Anytime Coordination Using Separable Bilinear Programs

Marek Petrik and **Shlomo Zilberstein**
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{petrik, shlomo}@cs.umass.edu

Abstract

Developing scalable coordination algorithms for multi-agent systems is a hard computational challenge. One useful approach, demonstrated by the Coverage Set Algorithm (CSA), exploits structured interaction to produce significant computational gains. Empirically, CSA exhibits very good anytime performance, but an error bound on the results has not been established. We reformulate the algorithm and derive both online and offline error bounds for approximate solutions. Moreover, we propose an effective way to automatically reduce the complexity of the interaction. Our experiments show that this is a promising approach to solve a broad class of decentralized decision problems. The general formulation used by the algorithm makes it both easy to implement and widely applicable to a variety of other AI problems.

Introduction

The success of Markov decision processes in modeling stochastic decision problems motivated researchers to extend the model to cooperative multi-agent settings. In these settings, several agents – each having different partial information about the world – must cooperate with each other in order to achieve some joint objective. Such problems are common in practice, but despite recent progress in this area, state-of-the-art algorithms are generally limited to very small problems (Seuken & Zilberstein 2005). This has motivated the development of algorithms that either solve a simplified class of problems (Kim *et al.* 2006) or provide approximate solutions (Emery-Montemerlo *et al.* 2004; Seuken & Zilberstein 2007).

One promising approach – called the Coverage Set Algorithm (CSA) (Becker *et al.* 2004) – was developed for domains with limited interaction between the agents. The problem is formalized as a decentralized Markov decision problem with transition and observation independence, which we denote as DEC-MDP. The objective is to maximize the cumulative reward over some finite horizon with no discounting. Essentially, the two agent case consists of two MDPs with an overall reward function that depends on both states. CSA works by first enumerating the policies of one agent that are best responses to at least one policy of the other agent, that is, policies that are not dominated. Then the algorithm searches over these policies to get the best joint policy for all agents. CSA uses a compact representation and avoids explicitly enumerating all possible policies.

Empirically, CSA was shown to be quite efficient, solving relatively large problems. It also exhibits good anytime performance (Becker *et al.* 2004). When solving a multi-rover coordination problem, a solution value within 1% of optimal is found within 1% of the total execution time on average. Unfortunately, this is only known in hindsight once the optimal solution is found. Additionally, the algorithm has several drawbacks. It is numerically unstable and its complexity increases exponentially with the number of best-response policies. Runtime varies widely over different problem instances. Finally, the algorithm is limited to a relatively small subclass of distributed coordination problems.

This paper makes several key contributions. First, we present a reformulation of CSA – using separable bilinear programs (Horst & Tuy 1996) – that is more general, more efficient, and easier to implement. Then, we derive an error bound using the convexity of the best-response function, without relying on the optimal solution. This generalizes similar approximate methods for POMDPs (Lovejoy 1991; Pineau *et al.* 2006). The new algorithm exhibits excellent anytime performance, making it suitable for time-constrained situations. Finally, we derive offline bounds on the approximation error and develop a general method for automatic dimensionality reduction. For clarity, we limit the discussion to cooperative two-agent problems, but the approach works for any number of agents and it requires only a minor modification to work in competitive settings.

The paper is organized as follows. First, we describe how to represent a DEC-MDP as a separable bilinear program. Then we outline the algorithm and derive the online error bound. A detailed description of the algorithm comes next, followed by the derivation of an offline error bound based on the number of iterations. We then present the automatic dimensionality-reduction procedure. Finally, we compare our algorithm to the original CSA and mixed-integer linear programming on the original *Mars rover* problem.

Outline of the Algorithm

The algorithm we develop was originally designed for solving multi-agent coordination problems that arise in the Mars rover domain (Becker *et al.* 2004). The domain involves two

autonomous rovers that visit several sites in a given order and may decide to perform certain experiments. The overall activity must be completed within a given time limit. The uncertainty about the duration of each experiment is modeled by a given discrete distribution. While the rovers operate independently and receive local rewards for each completed experiment, the global reward function depends also on some experiments completed by both rovers. The interaction between the rovers is limited to a relatively small number of such overlapping tasks.

This problem was formulated as a transition- and observation-independent decentralized Markov decision process (DEC-MDP). The problem is composed of two MDPs with state-sets S_1, S_2 and action sets D_1, D_2. The functions r_1 and r_2 define *local* rewards for action-state pairs. The initial state distributions are α_1 and α_2. The MDPs are coupled through a global reward function defined by the matrix R. Each entry $R(i,j)$ represents the joint reward for state-action i by one agent and j by the other.

Any DEC-MDP can be formulated as a bilinear mathematical program as follows. Vector variables x and y represent the state-action probabilities for each agent, as used in the dual linear formulation of MDPs. Given the transition and observation independence, the feasible regions may be defined by linear equalities $A_1x = \alpha_1$ and $x \geq 0$, and $A_2y = \alpha_2$ and $y \geq 0$. The matrices A_1 and A_2 are the same as for dual formulation of finite-horizon MDPs (Puterman 2005). That is, they represent inequalities of the following form:

$$\sum_{a' \in D_1} x(s',a') - \sum_{s \in S_1} \sum_{a \in D_1} \mathbf{P}\left[s'|s,a\right] x(s,a) = \alpha_1(s'),$$

for every $s' \in S_1$. This results in the following formulation of the problem.

$$\begin{array}{ll} \text{maximize} & r_1^T x + x^T R y + r_2^T y \\ \text{subject to} & A_1 x = \alpha_1 \quad x \geq 0 \\ & A_2 y = \alpha_2 \quad y \geq 0 \end{array} \tag{1}$$

A policy may be extracted from a solution of Eq. (1) in the same way as for the dual formulation of discounted infinite-horizon MDPs (Puterman 2005). Briefly, in state s, the policy is to take action a if and only if $x(s,a) > 0$. This can be further abstracted as the following mathematical program.

$$\begin{array}{ll} \text{maximize} & f(x,y,\hat{y}) = r^T x + x^T C y + s^T y + t^T \hat{y} \\ \text{subject to} & Ax \leq b \quad B_1 y + B_2 \hat{y} \leq c \end{array} \tag{2}$$

This formulation – generally known as a (separable) bilinear program – is used in our algorithm, making it both simple to present and more general. In addition to multi-agent coordination, it can be used to solve a variety of problems such as robotic manipulation (Pang *et al.* 1996), bilinear separation (Bennett & Mangasarian 1992), and even general linear complementarity problems (Mangasarian 1995).

For general C, the problem was shown to be NP complete (Becker *et al.* 2004). The membership in NP follows from that fact that there exists an optimal solution that corresponds to some vertex of the feasible region (Horst & Tuy 1996; Mangasarian 1995).

A large number of algorithms have been developed for solving Eq. (2), modeling it as a linear complementarity problem (Murty 1988) or a mixed integer linear program (Petrik & Zilberstein 2007). An iterative algorithm, which does not guarantee optimality, is described in (Bennett & Mangasarian 1992). A good overview of other algorithms can be found in (Horst & Tuy 1996). We focus here mostly on improving CSA rather than on a comprehensive comparison of these methods. Moreover, our approach focuses on problems in which the function C is constant in most dimensions due to the limited interaction.

In Eq. (2), variable x is an m-dimensional vector and y is n-dimensional. For simplicity, we denote the feasible sets as

$$X = \{x \,|\, Ax \leq b\} \quad Y = \{y \,|\, B_1 y + B_2 \hat{y} \leq c\},$$

which represent the feasible solutions for variables x and y respectively. We also implicitly assume that $x \in X$ and $y \in Y$. To develop the algorithm, we need the following assumption.

Assumption 1. Feasible sets X and Y are bounded.

The assumption does not present a significant limitation, since often in practice, the variables can be bounded by finite intervals. As mentioned before, the main idea in CSA is to compute a subset $\tilde{X}$ of X that contains only those elements that satisfy necessary optimality conditions. Specifically, x is optimal for at least one element of Y. The algorithm then solves either of the two modified problems:

$$\begin{array}{ll} \text{maximize} & f(x,y,\hat{y}) \\ \text{subject to} & x \in \tilde{X} \quad B_1 y + B_2 \hat{y} \leq c \end{array} \tag{3}$$

The latter problem can be solved easily by enumerating all $x \in \tilde{X}$ and solving for y using a linear program. It is possible since the set $\tilde{X}$ is finite (Mangasarian 1995). In some cases, such as when the agents are competitive, $y \in Y$ needs to be replaced by $y \in \tilde{Y}$. As a result, the approximation bounds we present later are doubled. The actual procedure to obtain $\tilde{X}$ is described in the next section.

To quantify the approximation error when using $\tilde{X}$, we define the following *best-response* and *approximate best-response* functions:

$$g(y) = \max_{x \in X} r^T x + x^T C y, \quad \tilde{g}(y) = \max_{x \in \tilde{X}} r^T x + x^T C y$$

Both $g(y)$ and $\tilde{g}(y)$ are maximum of a set of linear function, and as a result they are convex. Clearly we have $\tilde{g}(y) \leq g(y)$. This function defines the maximal objective value for any fixed value of y and it is used to determine the maximal bound on the current approximation error. We describe the properties of $g(y)$ and how to approximate it in the following sections. When the difference between $g(y)$ and $\tilde{g}(y)$ is at most ϵ, then the difference between optimal solutions of Eqs. (2) and (3) is also at most ϵ.

Let $f(x^*, y^*, \hat{y}^*)$ be the optimal solution. The maximal error when using Eq. (3) can be bounded as follows. Assuming $\tilde{g}(y) \geq g(y) - \epsilon$, for all $y \in Y$, then

$$\begin{aligned} f(x^*, y^*, \hat{y}^*) &= \max_{x \in X} f(x, y^*, \hat{y}^*) \geq \\ &\geq \max_{x \in \tilde{X}, B_1 y + B_2 \hat{y} \leq c} f(x,y,\hat{y}) - \epsilon. \end{aligned}$$

The difference $g(y) - \tilde{g}(y)$ can be bounded using the convexity of these functions. As we describe in more detail later, we construct the set $\tilde{X}$ such that for some $y \in Y$ we actually have $g(y) = \tilde{g}(y)$. As a result, we can get the maximal difference ϵ using the following upper bound based on Jensen's inequality.

Lemma 1. *Let $y_i \in Y$ for $i = 1, \ldots, n+1$. Then $g\left(\sum_{i=1}^{n+1} c_i y_i\right) \leq \sum_{i=1}^{n+1} c_i g(y_i)$, when $\sum_{i=1}^{n+1} c_i = 1$ and $c_i \geq 0$ for all i.*

Best-Response Calculation

We describe now in greater detail the algorithm to determine $\tilde{X}$. Some alternative approaches are discussed in the last section. The algorithm grows $\tilde{X}$ by evaluating $g(y)$ for multiple $y \in Y$ and by adding the corresponding best-response x into $\tilde{X}$. The choice of y is organized in a hierarchical fashion. The algorithm starts with evaluating $y_1 \ldots y_{n+1}$, the vertices of a polytope that contains Y, which exists based on Assumption 1. Given Lemma 1, we can now find such y_0 where the approximation error is maximal. Next we get $n+1$ new polytopes by replacing one of the vertices by $y_0, (y_0, y_2, \ldots), (y_1, y_0, y_3, \ldots) \ldots (y_1, \ldots, y_n, y_0)$. The old polytope is discarded and the above procedure is then repeatedly applied to the polytope with the maximal approximation error.

This procedure differs from the original CSA mainly in the choice of y_0. CSA does not keep any upper bound and evaluates $g(y)$ on all intersection points of planes defined by the current values in $\tilde{X}$, leading eventually to $g(y) = \tilde{g}(y)$ (Becker *et al.* 2004). Consequently, the number of these points rapidly increases with the increasing number of elements in $\tilde{X}$. In contrast, our approach is more selective and focuses on rapidly reducing the error bound.

For clarity, we simplified the above pseudo-code and did not address efficiency issues. In practice, $g(y_i)$ could be cached, and the errors ϵ_i could be stored in a prioritized heap or at least in a sorted array. In addition, a lower bound l_i and an upper bound u_i are calculated and stored for each polytope $S_i = (y_1 \ldots y_{n+1})$. The function $e(S_i)$ calculates their maximal difference on polytope S_i and the point where it is attained.

Let matrix Z have y_i as columns, and let $L = \{x_1 \ldots x_{n+1}\}$ be the set of the best responses for its vertices. We can represent a lower bound $l(y)$ for $\tilde{g}(y)$ and an upper bound $u(y)$ for $g(y)$ as

$$\begin{aligned} l(y) &= \max_{x \in L} r^T x + x^T C y \\ u(y) &= [g(y_1), g(y_2), \ldots]^T (Z + E_{n+1})^{-1} y, \end{aligned}$$

where E_{n+1} is a zero matrix with $(n+1)$-th row of ones. The upper bound correctness follows from Lemma 1. Notice that $u(y)$ is a linear function. That enables us to use a linear program to determine the maximal-error point.

Using all of $\tilde{X}$ instead of only L would lead to a tighter bound. This is easy to show in three-dimensional examples. However, this would substantially increase the computational complexity. Now, the maximal error on a polytope

Algorithm 1 Best-response function approximation

$S_1 \leftarrow (y_1 \ldots y_{n+1}), Y \subseteq S_1$
$\tilde{X} \leftarrow \{\arg\max_{x \in X} f(x, y_i, 0) \mid i = 1, \ldots, n\}$
$(\epsilon_1, \phi_1) \leftarrow e(S_1)$
$j \leftarrow 1$
while $\max_{i=1,\ldots,j} \epsilon_i \geq \epsilon_0$ **do**
  $i \leftarrow \arg\max_{k=1,\ldots,j} \epsilon_k$
  $y \leftarrow \phi_i$
  $\tilde{X} \leftarrow X \cup \{\arg\max_{x \in X} f(x, y, 0)\}$
  for $k = 1, \ldots, n+1$ **do**
    $j \leftarrow j + 1$
    $S_j \leftarrow (y, y_1 \ldots y_{i-1}, y_{i+1}, \ldots y_{n+1})$
    $(\epsilon_j, \phi_j) \leftarrow e(S_j)$
    $\epsilon_j \leftarrow \min\{\epsilon_i, \epsilon_j\}$
  end for
  $\epsilon_i \leftarrow 0$
end while
return $\tilde{X}$

S may be expressed as:

$$\begin{aligned} e(y) &\leq \max_{y \in S} u(y) - l(y) = \max_{y \in S} u(y) - \max_{x \in L} r^T x + x^T C y \\ &= \max_{y \in S} \min_{x \in L} u(y) - r^T x + x^T C y. \end{aligned}$$

We also have

$$y \in S \Leftrightarrow \left(y = Zz \wedge z \geq 0 \wedge e^T z = 1\right).$$

As a result, the point with the maximal error may be determined using the following linear program:

$$\begin{aligned} \text{maximize} \quad & \epsilon \\ \text{subject to} \quad & \epsilon \leq u(Zz) - r^T x + x^T C Z z \quad \forall x \in L \\ & e^T z = 1 \quad z \geq 0 \end{aligned} \tag{4}$$

Here e is a vector of all ones. The formulation is correct because all feasible solutions are bounded below the maximal error and any maximal-error solution is feasible.

Proposition 1. *The optimal solution of Eq. (4) is equivalent to* $\max_{y \in S} |u(y) - l(y)|$.

The maximal difference is actually achieved in points where some of the planes meet, as suggested in (Becker *et al.* 2004). However, it can be expected that checking these intersections is in fact very similar to running the simplex algorithm. In general, the simplex algorithm is preferable to interior point methods for this program because of its small size (Vanderbei 2001).

It remains to show that the iterative procedure of refining the polytopes is valid. This trivially implies that the maximal error is just the maximum of all errors on all polytopes. In addition, it shows that the polytopes do not overlap, and therefore there is no additional inefficiency in this regard.

Proposition 2. *In the proposed triangulation, the polytopes do not overlap and they cover the whole region.*

For lack of space, we do not provide the proof. Note that the calculated bound is not necessarily tight and thus it may actually increase rather than decrease. However, because the true error does not increase over successive iterations, the previous bound can be used when the new one is higher.

Offline Bound

In this section we develop an approximation bound that depends only on the number of points for which $g(y)$ is evaluated and the structure of the problem. In practice this is needed to be able to provide performance assurance without actually solving the problem. In addition the bound reveals which parameters of the problem influence the algorithm's performance.

We assume that the feasible sets have bounded L_2 norms. Given Assumption 1, this can usually be achieved by scaling the constraints.

Assumption 2. For all $x \in X$ and $y \in Y$, their norms satisfy $\|x\|_2 \leq 1$ and $\|y\|_2 \leq 1$.

The bound is derived based on the maximal slope of $g(y)$ and the maximal distance among the points.

Theorem 1. *To achieve an approximation error of at most ϵ, the number of points to be evaluated in a regular grid with k points in every dimension must satisfy: $k^n \geq \left(\frac{\|C\|_2\sqrt{n}^n}{\epsilon}\right)$.*

The theorem follows using basic algebraic manipulations from the following lemma.

Lemma 2. *Assume that for each $y_1 \in Y$ there exists $y_2 \in Y$ such that $\|y_1 - y_2\|_2 \leq \delta$ and $\tilde{g}(y_2) = g(y_2)$. Then the maximal approximation error is:*

$$e = \max_{y \in Y} g(y) - \tilde{g}(y) \leq \|C\|_2 \|y_1 - y_2\|_2.$$

Proof. Let y_1 be a point where the maximal error is attained. This point is in Y, because this set is compact. Now, let y_2 be the closest point in L_2 norm. Let x_1 and x_2 be the best responses for y_1 and y_2 respectively. From the definition of solution optimality we have:

$$\begin{aligned} r^T x_1 + s^T y_2 + x_1^T C y_2 &\leq r^T x_2 + s^T y_2 + x_2^T C y_2 \\ r^T (x_1 - x_2) &\leq -(x_1 - x_2)^T C y_2. \end{aligned}$$

The error now may be expressed as:

$$\begin{aligned} e &= r^T x_1 + s^T y_1 + x_1^T C y_1 - r^T x_2 - s^T y_1 - x_2^T C y_1 \\ &= r^T (x_1 - x_2) + (x_1 - x_2)^T C y_1 \\ &\leq -(x_1 - x_2)^T C y_2 + (x_1 - x_2)^T C y_1 \\ &\leq (x_1 - x_2)^T C (y_1 - y_2) \\ &\leq 2\|y_1 - y_2\|_2 \frac{(x_1 - x_2)}{\|(x_1 - x_2)\|_2}^T C \frac{(y_1 - y_2)}{\|y_1 - y_2\|_2} \\ &\leq 2\|y_1 - y_2\|_2 \max_{x \,|\, \|x\|_2 \leq 1} \max_{y \,|\, \|y\|_2 \leq 1} x^T C y \\ &\leq 2\delta \|C\|_2 \end{aligned}$$

The above derivation follows from Assumption 2, and the bound reduces to the matrix norm using Cauchy-Schwartz inequality. □

Not surprisingly, the bound is independent of the local rewards given the transition structure of the agents. Thus the complexity of achieving a fixed approximation is polynomial in the problem size given a constant matrix C. However, the computation time is exponential in the size of C. Note also that the bound is additive.

Dimensionality Reduction

Experimental results and the offline bound suggest that the performance of the approach degrades significantly with the increasing dimensionality of Y. Therefore, it is very important to represent problems with very low dimensionality of Y in Eq. (2). While this is straightforward in some cases, it is not trivial in general. Thus, to make the algorithm more useful, we derive a procedure for automatic dimensionality reduction.

Intuitively, the dimensionality reduction removes those dimensions where $g(y)$ is constant, or almost constant. Interestingly, these dimensions may be recovered based on the eigenvectors and eigenvalues of $C^T C$.

Given a problem represented using Eq. (2), let F be a matrix whose columns are all the eigenvectors of $C^T C$ with eigenvalues greater than λ. Let G be a matrix with all the remaining eigenvectors as columns. Notice that together, the matrices span the whole space and are real, since $C^T C$ is a symmetric matrix. Assume without loss of generality that the eigenvectors are unitary. Let the number of columns of F be f. The original problem may now be compressed as follows.

$$\begin{aligned} \text{maximize} \quad & \tilde{f}(x, y_1, y_2, \hat{y}) = x^T C F y_1 + \\ & + r^T x + s^T [F, G][y_1; y_2] + t^T \hat{y} \\ \text{subject to} \quad & Ax \leq b \quad B_1 [F, G][y_1; y_2] + B_2 \hat{y} \leq c \end{aligned} \tag{5}$$

The following theorem quantifies the maximal error when using the compressed program.

Theorem 2. *Let f^* and $\tilde{f}^*$ be optimal solutions of Eqs. (2) and (5) respectively. Then:*

$$\epsilon = |f^* - \tilde{f}^*| \leq \sqrt{\lambda}.$$

Moreover, this is the maximal linear dimensionality reduction possible with this error without considering the constraint structure.

Proof. We first show that indeed the error is at most $\sqrt{\lambda}$ and that any linearly compressed problem with the given error has at least f dimensions. Using a mapping that preserves the feasibility of both programs, the error is bounded by:

$$\epsilon \leq |f(x, [F, G][y_1; y_2], \hat{y}) - \tilde{f}(x, y_1, y_2, \hat{y})| = \left|x^T C G y_2\right|.$$

Denote the feasible region of y_2 as Y_2. From the orthogonality of $[F, G]$, we have that $\|y_2\|_2 \leq 1$. Now we have:

$$\begin{aligned} \epsilon &\leq \max_{y_2 \in Y_2} \max_{x \in X} \left|x^T C G y_2\right| \leq \max_{y_2 \in Y_2} \|C G y_2\|_2 \\ &\leq \max_{y_2 \in Y_2} \sqrt{y_2^T G^T C^T C G y_2} \leq \max_{y_2 \in Y_2} \sqrt{y_2^T L y_2} \leq \sqrt{\lambda} \end{aligned}$$

The result follows from Cauchy-Schwartz inequality, the fact that $C^T C$ is symmetric, and Assumption 2. Matrix L denotes diagonal matrix of eigenvalues corresponding to eigenvectors of G.

Now, let H be an arbitrary matrix that satisfies the preceding error inequality for G. Clearly, $H \cap F = \emptyset$, otherwise there exists y, such that $\|CHy\|_2 > \epsilon$. Therefore, we have $|H| \leq n - |F| \leq |G|$, because $|H| + |F| = |Y|$. Here $|\cdot|$ denotes the number of columns of the matrix. □

In these bounds we use L_2-norm; an extension to a different norm is not straightforward. Note also that this dimensionality reduction technique ignores the constraint structure. When the constraints have some special structure, it might be possible to obtain an even tighter bound. As described in the next section, the dimensionality reduction technique generalizes the reduction *implicitly* used in (Becker *et al.* 2004).

Experimental Results

We now turn to an empirical analysis of the performance of the algorithm. For this purpose we use the *Mars rover* problem described earlier. We compare the presented algorithm with the original CSA and with a mixed integer linear program (MILP), derived for Eq. (2) as in (Petrik & Zilberstein 2007). Though, Eq. (2) can also be modeled as a linear complementarity problem (LCP) (Murty 1988), we do not evaluate that option experimentally because LCPs are closely related to MILPs (Rosen 1986). We expect these two formulations to exhibit similar performance. We also do not compare to any of the methods described in (Horst & Tuy 1996; Bennett & Mangasarian 1992) due to their very different nature and high complexity, and because some of these algorithms do not provide any optimality guarantees.

In our initial experiments, we applied the algorithm to problem instances with similar parameters to those used in (Becker *et al.* 2004). Each problem instance includes 6 sites. The time limit is 15 time units. The local reward for performing an experiment is selected uniformly from the interval [0.1,1.0] for each site and it is identical for both rovers. The global reward, received when both rovers perform an experiment on a shared site, is 0.5 of the local reward. The time required to perform an experiment is drawn from a discretized normal distribution with the mean uniformly chosen from 4.0-6.0. The variance is 0.4 of the mean. The experiments were performed with the following variants of shared sites: $\{2,3\}$, $\{2,3,4\}$, $\{1,2,3,4\}$,$\{1,2,3,4,5\}$.

In these experiments, the dimensionality of Y in Eq. (1) is $6 * 15 * 2 = 180$. This dimensionality may be reduced to be one per each shared site using the automatic dimensionality reduction procedure. Each dimension then represents the probability that an experiment on a *shared* site is performed regardless of the time. Therefore, the dimension represents the sum of the individual probabilities. The same compression was achieved in (Becker *et al.* 2004) using *compound events*. That is, each compound event represents the fact that an experiment is performed on some site regardless of the specific time.

Although the dimensionality reduction is intuitive in the rover problem, it is less obvious in other domains such as the multi-agent broadcast channel (Petrik & Zilberstein 2007). That problem involves two agents that share a communication channel. Each agent could have several messages to transmit stored in a buffer, but only one message can be transmitted at a time. In this case, the dimensionality can still be reduced to 3 regardless of the buffer length. Intuitively, the dimensions in this problem roughly approximate the sum of the probabilities that a message is sent, ignoring the number of messages in the buffer.

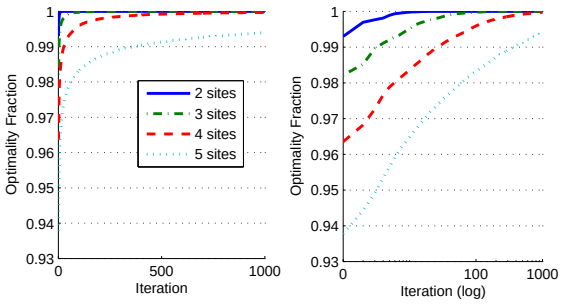

Figure 1: Mean approximation error across 200 problems for various numbers of shared sites.

Figure 1 shows the average error bound (ratio of the optimal value) achieved as a function of the number of iterations, that is, points for which $g(y)$ is evaluated. While it is possible that even the first iteration discovers the optimal solution, it still takes time to get a tight bound. The figure shows that within 1000 iterations, the solution quality is, on average, within 99.2% of the optimal reward. Calculating these 1000 iterations took about 20 seconds for the problems with 5 shared sites and about 12 seconds for problems with 2 shared sites.

In a very similar problem setup with at most 4 shared sites, CSA solved only 76% of the problems, and the longest solution took approximately 4 hours (Becker *et al.* 2004). In our case, all 200 problems with 4 shared sites were solved with 98.8% of the optimal solution in less than 20 seconds. And while CSA also offers very good anytime performance, it does not provide any guarantees of solution quality before it finds the optimal solution.

We also performed experiments with CPLEX – a state-of-the-art MILP solver. CPLEX was not able to solve any of the problems within 30 minutes, no matter how many of the sites were shared. The main reason for this is that it does not take any advantage of the limited interaction. Nevertheless, it is possible that some specialized MILP solvers may perform better.

To explore the scalability of the algorithm, we ran it on a large problem with 30 sites, 40 times steps, and 9 shared sites. Thus this problem had a total of 2402 states and 9 dimensions of Y. The problem was handcrafted to be difficult, because even some large problems may be trivial with only few best-responses. In 500 iterations, which took about 30 minutes, the algorithm found a solution provably within 84% of the optimal. We are not aware of any algorithm that can solve a decentralized MDP problem of this size optimally. In addition, note that the computation time depends only linearly on the number of states, but the error reduction degrades significantly with the increased dimensionality. As a result, the same problem with few interactions (shared sites) would be very easy to solve.

Interestingly, solving the same problem with more time steps, say 90, is much easier. In that case the optimal policy visits almost all the states, and therefore is the best response no matter what the other agent does. In addition, it is likely that significant performance gains can be achieved

by further tuning of the algorithm. Our experiments also showed that the algorithm's performance significantly depends on the implementation. For example, almost 3-fold speedup may be achieved using simplex instead of an interior point algorithm to solve program (4). We used linear programming algorithms from MOSEK.

Conclusion

We introduce a general algorithm for solving bilinear programs, especially ones that model decentralized stochastic planning problems. The algorithm is inspired by CSA. But unlike CSA, it provides error bounds. This is important because the algorithm is able to return near optimal results very rapidly, but it takes a long time to actually verify that a solution is optimal. In addition, we present a general procedure that reduces the dimensionality of the space. Reducing the dimensionality is crucial as it makes the algorithm much more useful in practice. The results we obtain represent a very significant improvement over the performance of CSA.

Applying the approach to more than two agents requires some modification. With three agents, for example, the mathematical program (2) must be modified to a multi-linear program with variables $x \in X, y \in Y, z \in Z$. Then, the best-response function $g(y, z)$ is not necessarily convex and therefore the method does not apply. This can be remedied by setting $\hat{Y} = Y \otimes Z$, and $\hat{y} = y \otimes z$. This transformation increases the dimensionality, but achieves the required piece-wise linearity. As a result, the algorithm is extensible to problems with multiple agents. In addition, (Becker *et al.* 2004) describes an extension of CSA to event-driven DEC-MDPs. Applying the presented algorithm to event-driven DEC-MDPs is also straightforward. However, the large dimensionality of the space limits the applicability of the algorithm. It is possible, however, that this may be remedied using the dimensionality reduction technique we introduced.

There are multiple ways in which the algorithm we presented can be improved. For example, it is possible to consider the actual shape of Y when approximating $g(y)$. Since in most cases the initial polytope does not exactly correspond to Y, it may be useful to restrict the search for the point with the largest error to only those that are in Y. This can be done by augmenting Eq. (4) with $Zz \in Y$. In addition, polytopes that do not overlap with Y can be discarded.

In future work, we will evaluate further ways to improve the performance of the algorithm and explore additional problems that can be modeled as bilinear programs. This includes many multi-agent coordination scenarios as well as other problems. It would be also interesting to further evaluate the performance of alternative methods for solving bilinear programs (Horst & Tuy 1996).

Acknowledgments

This work was supported in part by the Air Force Office of Scientific Research under Award No. FA9550-05-1-0254 and by the National Science Foundation under Grants No. IIS-0328601 and IIS-0535061. The findings and views expressed in this paper are those of the authors and do not necessarily reflect the positions of the sponsors.

References

Becker, R.; Zilberstein, S.; Lesser, V.; and Goldman, C. V. 2004. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research* 22:423–455.

Bennett, K. P., and Mangasarian, O. L. 1992. Bilinear separation of two sets in n-space. Technical report, Computer Science Department, University of Wisconsin.

Emery-Montemerlo, R.; Gordon, G.; Schneider, J.; and Thrun, S. 2004. Approximate solutions for partially observable stochastic games with common payoffs. In *Autonomous Agents and Multiagent Systems*.

Horst, R., and Tuy, H. 1996. *Global optimization: Deterministic approaches*. Springer.

Kim, Y.; Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2006. Exploiting locality of interaction in networked distributed POMDPs. In *AAAI Spring Symposium on Distributed Planning and Scheduling*.

Lovejoy, W. S. 1991. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research* 39:162–175.

Mangasarian, O. L. 1995. The linear complementarity problem as a separable bilinear program. *Journal of Global Optimization* 12:1–7.

Murty, K. G. 1988. *Linear complementarity, linear and nonlinear programming*. Helderman-Verlag.

Pang, J.-S.; Trinkle, J. C.; and Lo, G. 1996. A complementarity approach to a quasistatic rigid body motion problem. *Journal of Computational Optimization and Applications* 5(2):139–154.

Petrik, M., and Zilberstein, S. 2007. Average reward decentralized Markov decision processes. In *International Joint Conference on Artificial Intelligence*.

Pineau, J.; Gordon, G.; and Thrun, S. 2006. Point-based approximations for fast POMDP solving. *Jornal of Artificial Intelligence Research* 27(SOCS-TR-2005.4):335–380.

Puterman, M. L. 2005. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc.

Rosen, J. B. 1986. Solution of general LCP by 0-1 mixed integer programming. Technical Report Computer Science Tech. Report 8623, University of Minnesota, Minneapolis.

Seuken, S., and Zilberstein, S. 2005. Formal models and algorithms for decentralized control of multiple agents. Technical Report 2005-068, Computer Science Department, University of Massachusetts.

Seuken, S., and Zilberstein, S. 2007. Memory bounded dynamic programming for DEC-POMDPs. In *International Joint Conference on Artificial Intelligence*.

Vanderbei, R. J. 2001. *Linear Programming: Foundations and Extensions*. Second edition. Springer.