occur during any execution of π, denoted $P(e|\pi)$ can be expressed as follows.

$$P(e|\pi) = \sum_{t=0}^{\infty} P_t(s|\pi) P(a|s, \pi) P(s'|s, a) \qquad (1)$$

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

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

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:*

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

DEFINITION 11. *A* **global value function** *of a transition-independent decentralized MDP with respect to a joint policy (π_1, π_2), local reward functions R_1, R_2 and a joint reward structure ρ 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:*

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

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 exactly 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 the 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 per event that is initially 0. When a primitive event in the set occurs, the bit is set to 1. If we redefine each primitive event so that the corresponding bit switches from 0 to 1 when it occurs, we make the event set proper because the bit can switch from 0 to 1 only once in every possible history.

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 bit for each primitive event (ordering constraints among the primitive events could be exploited to reduce the number of bits necessary). The new set of events that describe the activity are those that flip the last bit from 0 to 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 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 event E_2 in agent 2, that is, the occurrence of E_1 *before* E_2 leads to some extra reward when E_2 occurs. If we include time in local states, then we could include in the joint reward structure a triplet for every possible combination of primitive events that satisfy the temporal constraint. Obviously, this leads to creating large numbers of event sets. 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

In this section, we develop a general algorithm for solving transition-independent, factored, 2-agent DEC-MDPs with a global value function shown in DEFINITION 11. The algorithm returns the optimal joint policy.

So far, no optimal algorithm has been presented for solving the general class of DEC-MDPs, short of complete enumeration and evaluation of policies. Some search algorithms in policy space and gradient decent techniques have been used to find approximate solutions, with no guarantee of convergence in the limit on the optimal joint policy (e.g., [10]). Here we present a novel algorithm that utilizes the structure of the problem to find the optimal joint policy. (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 factored DEC-MDP and a joint reward structure as described in Section 2, and returns an optimal joint policy.

3.1 Creating Augmented MDPs

The first step in the algorithm is to create the augmented MDPs, which are derived from 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.

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

    optset ← COVERAGE-SET(MDP_1, ρ)
    value ← −∞
    jointpolicy ← {}
    /* find best joint optimal policy */
    for each policy_1 in optset
        policy_2 ← SOLVE(AUGMENT(MDP_2, policy_1, ρ))
        v ← GV({policy_1, policy_2}, MDP_1, MDP_2, ρ)
        if (v > value)
            then value ← v
            else jointpolicy ← {policy_1, policy_2}
    return jointpolicy

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

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

Figure 1: Coverage Set Algorithm

DEFINITION 13. *The* **parameter space** *is an n 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 n events will occur when agent j follows policy π_j:*

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

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. The new decision problem is defined by an augmented MDP.

DEFINITION 14. *An* **augmented MDP**, *denoted $MDP_i^{\bar{x}_{\pi_j}}$, is defined as $< S_i, A_i, P_i, R_i', s_0^i, \bar{x}_{\pi_j}, \rho >$, where $\bar{x}_{\pi_j}$ is a point in 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 for $e = (s, a, s')$, $R(e)$ is the same as $R(s, a, s')$.

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.

The global value is equal to the local value of a policy for one agent plus the value of the corresponding augmented MDP for the other agent. This allows an augmented MDP to be used in an iterative, hill-climbing algorithm. In such an algorithm, the policy for agent j, π_j, is held fixed and the optimal corresponding policy for agent i is found by finding the optimal policy for $MDP_i^{\bar{x}\pi_j}$. Then the roles are reversed, and agent i's new policy is held fixed while the optimal corresponding policy for agent j is found. This process repeats until neither policy changes, i.e., until a Nash equilibrium is reached.

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 DEFINITION 11 and THEOREM 1:

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

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. □

While the hill-climbing algorithm reaches a Nash equilibrium, and the optimal joint policy is a Nash equilibrium, there is no guarantee that they are the same equilibrium. In many problems there are many local optima, with the optimal joint policy being the best of them. However, the hill-climbing algorithm, even with random restarts, does not guarantee finding the global maxima.

3.2 Finding the Optimal Coverage Set

An augmented MDP is defined over the parameter space, which is a continuous space. This means that for both agents, there is an infinite number of augmented MDPs, however, only a finite number of them are potentially useful: the ones where the point in parameter space corresponds to a policy of the other agent. This set is still quite large since the number of policies is exponential in the number of states. It turns out that in practice, only a very small number of these augmented MDPs are really interesting.

Given any particular augmented MDP, the only policy that is useful is the policy that maximizes the expected value of that MDP. Fortunately, most of the augmented MDPs have the *same optimal policy*, and the set of interesting augmented MDPs can be reduced to one per optimal policy. This set of optimal policies, the optimal coverage set, is what we are really after and the augmented MDPs provide a way to find them.

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.

THEOREM 2. *If two points $\bar{x}$ and $\bar{y}$ in n-dimensional parameter space have the same corresponding optimal policy π, 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.*

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.

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

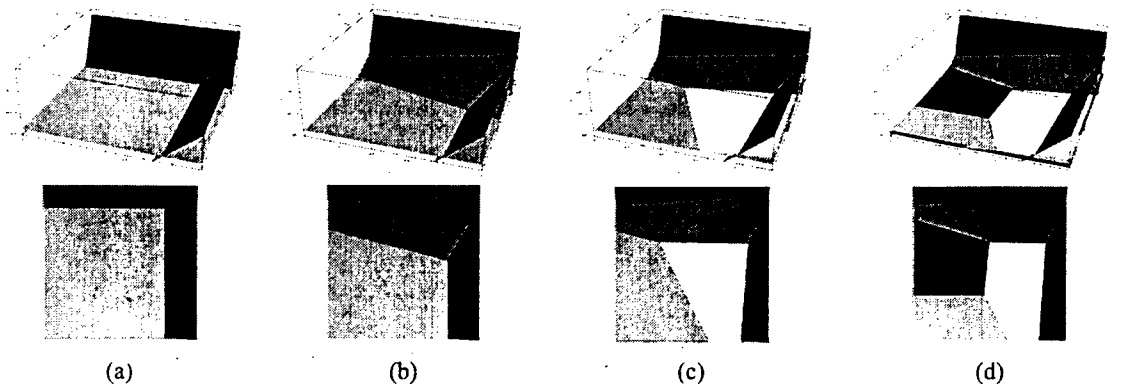

Figure 2: Intersecting Planes. (a) The first iteration checks the corners of the parameter space: $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, which yields three planes. In the second iteration four new interesting points are found. The top three all have the same optimal plane, which is added in (b). The fourth point yields the plane added in (c). The next iteration produces eight new interesting 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.

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

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

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

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

□

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, any point on the surface or in the interior of that polyhedron also has optimal policy π.*

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.
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. For any point within the polyhedron, any line passing through that point intersects two faces, with the interior point between the intersection points. From THEOREM 2, this point has optimal policy π. □

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 starts by calling 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, INTERSECT is called on the set of known policy hyperplanes and boundary hyperplanes. INTERSECT takes a set of hyperplanes and returns a set of points that represent the intersections of those hyperplanes. Linear programming would be an efficient way to compute these intersections because the majority of them are not *interesting*: they lie outside the parameter space or lie below another 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 120, but only 14 points are visible in the top view, and they are the only interesting points because they form the corners of the polyhedra (or polygons in this example) over the parameter space whose interior all have the same optimal policy (THEOREM 3).

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 is computed by SOLVE, which can use standard dynamic programming algorithms. The value of a policy and a point in parameter space is

$$\begin{aligned} V_{\pi_1}^{\bar{x}\pi_2}(s_0^1) &= P(E_1^1|\pi_1)P(E_1^2|\pi_2)c_1 + \\ &\quad P(E_2^1|\pi_1)P(E_2^2|\pi_2)c_2 + V_{\pi_1}(s_0^1). \end{aligned}$$

For a given π_1, the value function is a plane over the parameter space. The plane for the new optimal policies will either be equivalent (different policy but same value) or equal to a plane already in the coverage set, or it will be better than every other plane in the coverage set at this point in parameter space. If it is the latter case, this new plane is added 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 the optimal coverage set for agent i, finding the op-

timal joint policy is easy. From PROPOSITION 1, an optimal joint policy includes an optimal policy for agent i given some augmented MDP. Since the optimal coverage set includes all of the optimal policies it includes one that is a part of an optimal joint policy. To find the optimal joint policy the algorithm finds the corresponding optimal policy for agent j for every policy in the optimal coverage set. It does this by creating an augmented MDP for agent j for each policy in the optimal coverage set and then finding the optimal policy of that MDP. The optimal joint policy is the resulting joint policy with the highest global value.

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

THEOREM 4. *The coverage set algorithm is complete and optimal.*

PROOF. (sketch) To prove that the coverage set algorithm is complete and optimal, we must prove three claims: the algorithm terminates, it finds the optimal coverage set, and it returns the optimal joint policy. We avoid discussion of multiple equivalent policies for clarity.

Termination – There are four loops in this algorithm. The first is over a set of policies, which is finite. The second is over the number of dimensions in the parameter space, which is finite. The third is looping until no new policies are found. The new policies are added to the coverage set each iteration, so a particular policy could never be a new policy twice. Since the set of policies is finite, this loop will terminate. The fourth loop is over a set of points, which are a subset of intersection points of a finite set of planes, which is finite. All of the loops in this algorithm halt, therefore the algorithm halts.

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. From PROPOSITION 1, the best of those equilibria is the joint optimal 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 wasting 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

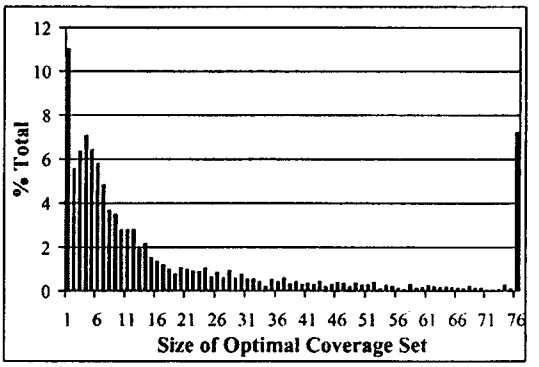

Figure 3: Distribution over the size of the optimal coverage set. The rightmost bar is ≥ 76.

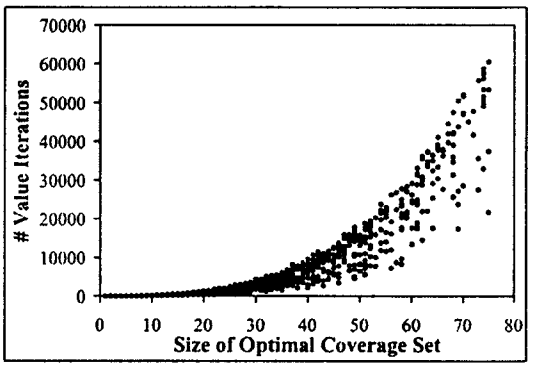

Figure 4: Number of value iterations by the size of the optimal coverage set.

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. A problem is composed of two underlying MDPs, and the complexity of a problem is really only as hard as the easiest of the two, which means that about 21% of the *problems* were trivial and less than 1% were hard. 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 been 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 non-trivial class of DEC-MDPs, the new algorithm can help establish a baseline for evaluating fast 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 identifying the optimal solution using the coverage set algorithm.

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 grants NAG-2-1394 and NAG-2-1463 and by the National Science Foundation under grant IIS-9907331. Any opinions, findings, and 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. To appear in *Mathematics of Operations Research*.

[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] 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.

[4] C. V. Goldman and S. Zilberstein. Optimizing Information Exchange in Cooperative Multi-agent Systems. Submitted for publication, 2002.

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

[6] 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.

[7] 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.

[8] C.H. Papadimitriou and J. Tsitsiklis. On the complexity of designing distributed protocols. *Information and Control*, 53:211–218, 1982.

[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. *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] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. *Twelfth National Conference on Artificial Intelligence*, 426–431, 1994.

[13] 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.

[14] 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.

[15] S. Zilberstein, R. Washington, D.S. Bernstein, and A.I. Mouaddib. Decision-theoretic control of planetary rovers. To appear in Springer Lecture Notes in AI, December 2002.