Mixed Integer Linear Programming For Exact Finite-Horizon Planning In Decentralized Pomdps

Raghav Aras, Alain Dutech, François Charpillet

Mixed Integer Linear Programming For Exact Finite-Horizon Planning In Decentralized Pomdps

Raghav Aras Alain Dutech François Charpillet
INRIA-Lorraine / Loria,
615 rue du Jardin Botanique,
54602 Villers-lès-Nancy, France
{aras, dutech, charp}@loria.fr

July 17, 2007

Abstract

We consider the problem of finding an n-agent joint-policy for the optimal finite-horizon control of a decentralized Pomdp (Dec-Pomdp). This is a problem of very high complexity (NEXP-hard in $n \geq 2$). In this paper, we propose a new mathematical programming approach for the problem. Our approach is based on two ideas: First, we represent each agent's policy in the sequence-form and not in the tree-form, thereby obtaining a very compact representation of the set of joint-policies. Second, using this compact representation, we solve this problem as an instance of combinatorial optimization for which we formulate a mixed integer linear program (MILP). The optimal solution of the MILP directly yields an optimal joint-policy for the Dec-Pomdp. Computational experience shows that formulating and solving the MILP requires significantly less time to solve benchmark Dec-Pomdp problems than existing algorithms. For example, the multi-agent tiger problem for horizon 4 is solved in 72 secs with the MILP whereas existing algorithms require several hours to solve it.

1 Introduction

In a *finite-horizon* Dec-Pomdp [1], a set of n agents cooperate to control a Markov decision process for κ steps under two constraints: *partial observability* and *decentralization*. Partial observability signifies that the agents are imperfectly informed about the state of the process during control. Decentralization signifies that the agents are *differently* imperfectly informed during the control. The agents begin the control of the process with the same, possibly imperfect, information about the state. During the control each agent receives *private* information about the state of the process, which he

cannot divulge to the other agents. The agents' private information can have an impact on what they collectively do. Thus, *before* the control begins, each agent must reason not only about the possible states of the process during the control (as in a Pomdp) but he must also reason about the information that *could* be held by other agents during the control. In effect, the agent must also reason about which policies the other agents would use. Partial observability and decentralization make Dec-Pomdps very difficult to solve. Finding an optimal solution to a Dec-Pomdp is NEXP-hard in the number of agents [1]; finding a locally optimal solution to a Dec-Pomdp is NP-hard in the size of the Dec-Pomdp problem (determined by κ and the sizes of the sets of joint-actions and joint-observations) [5].

1.1 Motivation for a new approach

The three existing exact algorithms DP [4], MAA* [11] and PBDP [10] are able to solve only very small Dec-Pomdps in reasonable time (2 agents, horizon ≤ 4, action and observation set sizes ≤ 3). Their lack of scalability is predictable from the negative complexity results. Therefore, the question is not so much whether these algorithms can be improved upon in the absolute, but rather if a relative improvement can be achieved. In other words, can we push the computational envelop a bit further on this problem? In this paper, we present a new approach based on *integer programming*, which does manifest a much superior performance in practice than the existing algorithms. For instance, through our approach, the multi-agent Tiger problem [8] for horizon 4 can be solved in 72 seconds as against the few hours required by the PBDP algorithm [10] (the only current algorithm able to solve this instance). Similarly, the MABC problem [4] for horizon 5 is solved in 25 seconds as against the 10^5 seconds required by PBDP. So we might tentatively answer in the positive to the above question. There is of course a more relevant reason for pushing this envelop. The three algorithms serve as a basis for approximate algorithms such as Approximate-DP [3] and MBDP [9], and these seem to scale to much longer horizons and to much larger problems. So, a more efficient exact algorithm is important from this perspective as well. We discuss this in more detail in the last section.

1.2 A new, mixed integer programming approach

Existing Dec-Pomdp algorithms represent an agent's policy as a *tree* and a joint-policy as a tuple of policy-trees. The size of the set of policy-trees of each agent is *doubly exponential* in the horizon. Hence, the set of joint-policies is doubly exponential in the horizon and exponential in the number of agents. This adversely impacts the space and time requirements of the algorithms. In our approach we discard the tree representation in favor of the *sequence-form* representation which was introduced in a seminal paper on computational game theory [6]. In the sequence-form, every finite-horizon *deterministic* policy of an agent can be represented as a *subset* of the set sequences of actions and observations of the agent. The problem of finding an optimal *deterministic* joint-policy is thus equivalent to the problem of finding for each agent a subset from a larger set. This problem thus becomes an instance of *combinatorial optimization* and we conceive a mixed integer linear program (MILP) to solve it. The key insight

of Koller's approach (and therefore of our approach) is that the size of the set of sequences from each subset is drawn is only exponential in the horizon and not doubly exponential in it, as is the case with the size of the set of policy trees. This allows us to formulate an MILP whose size is exponential in κ and n. For small problems such as MA-Tiger and MABC, it is feasible to represent the MILP in memory. Furthermore, and equally importantly, the constraints matrix of the MILP is *sparse*. The consequence of this is that in practice the MILP is solved very quickly (in the order of seconds). Thus, we have an effective method to compute an optimal deterministic finite-horizon joint-policy. Restricting attention to deterministic joint-policies does not limit the applicability of our approach in any way since in every finite-horizon Dec-Pomdp there exists at least one optimal joint-policy that is deterministic. It is also not evident that relaxing this restriction has any benefit. Implicitly, existing algorithms also restrict attention to deterministic joint-policies. In this paper 'policy' and 'joint-policy' shall mean deterministic policy and deterministic joint-policy respectively unless otherwise specified.

2 The finite-horizon Dec-Pomdp problem

A finite-horizon Dec-Pomdp problem is defined by the following elements. We are given N, a set of n agents and S, a set of states. The n agents in N are numbered from 1 to n. The states are numbered from 1 to $|S|$. For each ith agent, we are given A_i, the agent's set of actions and Ω_i, his set of observations. The cross-product $A_1 \times A_2 \ldots \times A_n$ is called the set of *joint-actions* and it is denoted by A. Similarly, the cross-product $\Omega_1 \times \Omega_2 \ldots \times \Omega_n$ is called the set of *joint-observations* and it is denoted by Ω. The joint-actions are numbered from 1 to $|A|$ and the joint-observations are numbered from 1 to $|\Omega|$. Then, we are given for each ath joint-action, the matrices T^a, Z^a and the vector R^a:

(a) $T^a_{ss'}$ is the probability of transitioning to the s'th state if the agents take the ath joint-action in sth state.

(b) $Z^a_{s'o}$ is the probability of the agents receiving the oth joint-observation and transitioning to s'th if they take the ath.

(c) R^a_s is the real-valued reward the agents obtain if they take the ath joint-action in the sth state.

We are given b_0, which represents the initial *belief state* and it is common knowledge amongst the agents. A belief state is a probability distribution over S. In a belief state b, the probability of the sth state is denoted by $b[s]$. Finally, we are given $\kappa \geq 1$, a finite number that is the *horizon* of the control. The control of the Dec-Pomdp is described as follows. At each step t of κ steps: the agents take a joint-action, they receive a joint-observation, they receive a common reward r_t, and the process transitions to a new belief state as a function of the previous belief state, the joint-action and the joint-observation. However, at each step, agents do not reveal to one another the actions they take and observations they receive at that step or at previous steps. Since an agent does not know the actions taken by the other agents and the observations received by the

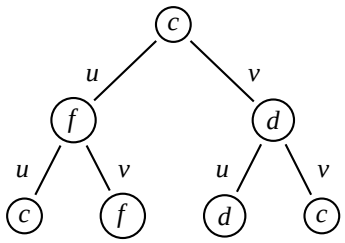

Figure 1: A 3-step policy ψ.

other agents during the κ steps, at each step he takes actions strictly as a function of the actions he has taken previously and observations he has received previously. This function is called his *policy*. To control the Dec-Pomdp for κ steps, each agent requires a κ-step policy, henceforth written as κ-policy. The tuple of the agents' policies forms a *joint-policy*. An optimal joint-policy is one which maximizes $E(\sum_{t=1}^{\kappa} r_t)$, the sum of expected rewards the agents obtain for the κ steps.

2.1 Policy in the tree-form

The canonical representation of a policy, used in existing Dec-Pomdp algorithms, is the *tree-form*. In this form, a κ-policy of the ith agent can be represented as a rooted tree with κ levels in which each non-terminal node has $|\Omega_i|$ children. This tree is called a κ-policy-tree. Each node is labeled by an action to take and each edge is labeled by an observation that may occur. Using a policy-tree, during the control, the agent follows a path from the root to a leaf depending on the observations he receives. An example of a policy-tree is shown in Figure 1. The number of nodes in a κ-policy-tree of the ith agent is $\frac{|\Omega_i|^\kappa - 1}{|\Omega_i| - 1}$. It is thus exponential in κ. For example, with $|\Omega_i| = 2$, a 3-policy-tree, as the one shown in Figure 1, has $\frac{2^3-1}{2-1} = 7$ nodes. The set of κ-policy-trees of the ith agent is the set of all the $\frac{|\Omega_i|^\kappa - 1}{|\Omega_i| - 1}$ sized permutations of the actions in A_i. Therefore, the size of the set of κ-policy-trees of the ith agent is $|A_i|^{\frac{|\Omega_i|^\kappa - 1}{|\Omega_i| - 1}}$, doubly exponential in κ.

3 Policy in the sequence-form

The double exponentiality associated with the set of policy-trees can be avoided by using the *sequence-form* representation of a policy. We begin a description of this representation by defining a sequence.

Definition 1 *A* sequence of length t of the ith agent *is an ordered list of $2t$ - 1 elements, $t \geq 1$, in which the elements in odd positions are actions from A_i and those in even positions are observations from Ω_i.*

Thus, in a sequence of length t there are t actions and t - 1 observations. The shortest possible sequence is of length 1, which consists of just an action and no observations.

We denote the set of all possible sequences of length t which can be conceived from A_i and O_i by $\mathcal{S}_i^t$. We denote the set $\mathcal{S}_i^1 \cup \mathcal{S}_i^2 \cup \ldots \mathcal{S}_i^\kappa$ by $\mathcal{S}_i$. We shall now see how a κ-policy can be represented as a set of sequences, or more precisely as a subset of $\mathcal{S}_i$. Assume that $\kappa = 3$ and the policy-tree ψ shown in Figure 1 is of the ith agent. Starting from the root-node and descending down the edges of the tree, we can enumerate the sequences of this tree. The first sequence we obtain is in the root-node itself, the sequence consisting of the action c and no observations. This is a sequence of length 1. Then, going down the edge labeled by u from the root-node, we come to the node labeled by the action f. At this point, we obtain a second sequence cuf, which is of length 2. It has two actions and one observation. Similarly, taking the other edge from the root-node, we come to the node labeled by d and obtain a third sequence cvd, also of length 2. When all the leaves of the tree have been visited, the set of sequences we obtain is,

$$\mathcal{S}(\psi) = \Big\{\, c, cuf, cvd, cufuc, cufvf, cvdud, cvdvc \,\Big\}$$

This set contains 1 sequence of length 1, 2 sequences of length 2 and 4 sequences of length 3 to give a total of 7 sequences corresponding to the 7 nodes in ψ. It is evident that the set $\mathcal{S}(\psi)$ is equivalent to the policy-tree ψ. That is, *given* the set $\mathcal{S}(\psi)$, the agent can use it as a 3-step policy. As this simple exercise shows, any finite-step policy can be written as a finite set of sequences. Now, $\mathcal{S}(\psi)$ *is a subset of* $\mathcal{S}_i$, the set of all possible sequences of lengths less than or equal to 3, and so is every 3-policy of the ith agent. Thus, for any given value of κ, every κ-policy of the ith agent is a subset of $\mathcal{S}_i$. This is main idea of the sequence-form representation of a policy.

3.1 Policy as a vector

We can streamline the subset-set relationship between a κ-policy and $\mathcal{S}_i$ by representing the former as a vector of binary values. Let the sequences in $\mathcal{S}_i$ be numbered from 1 to $|\mathcal{S}_i|$. Since every κ-policy of the ith agent is a subset of $\mathcal{S}_i$, every sequence in $\mathcal{S}_i$ is either in the policy or it is not. Thus a κ-policy of the ith agent can be represented as a $|\mathcal{S}_i|$-vector of binary values 0 or 1, such that if the jth sequence in $\mathcal{S}_i$ is in the policy then the jth element of the vector equals 1 and if it is not, then the jth element of the vector equals 0. Let the set of $|\mathcal{S}_i|$-vectors of binary values 0 or 1 be denoted by $\mathcal{X}_i$. Thus every κ-policy of the ith agent is member of the set $\mathcal{X}_i$. Let p be the jth sequence in $\mathcal{S}_i$. For a vector $x_i \in \mathcal{X}_i$, value of the jth element in x_i shall be conveniently represented as $x_i[p]$.

3.2 Policy constraints of the ith agent

Thus, every κ-policy of the ith agent is a member of $\mathcal{X}_i$. The inverse of this is course untrue; not every member of $\mathcal{X}_i$ is a κ-policy. We therefore need to define which vectors in $\mathcal{X}_i$ can represent a κ-policy. We shall give a more general definition, one that includes stochastic policies as well as deterministic policies. We shall in fact define which vectors in $\mathbb{R}^{|\mathcal{S}_i|}$ represent a κ-step policy, be it a stochastic policy or a deterministic one. The definition takes the form of a system of linear equations which must be

satisfied by a vector in $\mathbb{R}^{|\mathcal{S}_i|}$ if it is to represent a κ-policy. Given a sequence p, an action a and an observation o, let poa denote the sequence obtained on appending o and a to the end of p. Let $\mathcal{S}_i^{'}$ denote the set $\mathcal{S}_i^1 \cup \mathcal{S}_i^2 \cup \ldots \mathcal{S}_i^{\kappa-1}$.

Definition 2 *Let $|\mathcal{S}_i| = z$. A vector $w \in \mathbb{R}^z$ is a κ-step, possibly stochastic, policy of the ith agent if,*

$$\sum_{a \in A_i} w[a] = 1 \tag{1}$$

$$w[p] - \sum_{a \in A_i} w[poa] = 0, \quad \forall\, p \in \mathcal{S}_i^{'}, o \in \Omega_i \tag{2}$$

$$w \geq 0 \tag{3}$$

We call the system of linear equations (1)-(3) the *policy constraints of the ith agent*. Policy constraints recreate the tree structure of a policy. They appear in a slightly different form, as Lemma 5.1 in [6]. We can write the policy constraints in the matrix form as $C_i w = b_i$, $w \geq 0$, where C_i is the matrix of the coefficients of the variables in the equations (1)-(2) and b_i is a vector of appropriate length whose first element is 1 and the remaining elements are 0, representing the r.h.s of the equations. Note that it is implicit in the above definition that the value of each element of w is constrained to be in the interval [0,1]. Hence, we can define a deterministic κ-policy of the ith agent as follows.

Definition 3 *A vector $x_i \in \mathcal{X}_i$ is a κ-policy of the ith agent if $C_i x_i = b_i$.*

We shall call a policy represented as a vector as a *policy-vector* just to distinguish it from a policy-tree. The representation of a policy as a policy-vector is in fact the sequence-form representation we have been alluding to. Given a vector from $x_i \in \mathcal{X}_i$ which satisfies the policy constraints, the agent can use it just as he would use as a policy-tree without requiring any additional book-keeping. Let choosing a sequence mean taking the last action in the sequence. In using x_i, at the first step, he chooses the action a such that $x_i[a] = 1$. There will be only one such action. Then on receiving an observation, say o, he chooses the sequence aoa' such that $x_i[aoa'] = 1$. Again there will be only one such sequence. In general, if at step t he has chosen the sequence p and then received the observation o, then he chooses the unique sequence poa'' such that $x_i[poa''] = 1$ at the $(t+1)$th step. Thus, at each step, the agent must know the sequence of actions he has taken and the sequence of observations he has received till that step in order to know which action to take according to x_i. This requirement is called *perfect recall* in game theory, and it is implicit in the use of a policy-tree.

3.3 Advantage of the sequence-form representation

The size of $\mathcal{S}_i^t$ is $|A_i|^t|\Omega_i|^{t-1}$. The size of $\mathcal{S}_i$ is thus $\sum_{t=1}^{\kappa} |A_i|^t|\Omega_i|^{t-1}$, exponential in κ. Since every κ-policy is in theory available if the set $\mathcal{S}_i$ is available, the latter serves as a *search space* for κ-policies of the ith agent. The good news is of course that this search space is only exponential in κ. This compares favorably with the search space represented by the set of κ-policy-trees which is doubly exponential in κ. We thus

have at our disposal an exponentially smaller space in which to search for an agent's policy. More precisely, to find a κ-policy of the ith agent, we need to set up and solve the system the policy constraints. The number of equations in this system is $c_i = 1 + \sum_{t=1}^{\kappa-1} |A_i|^t |\Omega_i|^t$. C_i is thus a $c_i \times |\mathcal{S}_i|$ matrix. Now notice that C_i is a *sparse* matrix, that is, it has only a very small number of nonzero entries per row or column, while most of its entries are 0s. In C_i, the number of nonzero entries per row is only $1 + |A_i|$, and it is constant per row. Sparse matrices are typically easier to solve that dense matrices of the same size. The relatively small size of C_i and its sparsity combine to form a relatively efficient method to find a κ-policy of the ith agent.

4 Value of a Joint-policy

The agents control the the finite-horizon Dec-Pomdp by a κ-step joint-policy, henceforth written as a κ-joint-policy. A joint-policy is just the tuple formed by the agents' individual policies. Thus, a κ-joint-policy is an n-tuple of κ-policies. A κ-joint-policy may be an n-tuple of κ-policy-trees or it may be an n-tuple of κ-policy-vectors. Given a joint-policy π in either representation, the policy of the ith agent in it shall be denoted by π_i. A joint-policy is evaluated by computing its *value*. The value of a joint-policy represents the sum of expected rewards the agents obtain if it is executed starting from the given initial belief state b_0. The value of a joint-policy π shall be denoted by $\mathcal{V}(\pi)$.

4.1 Value of a joint-policy as an n-tuple of policy-trees

Given a t-policy σ of an agent, $t \leq \kappa$, let $a(\sigma)$ denote the action in the root node of σ and let $\sigma(o')$ denote the sub-tree attached to the root-node of σ into which the edge labeled by the observation o' enters. Furthermore, given a t-joint-policy π, let $a(\pi)$ denote the joint-action ($a(\pi_1)$, $a(\pi_2)$, ..., $a(\pi_n)$) and given a joint-observation o, let $\pi(o)$ denote the $(t-1)$-joint-policy ($\pi_1(o_1)$, $\pi_2(o_2)$, ..., $\pi_n(o_n)$). Now let π be a κ-joint-policy which is an n-tuple of κ-policy trees. The value of π is expressed in terms of the κ-step *value-function* of the Dec-Pomdp denoted by V^κ as follows,

$$\mathcal{V}(\pi) = \sum_{s \in S} b_0[s] V^\kappa(s, \pi) \tag{4}$$

in which V^κ is expressed recursively as,

$$V^\kappa(s, \pi) = R_s^{a(\pi)} + \sum_{o \in \Omega} \sum_{s' \in S} T_{ss'}^{a(\pi)} Z_{s'o}^{a(\pi)} V^{\kappa-1}(s', \pi(o)) \tag{5}$$

For $t = 1$, $V^t(s, a) = R_s^a$. An *optimal* κ-joint-policy is one whose value is the maximum.

4.2 Value of a joint-policy as an n-tuple of policy-vectors

The value of a κ-joint-policy that is an n-tuple of policy-vectors is expressed in terms of the values of its *joint-sequences*. A joint-sequence is defined analogously to a sequence.

Definition 4 *A* joint-sequence of length t *is an ordered list of 2t - 1 elements,* $t \geq 1$*, in which the elements in odd positions are joint-actions from* A *and those in even positions are joint-observations from* Ω*.*

Equivalently, we can also define a joint-sequence of length t as an n-tuple of sequences of length t. Given a joint-sequence q, the sequence of the ith agent in q shall be denoted by q_i. The set of joint-sequences of length t, denoted by $\mathcal{S}^t$, is thus the cross-product set $\mathcal{S}_1^t \times \mathcal{S}_2^t \ldots \times \mathcal{S}_n^t$. Given a joint-sequence q of length t, the $(j \leq t)$th joint-action in it shall be denoted by a_q^j and the $(h < t)$th joint-observation in it shall be denoted by o_q^h. We now define the value of a joint-sequence.

4.3 Joint-sequence value

The value of a joint-sequence q of length t, denoted by $\nu(q)$, is *independent* of any joint-policy. It is simply a property of the Dec-Pomdp model. It is a product of two quantities: $\rho(q)$, the probability of q occurring and $\mathcal{R}(q)$, the sum of expected rewards the joint-actions in q obtain:

$$\nu(q) = \rho(q)\mathcal{R}(q) \tag{6}$$

These quantities are defined and computed as follows. $\rho(q)$ is the probability,

$$\begin{aligned}
\rho(q) &= \Pr(o_q^1, o_q^2, \ldots, o_q^{t-1} | b_0, a_q^1, a_q^2, \ldots, a_q^{t-1}) && (7)\\
&= \prod_{j=1}^{t-1} \Pr(o_q^j | b_0, a_q^1, o_q^1, \ldots, o_q^{j-2}, a_q^{j-1}) && (8)\\
&= \prod_{j=1}^{t-1} \Pr(o_q^j | b_{j-1}^q) && (9)
\end{aligned}$$

where b_{j-1}^q is a belief state which, if computed as follows, serves a *sufficient statistic* for the joint-sequence (a_q^1, o_q^1, …, o_q^{j-2}, a_q^{j-1}). Let o denote o_q^j and a denote a_q^j. Let b_{j-1}^q be given. Then,

$$\Pr(o | b_{j-1}^q) = \sum_{s \in S} \sum_{s' \in S} b_{j-1}^q[s] T_{ss'}^a Z_{s'o}^a \tag{10}$$

and b_j^q is given as, (for each $s \in S$),

$$b_j^q[s] = \frac{\sum_{s' \in S} b_{j-1}[s'] T_{s's}^a Z_{so}^a}{\Pr(o | b_{j-1}^q)} \tag{11}$$

Thus $\rho(q)$ is computed as follows. We assign b_0 to b_0^q. For each non-zero $j < t$, we calculate $\Pr(o_q^j | b_{j-1}^q)$ using eq. (10). If for any t, we find that $\Pr(o_q^j | b_{j-1}^q)$ is 0, we set $\rho(q)$ to 0 and terminate. On the other hand, whenever $\Pr(o_q^j | b_{j-1}^q) > 0$, we compute $b_j^q[s]$ for each state $s \in S$ using eq.(11) and continue. The quantity $\mathcal{R}(q)$ is simply the sum of the expected rewards the joint-actions in q obtain in the belief states b_j^qs.

Assigning, as before, b_0 to b_0^q, and denoting a_q^j by a,

$$\mathcal{R}(q) = \sum_{j=1}^{t} \sum_{s \in S} b_{j-1}^q[s] R_s^a \tag{12}$$

Recall that fundamentally, a κ-policy is just a set of sequences of different lengths. Given a policy σ of the ith agent let the subset of σ containing sequences of length t be denoted by σ^t. Then given a joint-policy π, the set of joint-sequences of length t of π is simply the set $\pi_1^t \times \pi_2^t \times \ldots \times \pi_n^t$. Note that if a joint-sequence q is in π, then $\prod_{i=1}^{n} \pi_i[q_i] = 1$ and if it is not, then $\prod_{i=1}^{n} \pi_i[q_i] = 0$. We can now define the value of a κ-joint-policy π in terms of the values of the joint-sequences. In particular, we need consider only joint-sequences of length κ. Thus,

$$\mathcal{V}(\pi) = \sum_{q \in \mathcal{S}^\kappa} \nu(q) \prod_{i=1}^{n} \pi_i[q_i] \tag{13}$$

The derivation of eq. (13) from eq. (4) is quite straightforward and is omitted.

5 Algorithm

We shall now describe a mixed integer linear program (MILP) that finds an optimal κ-joint-policy. We start our description with the following naive mathematical program (MP) which just implements the definition of an optimal κ-joint-policy. This implies finding for each agent i a vector $\overline{x}_i \in \mathbb{R}^{|\mathcal{S}_i|}$ which satisfies the policy constraints of the ith agent and the quantity $\mathcal{V}(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$ is maximized. Letting $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$, the naive MP, denoted by `MP-Dec`, is as follows,

$$\text{maximize} \quad f(\overline{x}) \equiv \sum_{q \in \mathcal{S}^\kappa} \nu(q) \prod_{i=1}^{n} \overline{x}_i[q_i] \tag{14}$$

$$\text{s.t.} \quad \forall\, i \in N\text{:} \quad C_i \overline{x}_i = b_i \tag{15}$$

$$\overline{x}_i \geq 0 \tag{16}$$

An optimal solution to `MP-Dec` would yield an optimal (possibly, stochastic) κ-joint-policy $\overline{x}$. However, since $f(\overline{x})$ is a nonconcave, nonlinear function, not only is solving `MP-Dec` NP-hard, but more importantly, it is also not possible to guarantee finding a globally optimal κ-joint-policy. A simple fix to get rid of the nonconcave nonlinear $f(\overline{x})$ in `MP-Dec` is to somehow linearize $f(\overline{x})$, that is, to transform it into a linear function. Linearization of $f(\overline{x})$ is achieved by using more variables and more constraints in addition to those in `MP-Dec`. The additional variables pertain to joint-sequences and the additional constraints are required to relate the variables of joint-sequences to those of sequences. The linearization of $f(\overline{x})$ takes place in three steps. At the end of the three steps, `MP-Dec` is converted to an integer linear program (ILP) on which the proposed MILP is based.

5.1 Linearization of $f(\overline{x})$: step 1

The simple idea in linearizing a nonlinear function is to use a variable for each nonlinear term that appears in the function. In the case of $f(\overline{x})$, the nonlinear terms are, for each joint-sequence q of length κ, $\prod_{i=1}^{n} \overline{x}_i[q_i]$. Therefore, to replace the nonlinear terms in $f(\overline{x})$, we need to use a variable for every joint-sequence q of length κ. Let $\overline{y}[q] \geq 0$ be the variable for q and let,

$$f(\overline{y}) \equiv \sum_{q \in \mathcal{S}^\kappa} \nu(q) \overline{y}[q] \tag{17}$$

So the first step in linearizing $f(\overline{x})$ is to change the objective in `MP-Dec` to $f(\overline{y})$ and introduce the $|\mathcal{S}^\kappa|$-vector $\overline{y} \geq 0$ of variables in it. We denote this modified MP by `MP1-Dec`.

5.2 Linearization of $f(\overline{x})$: step 2

Once the objective is changed to $f(\overline{y})$, we need to relate the variables representing joint-sequences ($\overline{y}$) to those representing agents' sequences (the $\overline{x}_i$ vectors). In other words, we need to add the following constraints to `MP1-Dec`,

$$\prod_{i=1}^{n} \overline{x}_i[q_i] = \overline{y}[q], \quad \forall\, q \in \mathcal{S}^\kappa \tag{18}$$

But the constraints (18) are *nonconvex*. So, if they are added to `MP1-Dec`, it would amount to maximizing a linear function under nonconvex, nonlinear constraints, and again we would not have any guarantee of finding the globally optimal solution. We therefore must also linearize these constraints. We shall do this in this step and the next. Suppose that ($\overline{x}_1$, $\overline{x}_2$, ..., $\overline{x}_n$) is a solution to `MP1-Dec`. Then, for each joint-sequence q of length κ, $\prod_{i=1}^{n} \overline{x}_i[q_i]$ takes a value in [0,1]. In other words, it can take an infinite number of values. We can limit the values it can take by requiring that the vectors $\overline{x}_i$ be vectors of *binary* variables, 0 or 1. Moreover, since we want $\prod_{i=1}^{n} \overline{x}_i[q_i]$ to equal $\overline{y}[q]$, but want to avoid the constraints (18), we should also require that each $\overline{y}$ variable be a binary variable. Thus, the second step in linearizing $f(\overline{x})$ is to add the following constraints to `MP1-Dec`:

$$\overline{x}_i[p] \in \{0,1\}, \quad \forall\, i \in N, \forall\, p \in \mathcal{S}_i \tag{19}$$

$$\overline{y}[q] \in \{0,1\}, \quad \forall\, q \in \mathcal{S}^\kappa \tag{20}$$

Note that with these constraints in `MP1-Dec`, $\overline{x}_i$ would represent a deterministic κ-policy of the ith agent. Constraints (19)-(20) are called *integer constraints*. We denote the MP formed by adding integer constraints to `MP1-Dec` by `MP2-Dec`.

5.3 Linearization of $f(\overline{x})$: step 3

This is key step in the linearization. The number of sequences of length κ in a κ-policy of the ith agent is $\tau_i = |\Omega_i|^{\kappa-1}$. Hence the number of joint-sequences of length κ

in a κ-joint-policy $\tau = \prod_{i=1}^{n} \tau_i$. Let, $\tau_{-i} = \frac{\tau}{\tau_i}$. Now suppose ($\overline{x}_1$, $\overline{x}_2$, ..., $\overline{x}_n$) is a solution to MP2-Dec. Each $\overline{x}_i$ is a κ-step deterministic policy of the ith agent. The κ-joint-policy formed by them is also deterministic. If for a sequence p of length κ, $\overline{x}_i[p] = 1$, then it implies that for exactly τ_{-i} joint-sequences q of length κ in which the sequence of the ith agent is p, $\prod_{i=1}^{n} \overline{x}_i[q] = 1$. On the other hand, if $\overline{x}_i[p] = 0$, then for each joint-sequence q in which the sequence of the i agent is p, $\prod_{i=1}^{n} \overline{x}_i[q] = 0$. This can be represented mathematically as,

$$\sum_{q \in \mathcal{S}^\kappa : q_i = p} \prod_{j=1}^{n} \overline{x}_j[q_j] = \tau_{-i} \overline{x}_i[p], \quad \forall\, i \in N, \forall\, p \in \mathcal{S}_i^\kappa \tag{21}$$

The set of equations (21)is true for every κ-step deterministic joint-policy, and it allows us to linearize the constraints (18). All we have to do is to add the following set of linear constraints to MP2-Dec,

$$\sum_{q \in \mathcal{S}^\kappa : q_i = p} \overline{y}[q] = \tau_{-i} \overline{x}_i[p], \quad \forall\, i \in N, \forall\, p \in \mathcal{S}_i^\kappa \tag{22}$$

If these constraints are added to MP2-Dec then the following holds,

$$\prod_{j=1}^{n} \overline{x}_j[q_j] = \overline{y}[q], \quad \forall\, q \in \mathcal{S}^\kappa \tag{23}$$

because the r.h.s. of their corresponding equations are equal. Thus, we have achieved the linearization of the constraints (18) and therefore of $f(\overline{x})$. We shall call the constraints (22) as the *joint-policy constraints*. The MP obtained on adding the joint-policy constraints to MP2-Dec gives us the integer linear program ILP-Dec, on which the mixed ILP (MILP), the main contribution of this paper, is based We give ILP-Dec below for the sake of completeness.

5.4 Integer linear program ILP-Dec

1. Variables:
   (a) A $|\mathcal{S}^\kappa|$-vector of variables, $\overline{y}$.
   (b) For each agent $i \in N$, an $|\mathcal{S}_i|$-vector of variables, $\overline{x}_i$.
2. Objective:

$$\text{maximize} \qquad f(\overline{y}) \equiv \sum_{q \in \mathcal{S}^\kappa} \nu(q) \overline{y}[q] \tag{24}$$

3. Constraints: for each agent $i \in N$,
   (a) Policy constraints:

$$\sum_{a_i \in A_i} \overline{x}_i[a_i] = 1 \tag{25}$$

$$\forall\, t \in \{1, 2, \ldots, \kappa - 1\}, \forall\, p \in \mathcal{S}_i^t, \forall\, o_i \in \Omega_i,$$

$$\overline{x}_i[p] - \sum_{a \in A_i} \overline{x}_i[poa] = 0 \tag{26}$$

(b) Joint-policy constraints: for each $p \in \mathcal{S}_i^\kappa$,

$$\sum_{q \in \mathcal{S}^k : q_i = p} \overline{y}[q] = \tau_{-i} \overline{x}_i[p] \tag{27}$$

4. Integer constraints:

$$\overline{x}_i[p] \in \{0, 1\}, \qquad \forall\, i \in N, \forall\, p \in \mathcal{S}_i \tag{28}$$

$$\overline{y}[q] \in \{0, 1\}, \qquad \forall\, q \in \mathcal{S}^\kappa \tag{29}$$

We thus have the following result.

Theorem 1 *An optimal solution ($\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$) to* ILP-Dec *yields an optimal κ-joint-policy for the given Dec-Pomdp.* (Proof is omitted)

5.5 Mixed integer linear program MILP-Dec

An ILP is so called because it is an LP whose variables are constrained to take integer values. In ILP-Dec, each variable can be either 0 or 1. The principle method for solving an integer linear program is *branch and bound*. So when solving ILP-Dec, a tree of LPs is solved in which each LP is identical to the ILP-Dec but in which the integer constraints are replaced by non-negativity constraints (i.e., all the variables are allowed to take real values greater than or equal to 0). In general, the lesser the number of integer variables in an LP, the faster a solution will be obtained. Therefore it is desirable to minimize the number of integer variables in an LP. An LP in which some variables are allowed to take real values while the remaining ones are constrained to be integers is called a mixed ILP (MILP). Thus, an MILP may be solved faster than an ILP of the same size. We say that an MILP is equivalent to an ILP if every solution to the MILP is also a solution to the ILP. An MILP that is equivalent to ILP-Dec can be conceived as follows. Let this MILP be denoted by MILP-Dec. Let MILP-Dec be identical to ILP-Dec in all respects except the following: in each vector $\overline{x}_i$, only those variables representing sequences of length κ be constrained to take integer values 0 or 1; *all* the other variables in each $\overline{x}_i$ and *all* the variables in the vector $\overline{y}$ be allowed to take real values greater than or equal to 0. Due to the equivalence, we have the following result.

Theorem 2 *An optimal solution ($\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$) to* MILP-Dec *yields an optimal κ-joint-policy for the given Dec-Pomdp*

The proof of this theorem (and of the claim that MILP-Dec is equivalent to ILP-Dec) is omitted due to lack of space. The discussion henceforth applies to ILP-Dec as well.

6 Improving MILP-Dec

We now discuss two heuristics for improving the space and time requirement of formulating and solving MILP-Dec.

6.1 Identifying dominated sequences

The number of variables required in the MILP-Dec can be minimized by using variables for only those sequences of each agent that are not *dominated*. Dominated sequences need not be represented in the MILP-Dec because there always exists an optimal κ-joint-policy in which none of the policies contains a dominated sequence. We first define dominated sequences of length κ. Given sequences p and p' of length κ of the ith agent, p' shall be called a *co-sequence* of p if it is identical to p except for its last action. Let $\mathcal{C}(p)$ denote the set of co-sequences of p. Then, p is said to be dominated if there exists a probability distribution θ over $\mathcal{C}(p)$, such that for every joint-sequence q of length κ in which the sequence of the ith agent is p, the following is true:

$$\nu(q) \leq \sum_{p' \in \mathcal{C}(p)} \theta(p')\nu(q') \tag{30}$$

in which q' = (q_1, ..., q_{i-1}, p', q_{i+1}, ..., q_n). Dominated sequences of length κ can be identified through *iterated elimination*. Identifying sequences of lengths less than κ is easier. A sequence p of length t is a *descendant* of a sequence p'' of length $j < t$ if the first j actions and j - 1 observations in p are identical to the j actions and j - 1 observations in p''. A sequence p'' of length j is dominated if every descendant of p'' is dominated. So, for each agent, we first identify dominated sequences of length κ, and then working backwards, we identify dominated sequences of lengths less than κ. Note that *if* dominated sequences are not represented by variables in MILP-Dec, then in each joint-policy constraint the $=$ sign must be replaced by the $\leq$ sign. The MILP that results when dominated sequences of all the agents are not represented by variables in MILP-Dec and the above modifications are made shall be denoted by MILP-Pr-Dec.

6.2 Adding bounds into MILP-Dec

The MILP solver can be guided in its path selection in the tree of LP problems or made to terminate as early as possible by providing lower and/or upper bounds on the objective function. In this paper, we wish to illustrate the importance of integrating bounds in MILP-Dec, and so we have used rather loose bounds. Given $\mathcal{V}(t)$, the value of an optimal t-joint-policy, a lower bound on the value of the optimal $(t+1)$-joint-policy is,

$$\ell = \mathcal{V}(t) + \max_{a \in A} \min_{s \in S} R_s^a \tag{31}$$

For an upper bound, the value u of an optimal κ-step policy of the Pomdp corresponding to the Dec-Pomdp can be used. This value can be determined by the linear program (32)-(35) which also finds the optimal κ-step policy for the Pomdp. Let $\mathcal{S}^t$ denote

Algorithm	MABC			MA-tiger	
κ	3	4	5	3	4
MILP-Dec	0.86	900	–	3.7	·
MILP-Dec(u)	1.03	907	–	3.5	·
MILP-Dec(ℓ)	0.93	900	–	4.9	72
MILP-Pr-Dec	0.84	80	·	6.4	·
MILP-Pr-Dec(u)	0.93	10.2	25	6.2	·
MILP-Pr-Dec(ℓ)	0.84	120	·	7.6	175
DP	5	10^3			
MAA*	t_3	t_4		t_3	t_4
PBDP	1.0	2.0	10^5	t_3	t_4
DP-JESP				0	0.02
Approx-DP				0.05	1.0
MBDP	0.01	0.01	0.02	0.46	0.72

Table 1: Comparison of the runtimes in seconds of Dec-Pomdp algorithms. t_3 denotes several seconds and t_4 denotes several hours. "·" denotes a time-out of 30 minutes, "-" denotes insufficient memory and blank denotes that the application of the concerned algorithm to the concerned problem does not appear in the literature.

the set of joint-sequences of length t. Let qoa denote the joint-sequence obtained on appending the joint-observation o and the joint-action a to the joint-sequence q.

$$\text{maximize} \quad u = \sum_{q \in \mathcal{S}^\kappa} y[q] \quad \text{s.t.:} \tag{32}$$

$$\sum_{a \in A} y[a] = 1 \tag{33}$$

$$y[q] - \sum_{a \in A} y[qoa] = 0, \forall\, t < \kappa,\, q \in \mathcal{S}^t,\, o \in \Omega \tag{34}$$

$$y \geq 0 \tag{35}$$

A bound is added to MILP-Dec by adding a constraint. The constraint $f(\overline{y}) \geq \ell$ is added for adding the lower bound and the constraint $f(\overline{y}) \leq u$ is added for adding the upper bound.

7 Experiments

We formulated the MABC problem and MA-tiger problem as MILPs, and solved it using the ILOG Cplex 10 solver on an Intel P4 machine with 3.40 gigahertz processor speed and 2.0 GB ram. The runtime in seconds of MILP-Dec and MILP-Pr-Dec for different values of κ is shown in Table 1. In the first column, a parenthesis, if present indicates which bound is used. The runtime includes the time taken to identify dominated sequences and compute the bound (for e.g., solve the LP for the Pomdp),

where applicable. We have listed the runtime of existing exact and approximate dynamic programming Dec-Pomdp algorithms as reported in the literature. The three exact algorithms are DP, MAA* and PBDP. The approximate algorithms are DP-JESP [8], Approximate-DP and MBDP. As far as dominated sequences are concerned, the MABC problem had about 75% dominated sequences per agent for $\kappa = 5$, while MA-Tiger had *no* dominated sequences for any horizon.

8 Discussion and future directions

In this paper we have introduced a new exact algorithm that for solving finite-horizon Dec-Pomdps. The results from Table 1 show a clear advantage of the MILP algorithms over existing exact algorithm for the longest horizons considered in each problem. We now point out three directions in which this work can be extended.

Approximate algorithm: Our approach could be a good candidate to construct an approximate algorithm. For instance, if `MILP-Dec` or one of its variant is able to solve a problem optimally for horizon κ very quickly, then it can be used as a ratchet for solving approximately for longer horizons in divisions of κ steps. Our initial experiments with this simple method on the MABC and MA-Tiger problems indicate that it may be comparable in runtime and value of the joint-policy found with current approximate algorithms for solving long horizons (50,100). This is particularly useful when the Dec-Pomdp problem cycles back to the original state in a few steps. In the MA-Tiger problem, for example, upon the execution of the optimal 3-step joint-policy, denoted by σ^3, the process returns back to its initial belief state. The value of σ^3 is 5.19. So we can perpetually execute σ^3 to get in m steps, a total expected reward of ($5.19m/3$). Now, the value of σ^2, the optimal 2-step joint-policy is -2. For controlling the MA-Tiger problem for m steps, we may either (a) execute σ^3 $m/3$ times or (b) σ^2 $m/2$ times. The loss for doing (b) instead of (a) would be $2.73/m$ per step. This can be made arbitrarily high by changing the reward function. In other words, finding σ^3 is much more important that finding σ^2. We can arrange for a similar difference in quality between σ^4 and σ^3; and `MILP-Dec` is able to find σ^4 in 72 secs while other algorithms take hours. Thus, the role of an exact, fast algorithm, such as ours, may prove crucial even for very small problems.

Dynamic programming: In formulating `MILP-Dec` we are required to first generate the set $\mathcal{S}_i^\kappa$ for each agent i. The size of this set is exponential in κ. The generation of this set acts as the major bottleneck for formulating `MILP-Dec` in memory. However, we can use dynamic programming to create each set $\mathcal{S}_i^\kappa$ incrementally in a backward fashion. Such a procedure does not require the knowledge of b_0 and it is based on the same principle as the DP algorithm. In brief, the procedure is explained as follows. For each nonzero $t \leq \kappa$, we generate for each agent a set of sequences of length t by doing a *backup* of a previously generated set of sequences of length t - 1 of the agent. We then compute for

each joint-sequence of length t, an $|S|$-vector containing the values of the joint-sequence when the initial belief state is one of the states in S. We then *prune*, for each agent, sequences of length t that are dominated over belief space formed by the cross-product of S and the set of joint-sequences of length t. By starting out with the set $\mathcal{S}_i^1$ (which is in fact just the set A_i) for each agent i, we can incrementally build the set $\mathcal{S}_i^\infty$. Note that a backup of the set $\mathcal{S}_i^t$ creates $|A_i||\Omega_i||\mathcal{S}_i^t|$ new sequences; i.e., the growth is linear. In contrast, the backing-up of a set of policies represents an exponential growth. The merit of this procedure is that we may be able to compute an optimal joint-policy for a slightly longer horizon. But more importantly, due to the linear growth of sequences in each iteration, it may be possible to solve for the infinite-horizon by iterating until some stability or convergence in the values of joint-sequences in realized.

Pompds: Finally, the approach consisting of the use of the sequence-form and mathematical programming could be applied to Pomdps. We have already shown in this paper how a finite-horizon Pomdp can be solved. In conjunction with the dynamic programming approach analogous to the one described above, it may be possible to compute the infinite-horizon discounted value function of a Pomdp.

Acknowledgements

We are grateful to the anonymous reviewers for providing us with valuable comments on this work and suggestions for improving the paper.

References

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

[2] Blair, J. R.; Mutchler, D.; and van Lent, M. 1996. Perfect recall and pruning in games with imperfect information. *Computational Intelligence* 12:131–154.

[3] Emery-Montemerlo, R.; Gordon, G.; Schneider, J.; and Thrun, S. 2004. Approximate solutions for partially observable stochastic games with common payoffs. *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

[4] Hansen, E.; Bernstein, D.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. *Proceedings of the 19th National Conference on Artificial Intelligence* 709–715.

[5] Koller, D., and Megiddo, N. 1992. The complexity of zero-sum games in extensive form. *Games and Economic Behavior* 4(4):528–552.

[6] Koller, D.; Megiddo, N.; and von Stengel, B. 1994. Fast algorithms for finding randomized strategies in game trees. *In Proceedings of the 26th ACM Symposium on Theory of Computing* 750–759.

[7] Kuhn, H. 1953. Extensive games and the problem of information. *Contributions to the Theory of Games II* 193–216.

[8] Nair, R.; Pynadath, D.; Yokoo, M.; and Tambe, M. 2003. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. *Proceedings of the 18th International Joint Conference on Artificial Intelligence* 705–711.

[9] Seuken, S., and Zilberstein, S. 2007. Memory-bounded dynamic programming for dec-pomdps. *In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*.

[10] Szer, D., and Charpillet, F. 2006. Point-based dynamic programming for dec-pomdps. *In Proceedings of the 21st National Conference on Artificial Intelligence*.

[11] Szer, D.; Charpillet, F.; and Zilberstein, S. 2005. Maa*: A heuristic search algorithm for solving decentralized pomdps. *In Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*.

[12] Wilson, R. 1972. Computing equilibria of two-person games from the extensive form. *Management Science* 18:448–460.