Journal of Artificial Intelligence Research 22 (2004) 423–455 Submitted 05/04; published 12/04

Solving Transition Independent Decentralized Markov Decision Processes

Raphen Becker RAPHEN@CS.UMASS.EDU
Shlomo Zilberstein SHLOMO@CS.UMASS.EDU
Victor Lesser LESSER@CS.UMASS.EDU
Claudia V. Goldman CLAG@CS.UMASS.EDU
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003

Abstract

Formal treatment of collaborative multi-agent systems has been lagging behind the rapid progress in sequential decision making by individual agents. Recent work in the area of decentralized Markov Decision Processes (MDPs) has contributed to closing this gap, but the computational complexity of these models remains a serious obstacle. To overcome this complexity barrier, we identify a specific class of decentralized MDPs in which the agents' transitions are independent. The class consists of independent collaborating agents that are tied together through a structured global reward function that depends on all of their histories of states and actions. We present a novel algorithm for solving this class of problems and examine its properties, both as an optimal algorithm and as an anytime algorithm. To the best of our knowledge, this is the first algorithm to optimally solve a non-trivial subclass of decentralized MDPs. It lays the foundation for further work in this area on both exact and approximate algorithms.

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. One example of this is the Multi-agent Markov Decision Process (MMDP) proposed by Boutilier (1999).

The MMDP is a straightforward extension of the MDP to multiple agents by factoring the action space into actions for each of the agents. The advantage of the centralized approach is that it is easier to solve optimally than a general decentralized approach. Other researchers have also used this view of multi-agent systems. Guestrin, Venkataraman, and Koller (2002) focused on interactions between agents through the reward function, similar to what we do here, however their observation model more resembles the full observability of the MMDP than the partial observability of our model. Similarly, Ghavamzadeh and Mahadevan (2002) developed a reinforcement learning algorithm for the fully observed case. In contrast, we are interested in situations in which each agent has a partial and different view of the global state. We are also working with offline planning assuming that the model is known, not reinforcement learning.

The disadvantage of the MMDP and related approaches is that it makes an important assumption that renders it inappropriate for many multi-agent systems. This is that each agent has the same (complete) world view. Many multi-agent systems have each of the agents observing a different part of the environment. It is possible to map this into an MMDP by assuming that each agent communicates its observation with every other agent at every step with zero cost. Xuan and Lesser (2002) take this a step further by having the agents communicate only when there is ambiguity. They also introduce transformations of the policy that address the tradeoff between the expected amount of communication and the expected value of the final solution.

Ooi and Wornell (1996) studied a variation where all of the agents shared their state information every K time steps. They developed a dynamic programming algorithm 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. A more tractable algorithm was developed by Hsu and Marcus (1982) under the assumption that the agents share state information every time step (although it can take a time step for the information to propagate). The problem we examine, however, assumes that communication has a very high cost or is not possible.

There are other models that are more of an extension of Partially Observable MDPs (POMDPs) to multi-agent systems. These models are characterized by each agent only having a partial and different view of the world state. This includes the Partially Observable Identical Payoff Stochastic Game (POIPSG) proposed by Peshkin, Kim, Meuleau, and Kaelbling (2000), the Communicative Multiagent Team Decision Problem (COM-MTDP) proposed by Pynadath and Tambe (2002), the Decentralized Markov Decision Process (DEC-POMDP and DEC-MDP) proposed by Bernstein, Givan, Immerman, and Zilberstein (2002), and the DEC-POMDP with Communication (Dec-POMDP-Com) proposed by Goldman and Zilberstein (2003, 2004).

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-complete, even when only two agents are involved (Bernstein et al., 2002). This is in contrast to the best known bounds for MDPs (P-complete) and POMDPs (PSPACE-complete) (Papadimitriou & Tsitsiklis, 1987; Mundhenk, Goldsmith, Lusena, & Allender, 2000). 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 (Pynadath & Tambe, 2002; Xuan & Lesser, 2002). However, there are certain goal-oriented DEC-MDPs for which there are optimal, polynomial algorithms (Goldman & Zilberstein, 2004). For the general DEC-POMDP, the only known optimal algorithm is a new dynamic programming algorithm developed by Hansen, Bernstein, and Zilberstein (2004).

Xuan, Lesser, and Zilberstein (2001) use a model very similar to the one proposed in this paper with their silent commitment heuristic, in that each of the agents has its own local problem they are solving. The primary difference is their additional assumption that the local subproblems are independent given an easily computable optimal goal for the agents to share. Our approach computes the optimal goal as part of an optimal joint policy.

Wolpert, Wheeler, and Tumer (1999) and Tumer, Agogino, and Wolpert (2002) also study coordination in a decision-theoretic framework with their work on Collective Intelligence. While the algorithm presented here uses a reward function that fits their subworldfactored criteria, we use it in a novel way offline to find a globally optimal solution while they focus on reinforcement learning online to find high quality but non-optimal solutions.

Other research has approached this complexity barrier through approximations of the general problem (Goldman & Zilberstein, 2003; Nair, Tambe, Yokoo, Pynadath, & Marsella, 2003; Peshkin et al., 2000; Shen, Lesser, & Carver, 2003). However, the approach taken in this paper is two-fold. First, we exploit the structure of the domain offered by some special classes of DEC-MDPs to reduce the complexity of the model. Then we present a novel algorithm that can solve many large[1] problems in a reasonable amount of time. We also examine the anytime characteristics of this algorithm and show that it performs well as an approximation algorithm that converges on the optimal solution.

1.1 Problem Description

The class of problems studied in this paper is characterized by two or more cooperative agents solving (mostly) independent local problems. The actions taken by one agent can not affect any other agents' observation or local state. Consequently, an agent can not observe the other agents' states and actions and can not communicate with them. The interaction between the agents happens through a global value function that is not simply a sum of the values obtained through each of the agents' local problems. The non-linear rewards combined with the decentralized view of the agents make the problem more difficult to solve than the MMDP, while the independence of the local problems make it easier to solve than the general DEC-MDP. We call this class of problems Transition Independent DEC-MDPs and we formally define it in Section 2.

We are looking at problems where the value of an activity performed by one agent may depend on the activities of other agents. One example is information collection, which includes taking a picture, performing an experiment and performing complex calculations on data. The information collected may be *complementary* (e.g., different pictures of the same area can assist in building a 3D model), or they may be *redundant* (e.g., computing the same result wasting valuable time). Both complementary and redundant information present a problem: the global utility function is no longer additive over the 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. We are generally not looking at temporal constraints between agents or the information they collect, however, some temporal constraints can be captured by our model (see Section 2.2.3).

We motivate this class of problems with two examples. The first example is the problem of controlling the operation of multiple planetary exploration rovers, such as the ones used by NASA to explore the surface of Mars (Washington, Golden, Bresina, Smith, Anderson, & Smith, 1999). Periodically, the 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. Each rover has its own area to explore, and a mission consists of a set of sites at which a rover could take pictures, conduct experiments, and in general collect data. Some of these sites are overlapping sites (lie on the border between two or more rovers' area) and multiple rovers can visit them.

1. Large compared to what state-of-the-art algorithms for DEC-MDPs are able to solve.

The rovers must operate autonomously (without 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. The overall goal is to maximize the value of the information received by ground control. However, this is not the same as maximizing the value of the information collected by each rover individually because two rovers performing experiments at overlapping sites produce redundant information. In previous work, we have shown how to model and solve the single rover control problem by creating a corresponding MDP (Zilberstein, Washington, Bernstein, & Mouaddib, 2002).

Our second motivating example is UAVs (unmanned aerial vehicles) performing reconnaissance in a combat environment. Here, communication between the UAVs is too costly because it would reveal their location to the enemy. The UAVs are sent out to collect information from a number of locations and then to return to safety. The information sought by a UAV may depend on other information that it has collected. For example, if it discovers something interesting at one location it may spend additional time there collecting more detailed information and choose to skip other locations.

Independence between the local problems holds true in many domains, and in others it serves as an approximation. In the Mars rover example the rovers are operating in distinctly separate areas. For example, NASA's Mars Exploration Rover mission landed Spirit and Opportunity on separate sides of the planet[2]. An "overlapping site" could consist of similar geographic structures that lie on opposite sides of the planet. Given the rovers distance from each other there is no possibility of interaction of the local decision problems other than through the value of the information they collect, nor is direct communication feasible. The rover example used throughout this paper is an abstraction of this idea.

The UAV example illustrates a different type of situation where the local problems are independent except for the reward. Here the agents are highly mobile, but the level of abstraction used to represent this planning problem allows them to coexist in the same airspace. Essentially, one square of the grid world represents a mile of airspace so there is no problem with many UAVs traveling through it simultaneously, and the actions of one agent will not interfere with those of another agent.

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 solves this class of problems, and proves that this algorithm is both complete and optimal. Section 4 illustrates how the algorithm performs on a simple scenario modeled after the planetary rover example. We conclude with a summary of the contributions of this work.

2. Formal Problem Description

In this section we formalize the n-agent control problem as a transition independent, cooperative, decentralized decision problem. This formalism is an extension of previous work (Becker, Zilberstein, Lesser, & Goldman, 2003) to n agents. The domain involves any number of 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

2. http://marsrovers.jpl.nasa.gov/home/

one value function for the system as a whole that is being maximized. However, the agents can only communicate during the planning stage, not during the execution episodes, so they must derive an optimal joint policy that involves no exchange of information. Goldman and Zilberstein (2004), and Xuan and Lesser (2002) study decentralized MDPs in which the agents can communicate during execution.

Definition 1 *An n-agent* **DEC-MDP** *is defined by a tuple $\langle S, A, P, R, \Omega, O\rangle$, where*

- *S is a finite set of world states, with a distinguished initial state s^0.*
- *$A = A_1 \times A_2 \times \ldots \times A_n$ is a finite set of joint actions. A_i indicates the set of actions that can be taken by agent i.*
- *$P : S \times A \times S \to \Re$ is the transition function. $P(s'|s, (a_1 ... a_n))$ is the probability of the outcome state s' when the joint action $(a_1 ... a_n)$ is taken in state s.*
- *$R : S \times A \times S \to \Re$ is the reward function. $R(s, (a_1 ... a_n), s')$ is the reward obtained from taking joint action $(a_1 ... a_n)$ in state s and transitioning to state s'.*
- *$\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$ is a finite set of joint observations. Ω_i is the set of observations for agent i.*
- *$O : S \times A \times S \times \Omega \to \Re$ is the observation function. $O(s, (a_1 ... a_n), s', (o_1 ... o_n))$ is the probability of agents 1 through n seeing observations o_1 through o_n (agent i sees o_i) after the sequence s, $(a_1 ... a_n)$, s' occurs.*
- *Joint full observability: the n-tuple of observations made by the agents together fully determine the current state. If $O(s, (a_1 ... a_n), s', (o_1 ... o_n)) > 0$ then $P(s'|(o_1 ... o_n)) = 1$.*

Definition 2 *A* **factored***, n-agent DEC-MDP is a DEC-MDP such that the world state can be factored into $n + 1$ components, $S = S_0 \times S_1 \times ... \times S_n$.*

Factoring the state space of a DEC-MDP could be done in many ways. The intention of such a factorization is a separation of features of the world state that belong to one agent from those of the others and from the external features. This separation is strict, meaning that no feature of the world state may belong to more than one group. S_0 refers to external features, which are parts of the state that the agents may observe and be affected by but do not affect themselves, such as weather or time. S_i refers to the set of state features for agent i, which is the part of the world state that an agent observes and affects. This will be made clearer by the following definitions.

We refer to $\hat{s}_i \in S_i \times S_0$ as the *local* state, $a_i \in A_i$ as the local action, and $o_i \in \Omega_i$ as the local observation for agent i. Since the agent is affected by the external features its local state must include them. Just as in a DEC-MDP, a local policy (or just policy) for one agent is a mapping from sequences of observations to local actions (we will simplify this later). A **joint policy**, $(\pi_1 ... \pi_n)$, is a set of policies, one for each agent.

Let us illustrate this with the rover exploration problem introduced previously. There are two rovers, each exploring adjacent areas. The global state is composed of each rovers' location, the information collected so far (including the agent that collected it), and the time left in the day. This state can be factored into a local state for each agent containing that rover's location, the information collected by that rover, and the time left in the day. The action each of the rovers can take is a movement action or an information collection action.

Definition 3 *A factored, n-agent DEC-MDP is said to be* **transition independent** *if there exist P_0 through P_n such that*

$$P(s'_i|(s_0 \ldots s_n),(a_1 \ldots a_n),(s'_0 \ldots s'_{i-1},s'_{i+1} \ldots s'_n)) = \begin{cases} P_0(s'_0|s_0) & i=0 \\ P_i(s'_i|\hat{s}_i,a_i,s'_0) & 1 \leq i \leq n \end{cases}$$

That is, the new local state of each agent depends only on its previous local state, the action taken by that agent, and the current external features. The external features change based only on the previous external features. This implies that $P((s'_0 \ldots s'_n)|(s_0 \ldots s_n),(a_1 \ldots a_n)) = \prod_{i=0}^{n} P_i(\cdot)$.

Our example problem exhibits transition independence because the rovers have no interaction through their state transitions other than time (remember that we allow multiple rovers to collect information from the same site simultaneously without conflicting). Rover 1's actions affect its location and the information it has collected, but not any other agents' location or information collection.

Definition 4 *A factored, n-agent DEC-MDP is said to be* **observation independent** *if there exist O_1 through O_n such that $\forall o_i \in \Omega_i$*

$$P(o_i|(s_0 \ldots s_n),(a_1 \ldots a_n),(s'_0 \ldots s'_n),(o_1 \ldots o_{i-1},o_{i+1} \ldots o_n)) = P(o_i|\hat{s}_i,a_i,\hat{s}'_i)$$

That is, the observation an agent sees depends only on that agent's current and next local state and current action.

Definition 5 *A factored, n-agent DEC-MDP is said to be* **locally fully observable** *if $\forall o_i \; \exists \hat{s}_i \; : P(\hat{s}_i|o_i) = 1$.*

That is, each agent fully observes its own local state at each step. While local full observability and observation independence are related, it is possible for one to be true without the other. However, when both are true then Ω and O in the definition of a factored n-agent DEC-MDP are redundant and can be removed.

In our rover exploration example, we define the observations of each rover to be its own local state. When a rover takes an action it fully observes the outcome, which is that rover's next local state. By definition, therefore, the problem is locally fully observable. It is also observation independent because each rover only observes its own local state and sees nothing related to any other rover. An action by agent i is not allowed to change the global state in such a way that agent j could detect it. For example, if rover i could leave a marker on a rock then rover j must not be able to detect it.

Definition 6 *A factored, n-agent DEC-MDP is said to be* **reward independent** *if there exist f and R_1 through R_n such that*

$$R\left((s_0 ... s_n), (a_1 ... a_n), (s'_0 ... s'_n)\right) = f\left(R_1(\hat{s}_1, a_1, \hat{s}'_1) ... R_n(\hat{s}_n, a_n, \hat{s}'_n)\right)$$

and

$$\begin{array}{rcl} R_i(\hat{s}_i, a_i, \hat{s}'_i) & \leq & R_i(\hat{s}_i, a'_i, \hat{s}''_i) \quad \Leftrightarrow \\ f\left(R_1 ... R_i(\hat{s}_i, a_i, \hat{s}'_i) ... R_n\right) & \leq & f\left(R_1 ... R_i(\hat{s}_i, a'_i, \hat{s}''_i) ... R_n\right) \end{array}$$

That is, the overall reward is composed of a function of the local reward functions, each of which depends only on the local state and local action of one of the agents. This function is such that maximizing each of the local reward functions individually maximizes the function itself. An example function is $f(R_1 ... R_n) = \sum_i R_i$.

When Definitions 2 through 6 are satisfied a DEC-MDP (NEXP-complete) decomposes into n independent local MDPs (P-complete). However, if the problem is missing just one of these independence relations then it remains a non-trivial subclass of DEC-MDPs. This paper deals with the general class of factored n-agent DEC-MDPs that exhibit transition and observation independence, and local full observability, but not reward independence. Instead, the reward function is divided into two components. The first component is a set of local reward functions that are reward independent. The second component is a reward signal the system receives (not any individual agent) that depends on the actions of multiple agents and is therefore not reward independent. It is defined by the joint reward structure and is captured in the global value function being maximized.

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 7 *A* **history** *for agent i, $\Phi_i = [\hat{s}_i^0, a_i^0, \hat{s}_i^1, a_i^1, ...]$ is a valid execution sequence that records all of the local states and actions for one agent, beginning with the local starting state for that agent, $\hat{s}_i^0$.*

Definition 8 *A* **primitive event**, *$e = (\hat{s}_i, a_i, \hat{s}_i)$ is a tuple that includes a local state, an action, and an outcome state. An* **event** *$E = \{e_1, e_2, ..., e_h\}$ is a set of primitive events.*

Definition 9 *A primitive event $e = (\hat{s}_i, a_i, \hat{s}'_i)$* **occurs** *in history Φ_i, denoted $\Phi_i \models e$ iff the tuple $(\hat{s}_i, a_i, \hat{s}'_i)$ appears as a sub-sequence of Φ_i. An event $E = \{e_1, e_2, ..., e_h\}$* **occurs** *in history Φ_i, denoted $\Phi_i \models E$ iff*

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

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.

In the rover example, the events we want to capture represent an agent collecting data at a particular site. This event will take many primitive events to describe because time is part of the local state. For example, the event *Collect Data at Site 4* can be accomplished by the primitive event where the agent executes the *collect* action at site 4 starting at time 6 (and ending at any possible next state). It can also be accomplished by a similar primitive event where the agent starts collecting at time 8 instead of time 6. More formally, the event can be described by the following set of primitive events:

$$\{\ (\langle l,t\rangle, a, \langle l',t'\rangle)\ \mid \langle l,t\rangle = \langle 4,*\rangle,\ a = \mathit{collect},\ \langle l',t'\rangle = \langle *, < t\rangle\ \},$$

where the local state is $\langle l,t\rangle$; l is the current location and t is the time left in the day.

Definition 10 *A primitive event is said to be* **proper** *if it can occur at most once in each possible history of a given MDP. That is:*

$$\forall \Phi_i = \Phi_i^1 e \Phi_i^2\ :\ \neg(\Phi_i^1 \models e) \wedge \neg(\Phi_i^2 \models e)$$

Definition 11 *An event $E = \{e_1, e_2, ..., e_h\}$ is said to be* **proper** *if it consists of mutually exclusive proper primitive events with respect to some given MDP. Mutually exclusive means:*

$$\forall \Phi_i\ \neg\exists x \neq y\ :\ e_x \in E\ \wedge\ e_y \in E\ \wedge\ \Phi_i \models e_x\ \wedge\ \Phi_i \models e_y$$

The event *Collect Data at Site 4* described earlier can easily be turned into a proper event. First, because time is part of the local state, no single primitive event could happen more than once — time only flows in one direction. Time is not a necessary condition, but in this example it suffices. Second, it is not difficult to restrict a rover from collecting data more than once at a particular site, either through additional bits in the state or through a fixed acyclic ordering for visiting the sites built into the structure of the MDP.

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.

The joint reward structure can be viewed as a list of x constraints between the agents that describe how interactions between their local policies affect the global value of the system. A particular constraint k exists between some subset of agents G_k, where $|G_k| \geq 2$. The semantics of constraint k, $(E_k^{g_{k1}}, E_k^{g_{k2}} ... E_k^{g_{k|G_k|}}, c_k)$, is that each agent involved must satisfy its part of the constraint for the constraint to be satisfied. Agent g_{ki} satisfies its part of constraint k if event $E_k^{g_{ki}}$ occurs in that agent's history $\Phi_{g_{ki}}$. If each agent in a constraint does this, then the system would receive an additional c_k reward. There is no restriction on the number of the constraints that can be satisfied.

Definition 12 *Let Φ_1 through Φ_n be histories for n agents. Let G_1 through G_x be subsets of the n agents. Each set G_k contains $|G_k|$ agents g_{k1} through $g_{k|G_k|}$. A* **joint reward structure**

$$\rho = [(E_1^{g_{1,1}}, E_1^{g_{1,2}} ... E_1^{g_{1|G_1|}}, c_1), ..., (E_x^{g_{x1}}, E_x^{g_{x2}} ... E_x^{g_{x|G_x|}}, c_x)],$$

specifies the reward (or penalty) c_k that is added to the global value function if $\Phi_{g_{k1}} \models E_k^{g_{k1}}$ and $\Phi_{g_{k2}} \models E_k^{g_{k2}}$ and ... and $\Phi_{g_{k|G_k|}} \models E_k^{g_{k|G_k|}}$.

In this paper we focus on factored DEC-MDPs that are transition independent, observation independent with full local observability and whose global reward function R is composed of a local reward function for each agent R_1 through R_n plus a joint reward structure ρ. This allows us to define n *underlying* MDPs, $\langle S_i \times S_0, A_i, P_i, R_i\rangle$ even though the problem is **not** reward independent. These processes are indeed Markov, meaning that the local policies are mappings from local states to actions instead of histories to actions as with DEC-MDPs. A local state was proved to be a sufficient statistic for the history of observations sensed by an agent controlling a decentralized MDP with independent transitions and observations (Goldman & Zilberstein, 2004). Therefore, the optimal local policy of such an agent can be expressed as a mapping from local states to actions.

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

$$P(e|\pi_i) = \phi \sum_{v=0}^{\infty} P_v(\hat{s}_i|\pi_i) P_i(\hat{s}'_i|\hat{s}_i, a_i), \;\; \phi = \begin{cases} 1 & \pi_i(\hat{s}_i) = a_i \\ 0 & otherwise \end{cases}$$

where $P_v(\hat{s}_i|\pi_i)$ is the probability of being in state $\hat{s}_i$ at time step v. $P_v(\hat{s}_i|\pi_i)$ can be easily computed for a given MDP from its transition model, and $P_i(\hat{s}'_i|\hat{s}_i, a_i)$ is simply the transition probability. ϕ sets the probability to zero if the primitive event e is inconsistent with the policy π_i. Similarly, the probability that a proper event $E = \{e_1, e_2, ..., e_m\}$ will occur is:

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

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

Definition 13 *Given a joint policy* $(\pi_1 ... \pi_n)$ *and a joint reward structure* ρ*, the* **joint value** *is defined as:*

$$JV(\rho|\pi_1 ... \pi_n) = \sum_{k=1}^{|\rho|} c_k \prod_{i=1}^{|G_k|} P(E_k^{g_{ki}}|\pi_{g_{ki}})$$

Definition 14 *The* **global value function** *of a transition independent decentralized MDP with respect to a joint policy* $(\pi_1 ... \pi_n)$ *is:*

$$GV(\pi_1 ... \pi_n) = \sum_{i=1}^{n} V_{\pi_i}(\hat{s}_i^0) + JV(\rho|\pi_1 ... \pi_n)$$

where $V_{\pi_i}(\hat{s}_i^0)$ *is the standard value of the underlying MDP for agent* i *at starting state* $\hat{s}_i^0$ *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. Similarly, the global value function GV is defined over policies and not states because the interest is in the expected value of the starting state only. The goal is to find a joint policy that maximizes the global value function.

Definition 15 *An* **optimal joint policy***, denoted* $(\pi_1 \ldots \pi_n)^*$*, is a set of local policies that maximize the global value function, that is:*

$$(\pi_1 \ldots \pi_n)^* = \text{argmax}_{\pi'_1 \ldots \pi'_n} GV(\pi'_1 \ldots \pi'_n).$$

To summarize, a problem in our transition independent decentralized MDP framework is defined by n underlying MDPs, $\langle S_1 \times S_0, A_1, P_1, R_1 \rangle$ through $\langle S_n \times S_0, A_n, P_n, R_n \rangle$, 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. The joint reward structure ρ is just another way of representing the general reward function over all of the agents, $R((s_0 \ldots s_n)(a_1 \ldots a_n)(s'_0 \ldots s'_n))$. How efficient a representation the joint reward structure is of the general reward function depends on the level of independence in the reward function. Intuitively it is similar to the difference between standard matrix representations and the representations for sparse matrices.

Often if a problem meets the independence and observability requirements but does not obviously have a properly structured reward function, it can be represented by adding extra information (in the form of a bit) to the local state or by modifying the state in some way. In many cases these bits are already necessary to represent the problem as a DEC-MDP and do not cause an increase in the size of the state space to represent in the joint reward structure. Since MDPs suffer from the curse of dimensionality the size of the state space grows exponentially with the number of bits added.

This section will discuss several properties of the joint reward structure and how problems can be modified to fit. It will also show how some temporal constraints can be represented by this class of problems.

2.2.1 Mutual Exclusion

One property that some problems do not naturally adhere to is the mutual exclusion among primitive events. The mutual exclusion property guarantees that *at most one* primitive event within an event set can occur. This section presents three alternatives. For each of them, the bits mentioned are already present in the DEC-MDP representation of the problem.

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 x primitive events, *all* of which must occur to trigger the extra reward. In this case, each local state must be augmented with x 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 the last bit still 0 is flipped.

Counting occurrences – Suppose that an event is based on a primitive event (or another event set) repeating at least x times. Here, the local state can be augmented with $\log x$ 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.

2.2.2 All Events

Another property of the joint reward structure is that every event in a constraint must occur for the system to receive the extra reward. Other useful constraints that one may wish to express may involve the occurrence of *at most x events*, *exactly x events*, or *at least x events*. These can be represented by creating new constraints, one for each possible way of solving the existing constraint. For example, suppose there is a constraint between three agents (events E_1, E_2, E_3), and the constraint is satisfied (extra reward received) if exactly two of the three events occur. This constraint can be represented by three new constraints with the semantics of *all events must occur*: $E_1E_2\neg E_3$, $E_1\neg E_2E_3$, $\neg E_1E_2E_3$.

The event $\neg E_x$ represents the original event E_x not occurring. This event can be represented by adding two additional bits to the state space. One bit represents whether the original event has occurred and the other represents whether the original event could occur in the future. The new event $\neg E_x$ occurs when the original event has not occurred and the cannot-occur-in-the-future bit flips to *true*.

There is also the possibility that the algorithm presented in Section 3 could be modified to handle these other constraints more directly and efficiently. This remains the topic of future work.

2.2.3 Temporal Constraints

So far we have focused on global reward structures that do not impose any temporal constraints on the agents. Other types of constraints are soft temporal constraints like *facilitates* and *hinders* (Decker & Lesser, 1993). A *facilitates* constraint between activities A and B means that if A is finished before B is started, then execution of B is somehow facilitated. Facilitation can take many forms like reduced consumption of resources, less time to execute, higher reward received. The *hinders* constraint is similar, but instead involves making the execution of B more costly.

Soft temporal constraints that just involve a change in the expected reward can 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^1_i and E^2_i for all $1 \leq i \leq maxTime$, and $c_i = c$. E^1_i represents E^1 occurring at time i and E^2_i represents E^2 occurring after i. If both E^1_i and E^2_i occur, then reward c is gained because E^1 happened before E^2. ρ now becomes $[(E^1_1, E^2_1, c), (E^1_2, E^2_2, c)... (E^1_{maxTime}, E^2_{maxTime}, c)]$.

There are a couple of important limitations for temporal constraints. First is that not all types of temporal constraints can be handled. The temporal constraints are limited to affecting the reward and not the duration (or any other characteristic of the facilitated/hindered event that the agent observes) because that would violate the transition independence assumption. Second, representing a temporal constraint is not very practical because one temporal constraint gets represented as $maxTime$ non-temporal constraints in

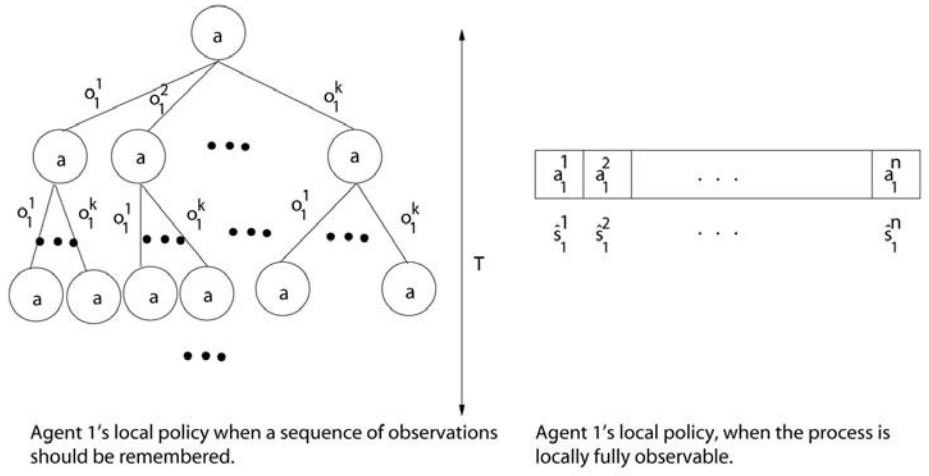

a
o_1^1
o_1^2
o_1^k
T
a_1^1 a_1^2 . . . a_1^n
$\hat{s}_1^1$ $\hat{s}_1^2$ . . . $\hat{s}_1^n$
Agent 1's local policy when a sequence of observations should be remembered.
Agent 1's local policy, when the process is locally fully observable.

Figure 1: Exponential-sized vs. Polynomial-sized Policies.

ρ. 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 have a more natural, compact representation, but some temporal constraints can also be captured.

2.3 Complexity Analysis

It is known that solving optimally a general DEC-MDP is NEXP-complete (Bernstein et al., 2002). The leftmost diagram in Figure 1 illustrates this result by showing that a local policy of a general DEC-MDP is exponential in the size of the observations. Each local policy is a mapping from sequences of observations to actions. Therefore, there are $|A_i|^{|\Omega|^T}$ policies for agent i, where T is the horizon. Each agent j needs to build a belief-state MDP for each one of agent i's local policies. The number of states in this MDP is exponential in the size of Ω_j. This results in an algorithm that will solve optimally a DEC-MDP problem using a complete search algorithm in double exponential time[3].

The class of DEC-MDPs studied in this paper is characterized by having independent transitions and observations and being locally fully observable. Because the current local view of a single agent is a sufficient statistic for the past history of observations of agent i (see Goldman & Zilberstein, 2004) a local policy for agent i is a mapping from its local states to its possible actions, i.e., $\pi_i : S_i \times S_0 \rightarrow A_i$ (see the rightmost diagram in Figure 1). This results in agents' policies being of size *polynomial* in the number of local states (as opposed to exponential in the size of the observation set as in the general case). Theorem 1

3. Assuming that the finite horizon T is similar in size to the number of global states of the system $|S|$. This assumption is necessary for the complexity of the general DEC-MDP, but not for the complexity of the class of problems dealt with in this paper.

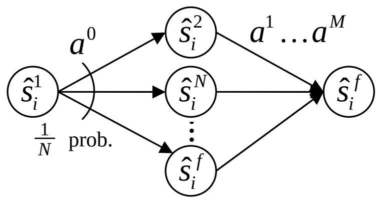


Figure 2: DTEAM reduces to an MDP of this form.

(adapted from Goldman & Zilberstein, 2004) shows that the complexity of solving optimally such a DEC-MDP is easier than solving the general case.

Theorem 1 *Deciding a DEC-MDP with independent transitions and observations, local full observability, and joint reward structure ρ is NP-complete.*

Proof. Since the current local state of agent i is a sufficient statistic, a local policy for that agent is of size polynomial in $|S_i \times S_0|$. There are $|A_i|^{|S_i \times S_0|}$ policies (mappings from $S_i \times S_0$ to A_i). The number of states in each agent i's belief-state MDP is polynomial in $|S_i \times S_0|$ (for a fixed and known policy for the other agents). Evaluating one such local policy can be done in polynomial time (by running dynamic programming on the belief-state MDP), but there are exponentially many such policies for which this should be done. Therefore, the upper bound for the decision problem stated in this theorem is NP.

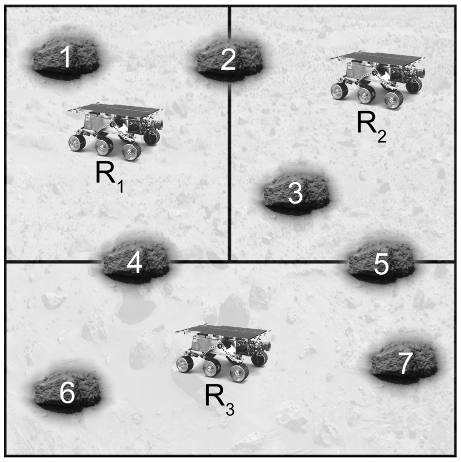
To prove the lower bound we will reduce the NP-complete problem DTEAM (Papadimitriou & Tsitsiklis, 1982, 1986) to this problem for two agents, which is sufficient for a lower bound. DTEAM is a single-step discrete team decision problem. There are two agents. Agent i, $i = 1, 2$, observes a random integer k_i, $1 \le k_i \le N$, and takes an action $\gamma_i(k_i) \in \{1, \ldots, M\}$. Their actions incur cost $c(k_1, k_2, \gamma_1(k_1), \gamma_2(k_2))$. The problem is to find policies γ_1 and γ_2 that minimize the expected cost:

$$\sum_{k_1=1}^{N} \sum_{k_2=1}^{N} c(k_1, k_2, \gamma_1(k_1), \gamma_2(k_2)).$$

The reduction is quite straightforward. The local MDP for agent i consists of an initial state ($\hat{s}_i^0$), N intermediate states ($\hat{s}_i^1 \ldots \hat{s}_i^N$), and a single final state ($\hat{s}_i^f$), see Figure 2. The observation is the current state. There is one action (a^0) available in the initial state, and M actions ($a^1 \ldots a^M$) in the intermediate states. The local rewards are always 0. The joint reward ρ includes an entry for every combination of intermediate state and action taken by the agents:

$$\begin{aligned} \rho = [\quad & (\ \{(\hat{s}_1^1, a^1, \hat{s}_1^f)\},\ \{(\hat{s}_2^1, a^1, \hat{s}_2^f)\},\ -c(1,1,1,1)\ ), \\ & (\ \{(\hat{s}_1^1, a^1, \hat{s}_1^f)\},\ \{(\hat{s}_2^1, a^2, \hat{s}_2^f)\},\ -c(1,1,1,2)\ ), \\ & \ldots \\ & (\ \{(\hat{s}_1^N, a^M, \hat{s}_1^f)\},\ \{(\hat{s}_2^N, a^M, \hat{s}_2^f)\},\ -c(N,M,N,M)\ )\ \ ] \end{aligned}$$

Finding the joint policy that maximizes the expected reward will be the same policy that minimizes the expected cost in the original problem. □



Site x	$t(x)$	$r(x)$	**Total Value**
1	10	6	$r(1)$
2	**12**	**5**	$\boldsymbol{3r(2)}$
3	8	7	$r(3)$
4	**11**	**9**	$\boldsymbol{r(4)}$
5	**13**	**8**	$\boldsymbol{1.5r(5)}$
6	9	6	$r(6)$
7	14	7	$r(7)$

Figure 3: Illustration of the Mars rover example. Each of the rovers can collect data from sites within their region. Sites 2, 4 and 5 fall on the boundaries and can be visited by two rovers each. The table entries in bold are the shared tasks.

2.4 Example

This example is an instance of the rover problem presented earlier. There are three rovers, each with their own region of space to explore, as shown in Figure 3. Each site x has a value for the data collected there, $r(x)$, and a time to collect the data, $t(x)$. The local state for rover 1 is composed of the current location l, the time left in the day t, and the data to be collected d_1, d_2, d_4 (0 if not collected, 1 if collected). The available actions are *collect* and *go to site* x. The transition on a *collect* action, when at site x, is from $\langle l = x, t, d_a, d_b, d_x = 0\rangle$ to $\langle l = x, t - t(x), d_a, d_b, d_x = 1\rangle$. The reward received is $r(x)$ if $t - t(x) \geq 0$, otherwise 0. The transition on a *go to site* x action when at site y is from $\langle l = y, t, d_1, d_2, d_4\rangle$ to $\langle l = x, t - t(y, x), d_1, d_2, d_4\rangle$ and a reward of 0. $t(y, x)$ is the time it takes to move from site y to site x. The joint reward structure is:

$$\rho = \Big[\ (E^1_2, E^2_2, r(2)),\ (E^1_4, E^3_4, -r(4)),\ (E^2_5, E^3_5, -0.5r(5))\Big].$$

Event E^i_x is an event that represents agent i collecting data at site x. This event is composed of primitive events of the form:

$$\{(\hat{s}, a, \hat{s}')|\hat{s}=\langle l=x, t, d_a, d_b, d_x=0\rangle,\ a=\mathit{collect},\ \hat{s}'=\langle l=x, t-t(x), d_a, d_b, d_x=1\rangle\}.$$

Each agent locally receives $r(x)$ when it collects at site x. The Total Value column in Figure 3 lists the total reward received by the system for each site. For sites $x = \{2, 4, 5\}$ this is the reward received after both agents collect there. The difference between the total value and $2r(x)$ is put into ρ as the extra reward the system receives for site x for those sites that overlap two agents. For example, for $x = 5$, $1.5r(5) - 2r(5) = -0.5r(5)$. When rover 1 collects data at site 5 it receives $r(5)$. When rover 2 collects data there it receives $r(5)$. The system also receives a penalty of $-0.5r(5)$ because both rovers collected that data. The net reward received by the system is $1.5r(5)$.

3. Coverage Set Algorithm

This section presents a novel algorithm to find optimal joint policies for transition independent decentralized MDPs. To the best of our knowledge this is the first algorithm to tractably and optimally solve a significant subclass of DEC-MDPs. We have also applied it to a different class of problems in which the agents were reward independent but not transition independent (Becker, Zilberstein, & Lesser, 2004), which demonstrates that it is not limited to the class of problems described in this paper. Most other work on distributed problems have used approximate solutions, such as heuristic policy search and gradient descent (e.g., Peshkin et al., 2000), or assumed complete communication at every step or when the optimal action is ambiguous (e.g., Boutilier, 1999; Xuan & Lesser, 2002). The former are not guaranteed to converge on the optimal solution and the latter are not practical when communication is not possible or very expensive.

While the formal problem description deals with n agents, for clarity the description of the algorithm will only deal with two agents (i and j). Section 3.5 discusses the n-agent extension of the algorithm.

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 4. 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.

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 16 *The* **parameter space** *is a $|\rho|$ 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:*

$$\bar{x}_{\pi_j} = [P(E^j_1|\pi_j), P(E^j_2|\pi_j), ..., P(E^j_{|\rho|}|\pi_j)].$$

```
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:   $MDP$, underlying 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 4: Coverage Set Algorithm

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 just 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 17 *An* **augmented MDP**, *$MDP_i^{\bar{x}_{\pi_j}}$, is defined as $\langle S_i \times S_0, A_i, P_i, R'_i, \bar{x}_{\pi_j}, \rho\rangle$, 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:*

$$R'_i(e) = R_i(e) + \sum_{k=1}^{|\rho|} \phi_k P(E_k^j|\pi_j)c_k,\ \phi_k = \begin{cases} 1 & e \in E_k^i \\ 0 & otherwise \end{cases}$$

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

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

$$V_{\pi_i}^{\bar{x}_{\pi_j}}(\hat{s}_i^0) = V_{\pi_i}(\hat{s}_i^0) + JV(\rho|\pi_i, \pi_j).$$

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

$$\begin{aligned}
V_{\pi_i}^{\bar{x}_{\pi_j}}(\hat{s}_i^0) &= \sum_{v=0}^{\infty}\sum_{e} P_v(e|\pi_i)R'(e) \\
&= \sum_{v=0}^{\infty}\sum_{e} P_v(e|\pi_i)\left(R_i(e) + \sum_{k=1}^{|\rho|}\phi_k P(E_k^j|\pi_j)c_k\right) \\
&= \sum_{v=0}^{\infty}\sum_{e} P_v(e|\pi_i)R_i(e) + \\
&\quad \sum_{v=0}^{\infty}\sum_{e}\sum_{k=1}^{|\rho|}\phi_k P_v(e|\pi_i)P(E_k^j|\pi_j)c_k \\
&= V_{\pi_i}(\hat{s}_i^0) + \sum_{k=1}^{|\rho|} P(E_k^j|\pi_j)c_k \sum_{v=0}^{\infty}\sum_{e\in E_k^i} P_v(e|\pi_i) \\
&= V_{\pi_i}(\hat{s}_i^0) + \sum_{k=1}^{|\rho|} P(E_k^j|\pi_j)P(E_k^i|\pi_i)c_k \\
&= V_{\pi_i}(\hat{s}_i^0) + JV(\rho|\pi_i, \pi_j).
\end{aligned}$$

□

The function AUGMENT in Figure 4 takes an MDP, a policy and a joint reward structure and returns an augmented MDP according to Definition 17.

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

$$GV(\pi_1, \pi_2) = V_{\pi_1}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) + V_{\pi_2}(\hat{s}_2^0) = V_{\pi_1}(\hat{s}_1^0) + V_{\pi_2}^{\bar{x}_{\pi_1}}(\hat{s}_2^0) \tag{1}$$

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:*

$$\begin{aligned} V_{\pi_1}^{\bar{x}_{\pi_2}} &= \max_{\pi_1'} V_{\pi_1'}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) \\ V_{\pi_2}^{\bar{x}_{\pi_1}} &= \max_{\pi_2'} V_{\pi_2'}^{\bar{x}_{\pi_1}}(\hat{s}_2^0). \end{aligned}$$

Proof. Assume $\exists \pi_1' \neq \pi_1 : V_{\pi_1'}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) > V_{\pi_1}^{\bar{x}_{\pi_2}}(\hat{s}_1^0)$.
From Equation 1:

$$\begin{aligned} GV(\pi_1', \pi_2) &= V_{\pi_1'}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) + V_{\pi_2}(\hat{s}_2^0) \\ GV(\pi_1, \pi_2) &= V_{\pi_1}^{\bar{x}_{\pi_2}}(\hat{s}_1^0) + V_{\pi_2}(\hat{s}_2^0) \end{aligned}$$

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. Nair et al. (2003), and Shen et al. (2003) study applications of this approximation algorithm to DEC-POMDPs.

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 problems we experimented with this is feasible because the number of local maxima is orders of magnitude fewer than the number of policies (though a problem could probably 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. Additionally, most of these 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.

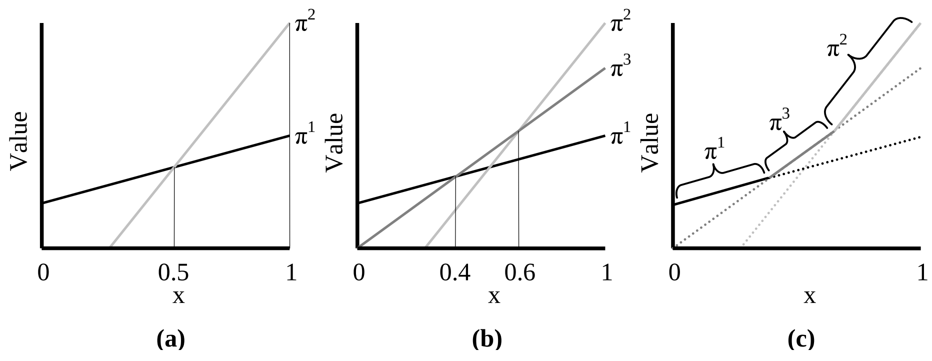


Figure 5: Search process in one dimension.

Definition 18 *The* **optimal coverage set**, *O_i, is the set of optimal policies for MDP$_i^{\bar{x}}$ given any point in parameter space, $\bar{x}$:*

$$O_i = \{\pi_i \mid \exists \bar{x}, \pi_i = \text{argmax}_{\pi'_i} V^{\bar{x}}_{\pi'_i}(\hat{s}^0_i)\}.$$

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 2, 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). The upper surface that we are interested in is a piecewise-linear and convex function formed by these hyperplanes. First we examine the one dimensional case (one constraint). Figure 5 shows a graph of the parameter space (x-axis) versus expected value of a policy. On this graph we plot the equation shown in Theorem 2 for particular policies. For every point in the parameter space, the optimal policy of the augmented MDP that corresponds to that point is a policy in the optimal coverage set.

The algorithm starts by finding the optimal policies in the corners of the parameter space, which in this example are $x = 0.0$ and $x = 1.0$. This yields two optimal policies, π^1 for $x = 0.0$ and π^2 for $x = 1.0$, whose value functions are graphed in Figure 5 (a). Those two lines intersect at $x = 0.5$, which is then chosen as the next point. The optimal policy of that point, π^3, is added in Figure 5 (b). New points in the top surface are found by intersecting the new line with the previous lines, $x = 0.4$ and $x = 0.6$. If new policies are found at either of those points, those lines are added and new intersections selected. This repeats until all of the intersection points have optimal policies represented in the graph. That set of policies form the optimal coverage set. In the example no new policies were found at $x = 0.4$ or $x = 0.6$ so Figure 5 (c) shows the piecewise-linear and convex surface we are searching for, and those three policies form the optimal coverage set for one agent.

The intersection points of the lines are key because that is where the optimal policy changes from one policy to a different policy. Intuitively, if a policy is optimal at $x = 0.6$

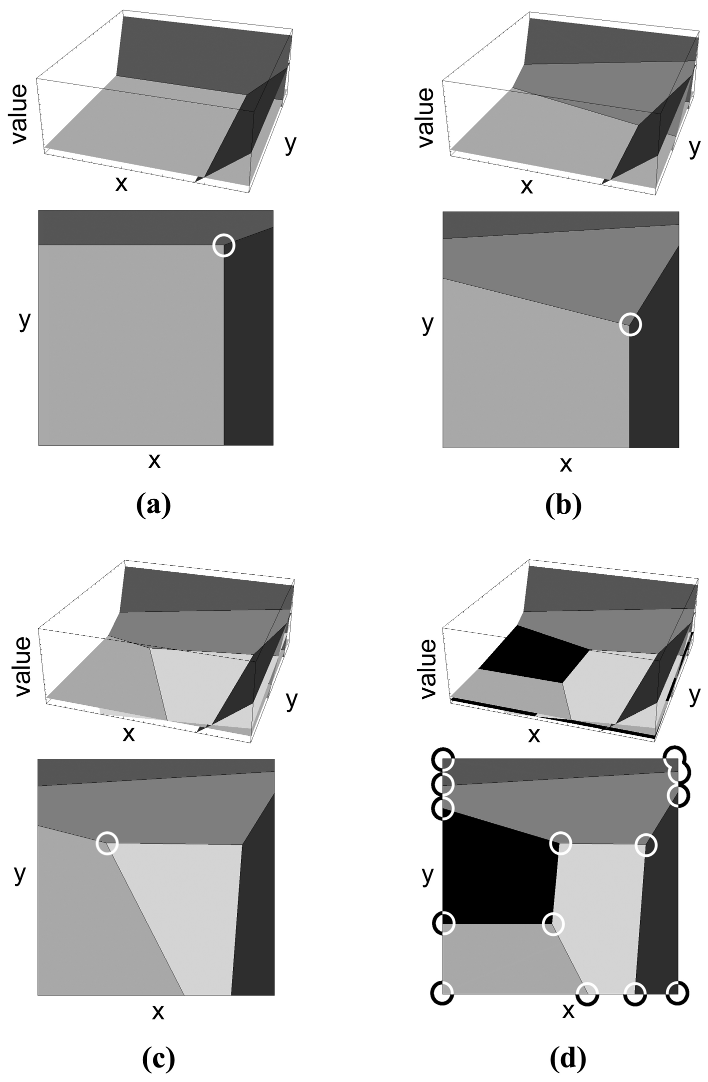


Figure 6: Intersecting Planes. (a) The first iteration checks the corners of the parameter space: $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$, which yields three planes. In the second iteration one of the intersection points in the top surface is chosen (circled), and a new optimal policy is found and added in (b). This process repeats until all fourteen of the intersections on the top surface between known optimal policies (circled in (d)) have been checked and no new optimal policies found. The six optimal policies in (d) form the optimal coverage set based on these intersection points.

and $x = 1.0$ then that same policy is optimal between those two points because we are working with linear equations.

Figure 6 shows the same algorithm with a two dimensional parameter space (two constraints). Here we are looking at planes in three dimensions. Figure 6 (a) shows the three planes (optimal policies) found at the four corners of the parameter space. The circled point is the next one selected and Figure 6 (b) shows the new plane found at that point. If all of the points circled in Figure 6 (d) have been checked and no new optimal policies found, then the algorithm terminates.

The next two theorems prove that the underlying idea for this step in the algorithm is correct. The idea is that if a set of points has the same optimal policy, then any point enclosed by those points also has that optimal policy because we are working with linear functions.

Theorem 3 *If two points $\bar{x}$ and $\bar{y}$ in n-dimensional parameter space have the same corresponding optimal policy π, then all points on the line segment between $\bar{x}$ and $\bar{y}$, $f(\alpha) = \bar{x} + \alpha(\bar{y} - \bar{x})$, $0 \leq \alpha \leq 1$, have optimal policy π.*

Proof. Let π be the optimal policy at $\bar{x}$ and $\bar{y}$, $\bar{z}$ be a point on the line between $\bar{x}$ and $\bar{y}$, and π' be the optimal policy at $\bar{z}$:

$\pi = \text{argmax}_\pi V_\pi^{\bar{x}}(\hat{s}_i^0) = \text{argmax}_\pi V_\pi^{\bar{y}}(\hat{s}_i^0)$,
$\bar{z} = f(\alpha_0)$, $0 < \alpha_0 < 1$, and
$\pi' = \text{argmax}_{\pi'} V_{\pi'}^{\bar{z}}(\hat{s}_i^0)$.

Assume that at $\bar{z}$ the value of the optimal policy π' is strictly greater than the value of π, $V_{\pi'}^{\bar{z}}(\hat{s}_i^0) > V_\pi^{\bar{z}}(\hat{s}_i^0)$. We know $V_\pi^{\bar{x}}(\hat{s}_i^0) \geq V_{\pi'}^{\bar{x}}(\hat{s}_i^0)$ because π is optimal at $\bar{x}$. Since $V(\cdot)$ and $f(\cdot)$ are linear functions, we can calculate their value at $f(1) = \bar{y}$ by computing the unit slope.

$$\begin{aligned}
V_\pi^{\bar{y}}(\hat{s}_i^0) &= \frac{V_\pi^{\bar{z}}(\hat{s}_i^0) - V_\pi^{\bar{x}}(\hat{s}_i^0)}{\alpha_0} + V_\pi^{\bar{x}}(\hat{s}_i^0) \\
&= \frac{1}{\alpha_0} V_\pi^{\bar{z}}(\hat{s}_i^0) - \left(\frac{1-\alpha_0}{\alpha_0}\right) V_\pi^{\bar{x}}(\hat{s}_i^0) \\
&\leq \frac{1}{\alpha_0} V_\pi^{\bar{z}}(\hat{s}_i^0) - \left(\frac{1-\alpha_0}{\alpha_0}\right) V_{\pi'}^{\bar{x}}(\hat{s}_i^0) \\
&< \frac{1}{\alpha_0} V_{\pi'}^{\bar{z}}(\hat{s}_i^0) - \left(\frac{1-\alpha_0}{\alpha_0}\right) V_{\pi'}^{\bar{x}}(\hat{s}_i^0) \\
&= \frac{V_{\pi'}^{\bar{z}}(\hat{s}_i^0) - V_{\pi'}^{\bar{x}}(\hat{s}_i^0)}{\alpha_0} + V_{\pi'}^{\bar{x}}(\hat{s}_i^0) \\
&< V_{\pi'}^{\bar{y}}(\hat{s}_i^0)
\end{aligned}$$

This contradicts the precondition that π is optimal at $\bar{y}$, therefore $V_\pi^{\bar{z}}(\hat{s}_i^0) = V_{\pi'}^{\bar{z}}(\hat{s}_i^0)$ for any $\bar{z}$ between $\bar{x}$ and $\bar{y}$. □

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 shows that for a line segment (polyhedron in 1 dimension) with endpoints with the same optimal policy, every point along that line also has the same optimal policy. This can be inductively extended to higher dimensions because any point in the interior falls on a line segment with endpoints on the edges of the polyhedron.

Corollary 1 *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.*

There is some similarity between this step of the algorithm and other algorithms that are trying to find a piecewise-linear and convex function, like dynamic programming for POMDPs. In the POMDP case the function represents the optimal actions over the belief states. In the coverage set algorithm it represents the optimal policies over the parameter space. The primary difference between these two is that here we can efficiently generate the optimal policy for a point in the parameter space, while with a POMDP there is no way to efficiently generate the optimal action for a given belief state. The pruning in dynamic programming for POMDPs is in response to that inefficiency and is not relevant to the coverage set algorithm. Kaelbling, Littman, and Cassandra (1998) give a detailed description of a POMDP and accompanying algorithm.

The part of the algorithm discussed in this section is handled by the function COVERAGE-SET in Figure 4. It takes an MDP and a joint reward structure and returns the optimal coverage set, based on Theorem 1. To illustrate how this works, we will step through a small example.

Consider an instance of the Mars 2-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, INTERSECT is called on the set of known 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 6(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 is computed by SOLVE, which can use standard dynamic programming algorithms. The value of a policy and a point in parameter space is:

$$\begin{aligned} V_{\pi_1}^{\bar{x}\pi_2}(\hat{s}_1^0) &= P(E_1^1|\pi_1)P(E_1^2|\pi_2)c_1 + \\ &\quad P(E_2^1|\pi_1)P(E_2^2|\pi_2)c_2 + V_{\pi_1}(\hat{s}_1^0). \end{aligned}$$

For a given π_1, the value function is a plane over the parameter space. The plane for each of 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 an optimal coverage set for agent i (considering the joint reward ρ), finding the optimal joint policy is straightforward. From Proposition 1 and Definition 18 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 14.

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.

1. **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.

2. **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 1 if no new optimal policies are found from those points, then the set of optimal policies is the optimal coverage set.

3. **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.

□

3.4 Running Time

An analysis of the algorithm as presented is quite straightforward. Constructing the augmented MDP in the function AUGMENT can be done in polynomial time with efficient representations of ρ and the MDP. The resulting augmented MDP is the same size as the original. SOLVE returns an optimal policy for an MDP, which for dynamic programming is known to be polynomial. INTERSECT finds a set of points. Each point involves solving a system of linear equations, which is polynomial in $|\rho|$. However, the number of points, while

$$\rho = [\ (E_1^1, E_1^2, E_1^4, c_1),\ (E_2^3, E_2^5, c_2),\ (E_3^2, E_3^3, c_3)\ ]$$

(a)

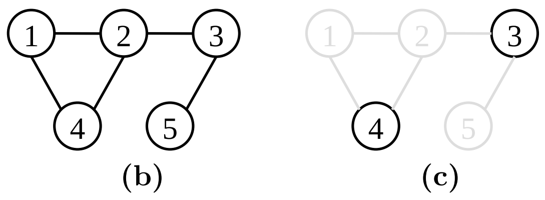


Figure 7: (a) A joint reward structure ρ over five agents. (b) The constraint graph. (c) The constraint graph after agents 1, 2, and 5 have found their optimal coverage set. The remaining agents 3 and 4 are not connected.

polynomial in $|OCS|$, is exponential in the number of dimensions $|\rho|$, $O(|ocs|^{|\rho|+1}/(|\rho|+1)!)$. For each point the optimal policy is found, which is polynomial in the size of the state space.

3.5 Extension to n Agents

Extending the coverage set algorithm to n agents is fairly straightforward. The reason is that the core of the algorithm, finding the optimal coverage set for agent i, does not depend on any particular behavior for agent j. When there are n agents, agent 1 can compute its optimal coverage set using the same algorithm by viewing the other $n-1$ agents as one other agent. The only real change is that the notation becomes much more difficult to read. A point in the parameter space (Definition 16) for agent 1 becomes

$$\bar{x}_{\pi_2 \ldots \pi_n} = \left[\prod_{k=2}^{|G_1|} P(E_1^{g_{1k}} | \pi_{g_{1k}}), \ldots, \prod_{k=2}^{|G_{|\rho|}|} P(E_{|\rho|}^{g_{|\rho|k}} | \pi_{g_{|\rho|k}}) \right],$$

assuming that g_{*1} refers to agent 1. With two agents, dimension k represented the probability that the other agent would satisfy its part of constraint k. With n agents, dimension k represents the probability that the other $n-1$ agents will satisfy their parts of constraint k. Since the agents are independent, this is simply the product of the probabilities of each of the other agents satisfying its part of constraint k. The value of each dimension still ranges between 0 and 1, so this change does not affect the computation of the optimal coverage set. The dimensionality of the parameter space for agent i is no longer $|\rho|$, but is only the number of constraints agent i participates in. For example, if agent i is only involved in three constraints then the dimensionality of the search for the optimal coverage set for agent i is three, no matter how many other constraints there are in ρ.

The interesting part of the algorithm that significantly changes is which agents must compute their optimal coverage set. When there were only two agents, we had to find the optimal coverage set for only one of the two agents. With n agents, we now have to find the optimal coverage set for some subset of the agents, possibly though not necessarily $n-1$ of them. For each constraint in the problem, at most one agent involved in that constraint does not have to find its optimal coverage set. This can easily be demonstrated by a simple reduction to a graph (see Figure 7). Let every agent be represented by a vertex. Add an edge between every two vertices that share a constraint (if there is not an edge there already). For example, for constraint 1, $G_1 = \{1, 2, 4\}$, add the edges $E(1, 2)$, $E(1, 4)$, and $E(2, 4)$. The optimal coverage set must be found for a subset of the agents such that when those vertices and all incident edges are removed from the graph, the remaining vertices are completely disconnected (no edges remain). Some valid subsets in this example are $\{1, 2, 5\}$, $\{2, 4, 5\}$, $\{1, 4, 3\}$, among others.

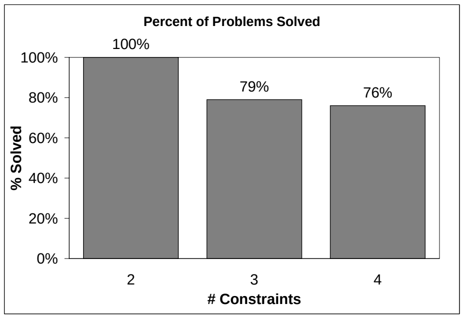
Given a valid subset of agents, the final step in the coverage set algorithm becomes a search process through all combinations of those agents' optimal policies. Given a policy for each in the subset of agents, maximizing the global value reduces to solving the corresponding augmented MDP for each of the remaining agents. As a distributed search process it maps nicely into a DCOP (Distributed Constraint Optimization Problem) (Liu & Sycara, 1995; Yokoo & Durfee, 1991). As a DCOP, each agent has one variable, which represents the policy that agent adopts. The domain of that variable is the optimal coverage set for that agent. There is a cost function for each of the agents' local rewards, and one for each constraint in ρ. Solving the DCOP finds the variable assignment that maximizes the sum of the cost functions.

Choosing any subset of agents to find their optimal coverage set is easy, but choosing the best subset is not. In fact, it is not entirely clear what best means in this case. The smallest valid subset is not necessarily the one with the lowest worst case complexity, which is not necessarily the one with the lowest average case complexity, which is not necessarily the one that is easiest to distribute. The complexity of finding the optimal coverage set is exponential in the number of constraints involved, so choosing a valid subset of agents that minimizes the maximum dimensionality is likely to be better than minimizing the number of agents. However, the complexity also depends polynomially on the number of policies in the optimal coverage set. If a particular local problem could be characterized in a way that indicates the expected number of optimal policies, then that could affect which valid subset should be chosen. Without this information, one possible solution is to work on finding all of the optimal coverage sets in parallel (possibly distributed, with each agent solving their own local problem) until a valid subset has been solved. Examining these issues in more depth remains the topic of future work.

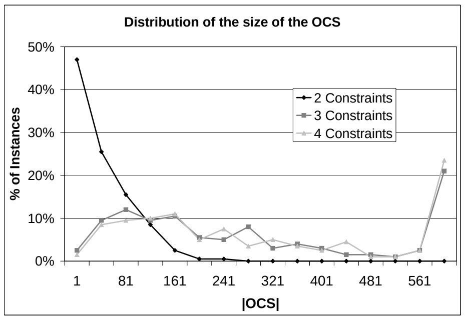
4. Experimental Results

We implemented a general version of this algorithm that works for any number of dimensions. The implementation varied slightly from what is presented in Figure 4 to improve the efficiency. The primary change is instead of computing all of the intersection points in the function INTERSECT(), we generate one at a time. Many of the points returned by INTERSECT() can be discarded without the need to run SOLVE() on the augmented MDP at that point. This is because many points lie beneath other known optimal policies or because those points are out of bounds (> 1 or < 0 in some dimension of the parameter space).

The results were verified by independently checking that every policy in the optimal coverage set was optimal for some point in the parameter space and by randomly choosing points in the parameter space and verifying that its optimal policy was in the set.



(a)



(b)

Figure 8: (a) The percentage of problems that had fewer than 600 policies in the optimal coverage set. Those with more than 600 were not solved. (b) The distribution over the size of the optimal coverage set for 2, 3 and 4 constraints.

4.1 Rover Exploration Problem

We tested this algorithm on problem instances 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. The uncertainty in the problem is the length of time it takes to execute the tasks.

The problem instances generated had a total time of 10 units, and 5 sites for each rover. The original DEC-MDP had 250 world states ($10 \times 5 \times 5$), and both rovers' local decision problem had 50 states (10×5). On average, each rover was able to collect data at only half of its sites.

Some of the sites were shared between the two agents. These sites were assumed to be half redundant, which means that if both rovers collect at that site then the reward received by the system is 1.5 times that received if only one rover does. For example, if rover 1 collects then it receives local reward r. Similarly, if rover 2 collects it receives local reward r. Now if both rover 1 and rover 2 collect at that site, then they each receive r and the system receives a penalty of $-0.5r$ for a net total global reward of $1.5r$.

Data was collected from 200 randomly generated problems. Each problem was solved three times, once with sites 3 and 4 shared between the agents, once with $2, 3$ and 4, and also with $1, 2, 3$ and 4. Problems that had more than 600 policies in their optimal coverage set were considered hard problems and skipped. Figure 8(a) shows the percentage of problems that were solved for the given number of constraints. Figure 8(b) shows the distribution of the size of the optimal coverage set. Of the problems solved, the average number of optimal policies in the optimal coverage set was 59, 217 and 230 for 2, 3 and 4 constraints respectively. While only 76% of the problems with four constraints were solved in the available time, this should be viewed in the context that complete policy search would not have been able to solve a single one of these problems in that time.

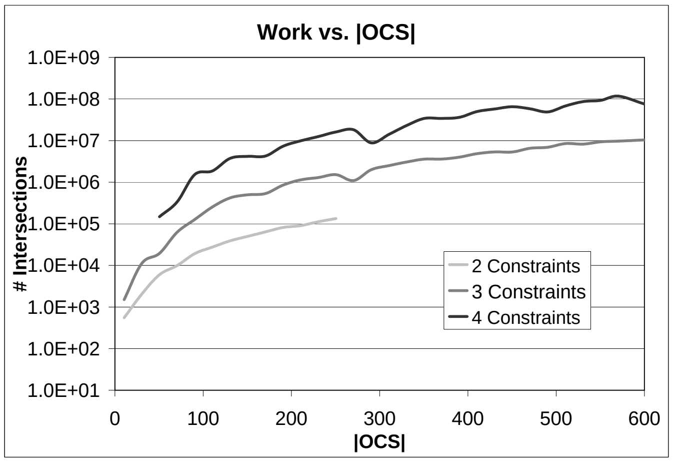


Figure 9: The amount of work measured by the number of intersections increases polynomially with the size of the optimal coverage set, $|OCS|$.

To get a general idea about the length of time to solve these problems, with four constraints the fastest problem had 23 optimal policies in the optimal coverage set and took about 1.5 seconds on a 2.8GHz Pentium 4. The longest one took about 4 hours and had 597 optimal policies.

There are two measures of work that we use to evaluate the performance. First is the number of times the planes are intersected. Each intersection represents solving a system of linear equations. The second measure is the number of times dynamic programming was used to solve an augmented MDP. The graphs were essentially the same regardless which measure we used, so most will be presented with the number of intersections as the measure of work. Figure 9 shows how the amount of work required to solve a problem relates to the size of the optimal coverage set.

The difficulty in solving a particular instance depends on many characteristics of the problem. The characteristic that we focus on here is the magnitude of the constant rewards in ρ, $c_1 \dots c_x$, as compared to the other rewards in the system. Intuitively, if the value of a shared task is much greater or much less than the other tasks available to an agent, then the difficulty to decide whether to do the shared task decreases. This idea can be seen in Figure 10. When looking along either the x or y dimension, the density of optimal policies is greatest around the middle, and much less at either extreme.

Previously, we collected data on problems that were half redundant. The value of the shared sites ranged from r to $0.5r$, i.e., $r + P \times (-0.5r)$ where P is the probability of the other agent collecting at that site. Instead consider a problem in which the shared tasks were complementary. If either agent collects data it receives $0.5r$, and if they both collect data then the system receives an extra $0.5r$ for a total of $1.5r$. The values in this example range from $0.5r$ to r. The complexity of these two problems is identical because the policies in the optimal coverage set are identical. However, the optimal joint policy will be different because the problems are not identical but mirror images of each other. Similarly, if we

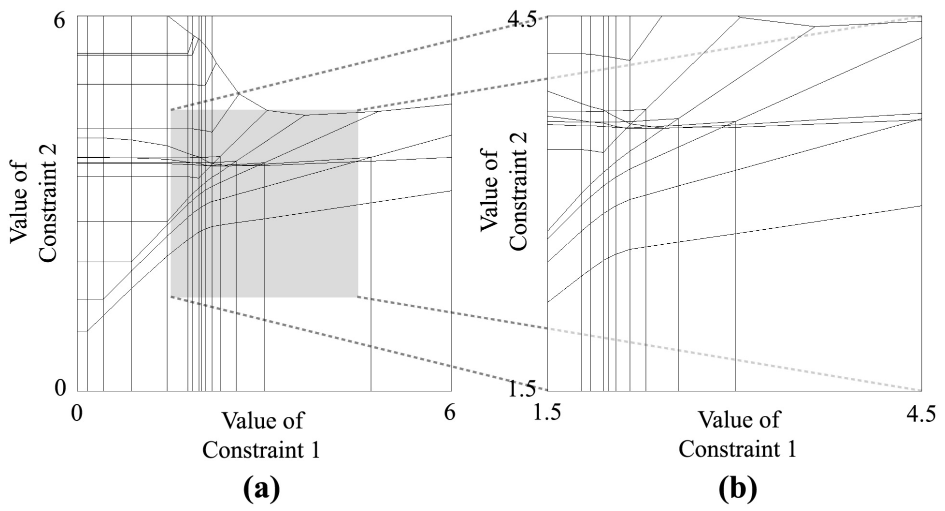


Figure 10: Illustration of the upper surface of the optimal coverage set for a 2 constraint problem. Each bounded region corresponds to one policy and the area over which it is optimal. (a) Both constraints have a local reward of 0 and an additional reward of 6. (b) Local reward is 1.5 and additional reward is 3. Notice how this set of policies is equal to the middle area of (a). Problem (b) is a subset of (a).

increase c then the new optimal coverage set is a superset of the original and therefore more difficult to solve. Figure 10 illustrates this relation.

The focus of the experimental results is primarily on step two of the algorithm (finding the optimal coverage set) because that is where most of the computational complexity lies. That changes, however, as the number of agents increases. Finding the optimal coverage set is internal to each agent and depends on the number of constraints that agent has, not the number of agents in the system. Therefore, as the number of agents increase, the complexity of the third step (searching through the optimal coverage sets for the optimal joint policy) becomes more important. The third step is a distributed constraint optimization problem (DCOP), the evaluation of which is out of the scope of this paper. Mailler and Lesser (2004), and Modi, Shen, Tambe, and Yokoo (2003) present state-of-the-art algorithms for solving DCOPs.

4.2 Approximation

While the complexity of transition independent decentralized MDPs is significantly lower than models like the DEC-MDP, it is still intractable for large problems and certain hard instances of small problems. Fortunately, it turns out that the coverage set algorithm is naturally an anytime algorithm with some very nice characteristics.

Figure 11 shows the anytime performance profile over the same set of problems discussed earlier. The very first solution generated by the algorithm was better than 90% of optimal in nearly every problem. After only 1000 intersections, 88% of the 4-constraint problems

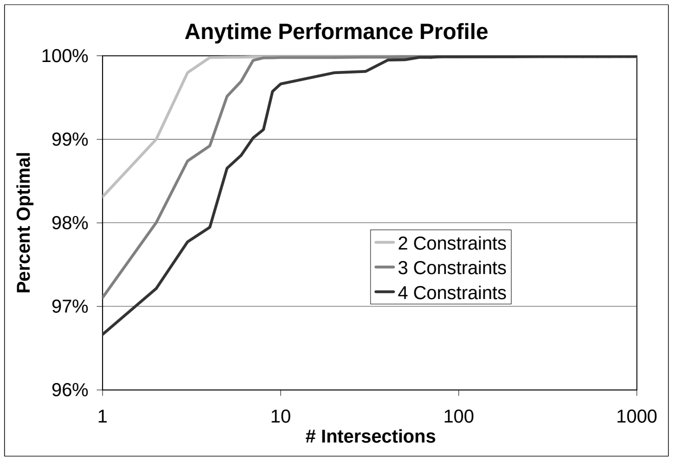


Figure 11: The value of the best known joint policy (as a percentage of the optimal policy) as a function of the total work done.

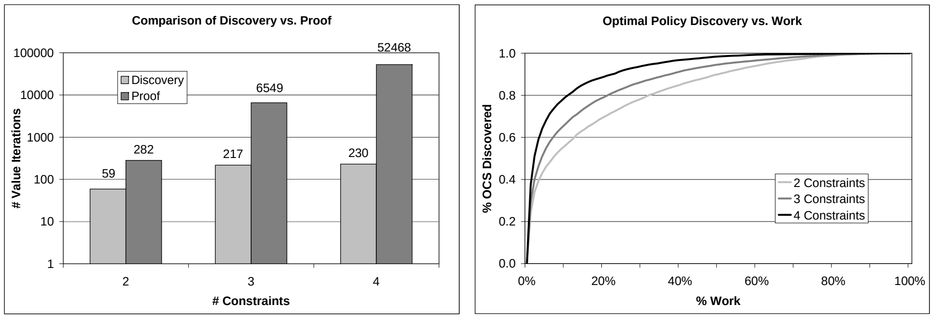
had already found the optimal solution and the others were within 99% of optimal. 1000 intersections is only a small fraction of the total work done to solve one of those problems (average was 18 million).

There are several reasons for the excellent performance. First, each solution generated by the algorithm is a locally optimal solution. This means that agent 1 has a belief about what agent 2 will do and generates its optimal corresponding policy. Agent 2 then generates its optimal policy given agent 1's policy. This pair of policies is not a Nash equilibrium because Agent 1's belief about Agent 2 and Agent 2's actual policy may be different. However, in these randomly generated problems it is usually quite good.

A second reason is that the amount of work required to discover the policies in the optimal coverage set is significantly less than that required to prove that the set is complete, as shown in Figure 12(a). As the number of constraints grows, the gap between the discovery work and the proof work increases. In addition, the discovery is front-loaded as seen in Figure 12(b). Finally, the more area over which a policy is optimal, the more likely that policy will be discovered earlier and the more likely that policy will be in the optimal joint policy (for randomly generated problems). Taken together, this explains why the optimal joint policy was usually discovered after only 1000 intersections. For the 4-constraint problems, this is less than one hundredth of one percent of the total work to solve the problem.

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-complete, and the only general algorithm for solving this problem optimally (Hansen et al., 2004) requires further development before it can be applied to the kind of problems we target in this paper. We have identified an inter-



(a) **(b)**

Figure 12: (a) As the number of constraints increase, the gap between the work to discover new optimal policies and the work to prove the set is complete increases. (b) The discovery of the optimal planes happens primarily in the beginning.

esting subset of problems called transition independent decentralized MDPs, and designed and implemented an algorithm that returns an optimal joint policy for these problems.

This algorithm is the first optimal algorithm we are aware of able to solve (in a reasonable amount of time) a non-trivial subclass of DEC-MDPs. Of the randomly generated problems that we tested with four constraints, we solved 76% of them in the available time. Viewed in the context that none of these problems were solvable using other existing methods, we feel that this success rate is significant. In the future, we also plan to test this algorithm with other domains besides the rover one presented here. For example, another 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.

We do acknowledge, however, that even with the coverage set algorithm, problems can quickly become very difficult to solve. To address this issue we examined the anytime performance of the algorithm. Discovering the optimal policies in the optimal coverage set is a small fraction of the work involved in proving that the optimal coverage set is complete. In addition, most of the discovery happens early on and as the number of dimensions scales up the discovery is increasingly front-loaded. This is a very nice anytime characteristic that leads to good anytime performance. How much this performance is due to the algorithm and how much is due to the problem domain is an area we plan to explore as we test new domains with this algorithm.

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. Using it, we can quickly determine how much the other agents' probabilities must change before this agent would change its 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, by NSF under grants IIS-0219606 and IIS-0004112, and by AFOSR under grant F49620-03-1-0090. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of NASA, NSF or AFOSR.

References

Becker, R., Zilberstein, S., & Lesser, V. (2004). Decentralized Markov decision processes with structured transitions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, Vol. 1, pp. 302–309, New York City, New York. ACM Press.

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. V. (2003). Transition-independent decentralized Markov decision processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pp. 41–48, Melbourne, Australia. ACM Press.

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, *27*(4), 819–840.

Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 478–485.

Decker, K., & Lesser, V. (1993). 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*, *2*, 215–234.

Ghavamzadeh, M., & Mahadevan, S. (2002). A multiagent reinforcement learning algorithm by dynamically merging Markov decision processes. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems*, Bologna, Italy. ACM Press.

Goldman, C. V., & Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pp. 137–144, Melbourne, Australia. ACM Press.

Goldman, C. V., & Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. To appear in *Journal of Artificial Intelligence Research.*

Guestrin, C., Venkataraman, S., & Koller, D. (2002). Context specific multiagent coordination and planning with factored MDPs. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 253–259, Edmonton, Canada.

Hansen, E., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. *Proceedings of the Ninteenth National Conference on Artificial Intelligence*, 709–715.

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

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Liu, J., & Sycara, K. P. (1995). Exploiting problem structure for distributed constraint optimization. In *Proceedings of the First International Conference on Multi-Agent Systems*, pp. 246–253.

Mailler, R., & Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, 438–445.

Modi, P. J., Shen, W.-M., Tambe, M., & Yokoo, M. (2003). An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pp. 161–168.

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

Nair, R., Tambe, M., Yokoo, M., Pynadath, D., & Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*.

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

Papadimitriou, C. H., & Tsitsiklis, J. (1982). On the complexity of designing distributed protocols. *Information and Control*, *53*, 211–218.

Papadimitriou, C. H., & Tsitsiklis, J. (1986). Intractable problems in control theory. *SIAM Journal on Control and Optimization*, *24*(4), 639–654.

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

Peshkin, L., Kim, K.-E., Meuleau, N., & Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 489–496, San Francisco, CA. Morgan Kaufmann.

Pynadath, D. V., & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, *16*, 389–423.

Shen, J., Lesser, V., & Carver, N. (2003). Minimizing communication cost in a distributed Bayesian network using a decentralized MDP. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pp. 678–685, Melbourne, Australia. ACM Press.

Tumer, K., Agogino, A. K., & Wolpert, D. H. (2002). Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems*.

Washington, R., Golden, K., Bresina, J., Smith, D., Anderson, C., & Smith, T. (1999). Autonomous rovers for Mars exploration. In *Proceedings of the IEEE Aerospace Conference*.

Wolpert, D., Wheeler, K., & Tumer, K. (1999). General principles of learning-based multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents*, pp. 77–83, Seattle, WA.

Xuan, P., & Lesser, V. (2002). Multi-agent policies: From centralized ones to decentralized ones. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems*, pp. 1098–1105. ACM Press.

Xuan, P., Lesser, V., & Zilberstein, S. (2001). Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 616–623, Montreal, Canada. ACM Press.

Yokoo, M., & Durfee, E. H. (1991). Distributed constraint optimization as a formal model of partially adversarial cooperation. Tech. rep. CSE-TR-101-91, The University of Michigan, Ann Arbor, Michigan.

Zilberstein, S., Washington, R., Bernstein, D. S., & Mouaddib, A. I. (2002). Decision-theoretic control of planetary rovers. In Beetz, M., et al. (Eds.), *Plan-Based Control of Robotic Agents*, No. 2466, in Lecture Notes in Artificial Intelligence, pp. 270–289. Springer.