Security in Multiagent Systems by Policy Randomization

Praveen Paruchuri, Milind Tambe, Fernando Ordóñez, Sarit Kraus*
University of Southern California, Los Angeles, CA 90089, {paruchur, tambe, fordon}@usc.edu
* Bar-Ilan University, Ramat-Gan 52900, Israel, sarit@cs.biu.ac.il

ABSTRACT

Security in multiagent systems is commonly defined as the ability of the system to deal with intentional threats from other agents. This paper focuses on domains where such intentional threats are caused by unseen adversaries whose actions or payoffs are unknown. In such domains, action randomization can effectively deteriorate an adversary's capability to predict and exploit an agent/agent team's actions. Unfortunately, little attention has been paid to intentional randomization of agents' policies in single-agent or decentralized (PO)MDPs without significantly sacrificing rewards or breaking down coordination. This paper provides two key contributions to remedy this situation. First, it provides three novel algorithms, one based on a non-linear program and two based on linear programs (LP), to randomize single-agent policies, while attaining a certain level of expected reward. Second, it provides Rolling Down Randomization (*RDR*), a new algorithm that efficiently generates randomized policies for decentralized POMDPs via the single-agent LP method.

Categories and Subject Descriptors

I.2.8 [Artificial Intelligence]: Distributed Artificial Intelligence - Multi Agent Systems

General Terms

Security, Design, Theory

Keywords

Safety and Security in Agent Systems, Teamwork, MDP, POMDP, Decentralized POMDP

1. INTRODUCTION

In many adversarial domains, it is crucial for agent and agent teams based on single-agent or decentralized (PO)MDPs to randomize policies in order to avoid action predictability. Such policy randomization is crucial for security in domains where we cannot explicitly model our adversary's actions and capabilities or its payoffs, but the adversary observes our agents' actions and exploits any action predictability in some unknown fashion. Consider agents that schedule security inspections, maintenance or refueling at seaports or airports. Adversaries may be unobserved terrorists with unknown capabilities and actions, who can learn the schedule from observations. If the schedule is deterministic, then these adversaries may exploit schedule predictability to intrude or attack and cause tremendous unanticipated sabotage. Alternatively, consider a team of UAVs (Unmanned Air Vehicles)[2] monitoring a region undergoing a humanitarian crisis. Adversaries may be humans intent on causing some significant unanticipated harm — e.g. disrupting food convoys, harming refugees or shooting down the UAVs — the adversary's capabilities, actions or payoffs are unknown and difficult to model explicitly. However, the adversaries can observe the UAVs and exploit any predictability in UAV surveillance, e.g. engage in unknown harmful actions by avoiding the UAVs' route.

While we cannot explicitly model the adversary's actions, capabilities or payoffs, in order to ensure security of the agent/agent-team we make two worst case assumptions about the adversary. (We show later that a weaker adversary, i.e. one who fails to satisfy these assumptions, will in general only lead to enhanced security.) The first assumption is that the adversary can estimate the agent's state or belief state. In fully observable domains, the adversary estimates the agent's state to be the current world state which both can observe fully. If the domain is partially observable, we assume that the adversary estimates the agent's belief states, because: (i) the adversary eavesdrops or spies on the agent's sensors such as sonar or radar (e.g., UAV/robot domains); or (ii) the adversary estimates the most likely observations based on its model of the agent's sensors; or (iii) the adversary is co-located and equipped with similar sensors. The second assumption is that the adversary knows the agents' policy, which it may do by learning over repeated observations or obtaining this policy via espionage or other exploitation.

Thus, we assume that the adversary may have enough information to predict the agents' actions with certainty if the agents followed a deterministic policy. Hence, our work maximizes policy randomization to thwart the adversary's prediction of the agent's actions based on the agent's state and minimize adversary's ability to cause harm . Unfortunately, while randomized policies are created as a side effect [1] and turn out to be optimal in some stochastic games [10], little attention has been paid to intentionally maximizing randomization of agents' policies even for single agents. Obviously, simply randomizing an MDP/POMDP policy can degrade an agent's expected rewards, and thus we face a randomization-reward tradeoff problem: how to randomize policies with only a limited loss in expected rewards. Indeed, gaining an explicit understanding of the randomization-reward tradeoff requires new techniques for policy generation rather than the traditional single-objective maximization techniques. However, generating policies that provide appropriate randomization-reward tradeoffs is difficult, a difficulty that is exacerbated in agent teams based on decentralized POMDPs, as randomization may create miscoordination.

This paper provides two key contributions to remedy this situation. First, we provide novel techniques that enable policy randomization in single agents, while attaining a certain expected reward threshold. We measure randomization via an entropy-based metric (although our techniques are not dependent on that metric). In particular, we illustrate that simply maximizing entropy-based metrics introduces a non-linear program that does not guarantee polynomial run-time. Hence, we introduce our CRLP (Convex combination for Randomization) and BRLP (Binary search for Randomization) linear programming (LP) techniques that randomize policies in polynomial time with different tradeoffs as explained later. The second part of the paper provides a new algorithm, RDR (Rolling Down Randomization), for generating randomized policies for decentralized POMDPs, given a threshold on the expected team reward loss. RDR starts with a joint deterministic policy for decentralized POMDPs, then iterates, randomizing policies for agents turn-by-turn, keeping policies of all other agents fixed. A key insight in RDR is that given fixed randomized policies for other agents, we can generate a randomized policy via CRLP or BRLP, i.e., our efficient single-agent methods.

The motivation for use of entropy-based metrics to randomize our agents' policies stems from information theory. It is well known that the expected number of probes (e.g., observations) needed to identify the outcome of a distribution is bounded below by the entropy of that distribution [15]. Thus, by increasing policy entropy, we force the opponent to execute more probes to identify the outcome of our known policy, making it more difficult for the opponent to anticipate our agent's actions and cause harm. In particular, in our (PO)MDP setting, the conflict between the agents and the adversary can be interpreted as a game, in which the agents generate a randomized policy above a given expected reward threshold; the adversary knows the agent's policy and the adversary's action is to guess the exact action of the agent/agent-team by probing. For example, in the UAV setting, given our agent's randomized policy, the adversary generates probes to determine the direction our UAV is headed from a given state. Thus, in the absence of specific knowledge of the adversary, we can be sure to increase the average number of probes the adversary uses by increasing the lower bound given by the entropy of the policy distribution at every state.

2. RANDOMIZATION: SINGLE AGENTS

Before considering agent teams, we focus on randomizing single agent MDP policies, e.g. a single MDP-based UAV agent is monitoring a troubled region, where the UAV gets rewards for surveying various areas of the region, but as discussed above, security requires it to randomize its monitoring strategies to avoid predictability.

An MDP is denoted as a tuple: $\langle S, A, P, R\rangle$. S is a set of world states $\{s_1, \ldots, s_m\}$, A the set of actions $\{a_1, \ldots, a_k\}$, P the set of tuples $p(s, a, j)$ representing the transition function and R the set of tuples $r(s, a)$ denoting the immediate reward. If $x(s, a)$ represents the number of times the MDP visits state s and takes action a and α_j represents the number of times that the MDP starts in each state $j \in S$, then the optimal policy, maximizing expected reward, is derived via the following linear program [4]:

$$\begin{array}{lll} \max & \sum_{s\in S}\sum_{a\in A} r(s,a)x(s,a) & \\ \text{s.t.} & \sum_{a\in A} x(j,a) - \sum_{s\in S}\sum_{a\in A} p(s,a,j)x(s,a) = \alpha_j & \\ & \qquad\qquad\qquad\qquad\qquad \forall j \in S & (1) \\ & x(s,a) \geq 0 \qquad\qquad \forall s\in S, a \in A & \end{array}$$

If x^* is the optimal solution to (1), optimal policy π^* is given by (2) below, where $\pi^*(s, a)$ is the probability of taking action a in state s. It turns out that π^* is deterministic and uniformly optimal regardless of the initial distribution $\{\alpha_j\}_{j\in S}$ [4] i.e., $\pi^*(s, a)$ has a value of either 0 or 1. However, such deterministic policies are undesirable in domains like our UAV example.

$$\pi^*(s,a) = \frac{x^*(s,a)}{\sum_{\hat{a}\in A} x^*(s,\hat{a})} \ . \tag{2}$$

2.1 Randomness of a policy

We borrow from information theory the concept of entropy of a set of probability distributions to quantify the randomness, or information content, in a policy of the MDP. For a discrete probability distribution $p_1, \ldots, p_n$ the only function, up to a multiplicative constant, that captures the randomness is the entropy, given by the formula $H = -\sum_{i=1}^{n} p_i \log p_i$ [15]. We introduce a *weighted entropy* function to quantify the randomness in a policy π of an MDP and express it in terms of the underlying frequency x. Note from the definition of a policy π in (2) that for each state s the policy defines a probability distribution over actions. The weighted entropy is defined by adding the entropy for the distributions at every state weighted by the likelihood the MDP visits that state, namely

$$\begin{aligned} H_W(x) &= -\sum_{s\in S}\frac{\sum_{\hat{a}\in A} x(s,\hat{a})}{\sum_{j\in S}\alpha_j}\sum_{a\in A}\pi(s,a)\log\pi(s,a) \\ &= -\frac{1}{\sum_{j\in S}\alpha_j}\sum_{s\in S}\sum_{a\in A} x(s,a)\log\left(\frac{x(s,a)}{\sum_{\hat{a}\in A} x(s,\hat{a})}\right) . \end{aligned}$$

We note that the randomization approach we propose works for alternative functions of the randomness yielding similar results. For example we can define an *additive entropy* taking a simple sum of the individual state entropies as follows:

$$\begin{aligned} H_A(x) &= -\sum_{s\in S}\sum_{a\in A}\pi(s,a)\log\pi(s,a) \\ &= -\sum_{s\in S}\sum_{a\in A}\frac{x(s,a)}{\sum_{\hat{a}\in A} x(s,\hat{a})}\log\left(\frac{x(s,a)}{\sum_{\hat{a}\in A} x(s,\hat{a})}\right) , \end{aligned}$$

We now present three algorithms to obtain random solutions that maintain an expected reward of at least $E_{\min}$ (a certain fraction of the maximal expected reward E^* obtained solving (1)). These algorithms result in policies that, in our UAV-type domains, enable an agent to get a sufficiently high expected reward, e.g. surveying enough area, using randomized flying routes to avoid predictability.

2.2 Maximal entropy solution

We can obtain policies with maximal entropy but a threshold expected reward by replacing the objective of Problem (1) with the definition of the weighted entropy $H_W(x)$. The reduction in expected reward can be controlled by enforcing that feasible solutions achieve at least a certain expected reward $E_{\min}$. The following problem maximizes the weighted entropy while maintaining the expected reward above $E_{\min}$:

$$\begin{array}{lll} \max & -\frac{1}{\sum_{j\in S}\alpha_j}\sum_{s\in S}\sum_{a\in A} x(s,a)\log\left(\frac{x(s,a)}{\sum_{\hat{a}\in A} x(s,\hat{a})}\right) & \\ \text{s.t.} & \sum_{a\in A} x(j,a) - \sum_{s\in S}\sum_{a\in A} p(s,a,j)x(s,a) = \alpha_j & \\ & \qquad\qquad\qquad\qquad\qquad \forall j \in S & (3) \\ & \sum_{s\in S}\sum_{a\in A} r(s,a)x(s,a) \geq E_{\min} & \\ & x(s,a) \geq 0 \qquad\qquad \forall s\in S, a \in A & \end{array}$$

$E_{\min}$ is an input domain parameter (e.g. UAV mission specification). Alternatively, if E^* denotes the maximum expected reward from (1), then by varying the expected reward threshold $E_{\min} \in [0,E^*]$ we can explore the tradeoff between the achievable expected reward and entropy, and then select the appropriate $E_{\min}$. Note that for $E_{\min} = 0$ the above problem finds the maximum weighted entropy policy, and for $E_{\min} = E^*$, Problem (3) returns the maximum expected reward policy with largest entropy. Solving Problem (3) is our first algorithm to obtain a randomized policy that achieves at least $E_{\min}$ expected reward (Algorithm 1).

Algorithm 1 MAX-ENTROPY($E_{\min}$)

```
1: Solve Problem (3) with $E_{\min}$, let $x_{E_{\min}}$ be optimal solution
2: return $x_{E_{\min}}$ (maximal entropy, expected reward $\geq E_{\min}$)
```

Unfortunately entropy-based functions like $H_W(x)$ are neither convex nor concave in x, hence there are no complexity guarantees in solving Problem (3), even for local optima [16]. This negative complexity motivates the polynomial methods presented next.

2.3 Efficient single agent randomization

The idea behind these polynomial algorithms is to efficiently solve problems that obtain policies with a high expected reward while maintaining some level of randomness. (A very high level of randomness implies a uniform probability distribution over the set of actions out of a state, whereas a low level would mean deterministic action being taken from a state). We then obtain a solution that meets a given minimal expected reward value by adjusting the level of randomness in the policy. The algorithms that we introduce in this section consider two inputs: a minimal expected reward value $E_{\min}$ and a randomized solution $\bar{x}$ (or policy $\bar{\pi}$). The input $\bar{x}$ can be any solution with high entropy and is used to enforce some level of randomness on the high expected reward output, through linear constraints. For example, one such high entropy input for MDP-based problems is the uniform policy, where $\bar{\pi}(s,a) = 1/|A|$. We enforce the amount of randomness in the high expected reward solution that is output through a parameter $\beta \in [0,1]$. For a given β and a high entropy solution $\bar{x}$, we output a maximum expected reward solution with a certain level of randomness by solving (4).

$$\begin{aligned}
\max \quad & \sum_{s\in S}\sum_{a\in A} r(s,a)x(s,a) \\
\text{s.t.} \quad & \sum_{a\in A} x(j,a) - \sum_{s\in S}\sum_{a\in A} p(s,a,j)x(s,a) = \alpha_j \qquad \forall j \in S \\
& x(s,a) \geq \beta\bar{x}(s,a) \qquad \forall s\in S, a \in A\ .
\end{aligned} \tag{4}$$

which can be referred to in matrix shorthand as

$$\begin{aligned}
\max \quad & r^T x \\
\text{s.t.} \quad & Ax = \alpha \\
& x \geq \beta\bar{x}\ .
\end{aligned}$$

As the parameter β is increased, the randomness requirements of the solution become stricter and hence the solution to (4) would have smaller expected reward and higher entropy. For $\beta = 0$ the above problem reduces to (1) returning the maximum expected reward solution E^*; and for $\beta = 1$ the problem obtains the maximal expected reward (denoted $\overline{E}$) out of all solutions with as much randomness as $\bar{x}$. If E^* is finite, then Problem (4) returns $\bar{x}$ for $\beta = 1$ and $\overline{E} = \sum_{s\in S}\sum_{a\in A} r(s,a)\bar{x}(s,a)$.

Our second algorithm to obtain an efficient solution with a expected reward requirement of $E_{\min}$ is based on the following result which shows that the solution to (4) is a convex combination of the deterministic and highly random input solutions.

THEOREM 1. *Consider a solution $\bar{x}$, which satisfies $A\bar{x} = \alpha$ and $\bar{x} \geq 0$. Let x^* be the solution to (1) and $\beta \in [0,1]$. If x_β is the solution to (4) then $x_\beta = (1-\beta)x^* + \beta\bar{x}$.*

proof: We reformulate problem (4) in terms of the slack $z = x - \beta\bar{x}$ of the solution x over $\beta\bar{x}$ leading to the following problem :

$$\begin{aligned}
\beta r^T\bar{x} + \quad \max \quad & r^T z \\
\text{s.t.} \quad & Az = (1-\beta)\alpha \\
& z \geq 0\ ,
\end{aligned}$$

The above problem is equivalent to (4), where we used the fact that $A\bar{x} = \alpha$. Let z^* be the solution to this problem, which shows that $x_\beta = z^* + \beta\bar{x}$. Dividing the linear equation $Az = (1-\beta)\alpha$, by $(1-\beta)$ and substituting $u = z/(1-\beta)$ we recover the deterministic problem (1) in terms of u, with u^* as the optimal deterministic solution. Renaming variable u to x, we obtain $\frac{1}{1-\beta}z^* = x^*$, which concludes the proof. □

Since $x_\beta = (1-\beta)x^* + \beta\bar{x}$, we can directly find a randomized solution which obtains a target expected reward of $E_{\min}$. Due to the linearity in relationship between x_β and β, a linear relationship exists between the expected reward obtained by x_β (i.e $r^T x_\beta$) and β. In fact setting $\beta = \frac{r^T x^* - E_{\min}}{r^T x^* - r^T\bar{x}}$ makes $r^T x_\beta = E_{\min}$. We now present below algorithm CRLP based on the observations made about β and x_β.

Algorithm 2 CRLP($E_{\min}, \bar{x}$)

```
1: Solve Problem (1), let $x^*$ be the optimal solution
2: Set $\beta = \frac{r^T x^* - E_{\min}}{r^T x^* - r^T\bar{x}}$
3: Set $x_\beta = (1-\beta)x^* + \beta\bar{x}$
4: return $x_\beta$ (expected reward $= E_{\min}$, entropy based on $\beta\bar{x}$)
```

Algorithm CRLP is based on a linear program and thus obtains, in polynomial time, solutions to problem(4) with expected reward values $E_{\min} \in [\overline{E}, E^*]$. Note that Algorithm CRLP might unnecessarily constrain the solution set as Problem(4) implies that at least $\beta\sum_{a\in A}\bar{x}(s,a)$ flow has to reach each state s. This restriction may negatively impact the entropy it attains, as experimentally verified in Section 5. This concern is addressed by a reformulation of Problem (4) replacing the flow constraints by policy constraints at each stage. For a given $\beta \in [0,1]$ and a solution $\bar{\pi}$ (policy calculated from $\bar{x}$), this replacement leads to the following linear program

$$\begin{aligned}
\max \quad & \sum_{s\in S}\sum_{a\in A} r(s,a)x(s,a) \\
\text{s.t.} \quad & \sum_{a\in A} x(j,a) - \sum_{s\in S}\sum_{a\in A} p(s,a,j)x(s,a) = \alpha_j, \quad \forall j \in S \\
& x(s,a) \geq \beta\bar{\pi}(s,a)\sum_{b\in A} x(s,b), \qquad \forall s\in S, a\in A\ .
\end{aligned} \tag{5}$$

For $\beta = 0$ this problem reduces to (1) returning E^*, for $\beta = 1$ it returns a maximal expected reward solution with the same policy as $\bar{\pi}$. This means that for β at values 0 and 1, problems (4) and (5) obtain the same solution if policy $\bar{\pi}$ is the policy obtained from the flow function $\bar{x}$. However, in the intermediate range of 0 to 1 for β, the policy obtained by problems (4) and (5) are different even if $\bar{\pi}$ is obtained from $\bar{x}$. Thus, theorem 1 holds for problem (4) but not for (5). Table 1, obtained experimentally, validates our claim by showing the maximum expected rewards and entropies obtained (entropies in parentheses) from problems (4) and (5) for various settings of β, e.g. for β = 0.4, problem (4) provides a maximum expected reward of 26.29 and entropy of 5.44, while problem (5) provides a maximum expected reward of 25.57 and entropy of 6.82.

Table 1 shows that for the same value of β in Problems (4) and

(5) we get different maximum expected rewards and entropies implying that the optimal policies for both problems are different, hence Theorem 1 does not hold for (5). Indeed, while the expected reward of Problem (4) is higher for this example, its entropy is lower than Problem (5). Hence to investigate another randomization-reward tradeoff point, we introduce our third algorithm BRLP, which uses problem (5) to perform a binary search to attain a policy with expected reward $E_{\min} \in [\overline{E}, E^*]$, adjusting the parameter β.

Beta	.2	.4	.6	.8
$Problem(4)$	29.14 (3.10)	26.29 (5.44)	23.43 (7.48)	20.25 (9.87)
$Problem(5)$	28.57 (4.24)	25.57 (6.82)	22.84 (8.69)	20.57 (9.88)

Table 1: Maximum expected rewards(entropies) for various β

Algorithm 3 BRLP$(E_{\min}, \bar{x})$

```
1: Set $\beta_l = 0$, $\beta_u = 1$, and $\beta = 1/2$.
2: Obtain $\bar{\pi}$ from $\bar{x}$
3: Solve Problem (5), let $x_\beta$ and $E(\beta)$ be the optimal solution and expected reward value returned
4: while $|E(\beta) - E_{\min}| > \epsilon$ do
5:    if $E(\beta) > E_{\min}$ then
6:       Set $\beta_l = \beta$
7:    else
8:       Set $\beta_u = \beta$
9:    $\beta = \frac{\beta_u + \beta_l}{2}$
10:   Solve Problem (5), let $x_\beta$ and $E(\beta)$ be the optimal solution and expected reward value returned
11: return $x_\beta$ (expected reward $= E_{\min} \pm \epsilon$, entropy related to $\beta\bar{x}$)
```

Given input $\bar{x}$, algorithm BRLP runs in polynomial time, since at each iteration it solves an LP and for tolerance of ϵ, it takes at most $O\left(\frac{E(0)-E(1)}{\epsilon}\right)$ iterations to converge (E(0) and E(1) expected rewards correspond to 0 and 1 values of β).

2.4 Incorporating models of the adversary

Throughout this paper, we set $\bar{x}$ based on uniform randomization $\bar{\pi} = 1/|A|$. By manipulating $\bar{x}$, we can accommodate the knowledge of the behavior of the adversary. For instance, if the agent knows that a specific state s cannot be targeted by the adversary, then $\bar{x}$ for that state can have all values 0, implying that no entropy constraint is necessary. In such cases, $\bar{x}$ will not be a complete solution for the MDP but rather concentrate on the sets of states and actions that are under risk of attack. For $\bar{x}$ that do not solve the MDP Theorem 1 does not hold and therefore Algorithm CRLP is not valid. In this case, a high-entropy solution that meets a target expected reward can still be obtained via Algorithm BRLP.

Before turning to agent teams next, we quickly discuss applying these algorithms in single agent POMDPs. For single-agent finite-horizon POMDPs with known starting belief states[12], we convert the POMDP to (finite horizon) *belief MDP*, allowing BRLP/CRLP to be applied; returning a randomized policy. However, addressing unknown starting belief states is an issue for future work.

3. FROM SINGLE AGENT TO AGENT TEAMS

The logical first step in moving to agent teams would be to start with multiagent MDPs, where a centralized planner provides policies to multiple agents in a fully observable setting. However, such multiagent MDPs are equivalent to single agent MDPs [14]. Indeed, assuming a shared, observable random device — justified, given a fully observable setting — our BRLP/CRLP algorithms can already generate randomized policies for such multiagent MDPs. Hence, this section focuses on the more general decentralized POMDPs, that are not equivalent to single agent POMDPs [3].

3.1 MTDP: A Decentralized POMDP model

We use notation from MTDP (Multiagent Team Decision Problem) [14] for our decentralized POMDP model; other models are equivalent [3]. Given a team of n agents, an MTDP is defined as a tuple: $\langle S, A, P, \Omega, O, R\rangle$. S is a finite set of world states $\{s_1, \ldots, s_m\}$. $A = \times_{1\leq i\leq n} A_i$, where $A_1, \ldots, A_n$, are the sets of action for agents 1 to n. A joint action is represented as $\langle a_1, \ldots, a_n\rangle$. $P(s_i, \langle a_1, \ldots, a_n\rangle, s_f)$, the transition function, represents the probability that the current state is s_f, if the previous state is s_i and the previous joint action is $\langle a_1, \ldots, a_n\rangle$. $\Omega = \times_{1\leq i\leq n}\Omega_i$ is the set of joint observations where Ω_i is the set of observations for agents i. $O(s, \langle a_1, \ldots, a_n\rangle, \omega)$, the observation function, represents the probability of joint observation $\omega \in \Omega$, if the current state is s and the previous joint action is $\langle a_1, \ldots, a_n\rangle$. We assume that observations of each agent are independent of each other's observations, i.e. $O(s, \langle a_1, \ldots, a_n\rangle, \omega) = O_1(s, \langle a_1, \ldots, a_n\rangle, \omega_1) \cdot \ldots \cdot O_n(s, \langle a_1, \ldots, a_n\rangle, \omega_n)$. The agents receive a single, immediate joint reward $R(s, \langle a_1, \ldots, a_n\rangle)$. For deterministic policies, each agent i chooses its actions based on its policy, Π_i, which maps its observation history to actions. Thus, at time t, agent i will perform action $\Pi_i(\vec{\omega}_i^t)$ where $\vec{\omega}_i^t = \omega_i^1, \ldots\ldots, \omega_i^t$. $\Pi = \langle \Pi_1, \ldots\ldots, \Pi_n\rangle$ refers to the joint policy of the team of agents. In this model, execution is distributed but planning is centralized; and agents don't know each other's observations and actions at run time.

Unlike previous work, in our work, policies are randomized and hence agents obtain a probability distribution over a set of actions rather than a single action. Furthermore, this probability distribution is indexed by a sequence of action-observation tuples rather than just observations, since observations do not map to unique actions. Thus in MTDP, a randomized policy maps $\mathbf{\Psi_i^t}$ to a probability distribution over actions, where $\mathbf{\Psi_i^t} = \langle \psi_i^1, \ldots, \psi_i^t\rangle$ and $\psi_i^t = \langle a_i^{t-1}, \omega_i^t\rangle$. Thus, at time t, agent i will perform an action selected randomly based on the probability distribution returned by $\Pi_i(\mathbf{\Psi_i^t})$. Furthermore we denote the probability of an individual action under policy Π_i given $\mathbf{\Psi_i^t}$ as $P^{\Pi_i}(a_i^t|\mathbf{\Psi_i^t})$.

3.2 Illustrative UAV team Domain

To demonstrate key concepts in our algorithms, we introduce a simple UAV team domain that is analogous to the illustrative multi-agent tiger domain [11] except for an adversary – indeed, to enable replicable experiments, rewards, transition and observation probabilities from [11] are used. Consider a region in a humanitarian crisis, where two UAVs execute daily patrols to monitor safe food convoys. However, these food convoys may be disrupted by landmines placed in their route. The convoys pass over two regions: Left and Right. For simplicity, we assume that only one such landmine may be placed at any point in time, and it may be placed in any of the two regions with equal probability. The UAVs must destroy the landmine to get a high positive reward whereas trying to destroy a region without a landmine disrupts transportation and creates a high negative reward; but the UAV team is unaware of which region has the landmine. The UAVs can perform three actions *Shoot-left*, *Sense* and *Shoot-right* but they cannot communicate with each other. We assume that both UAVs are observed with equal probability by some unknown adversary with unknown capabilities, who wishes to cause sabotage. Our worst case assumption about the adversary is as stated in Section 1: (a) the adversary has access to UAV policies due to learning or espionage (b) the adversary eavesdrop or estimates the UAV observations. Thus, if the policy is not randomized, the adversary may exploit UAV action predictability in some unknown way such as jamming UAV sensors, shooting down UAVs or attacking the food convoys, etc. Since little is known about the adversary's ability to cause sabotage, the UAV team must maximize the adversary's uncertainty via policy randomization, while ensuring an above-threshold reward.

When an individual UAV takes action *Sense*, it leaves the state unchanged, but provides a noisy observation OR or OL, to indicate whether the landmine is to the left or right. The *Shoot-left* and *Shoot-right* actions are used to destroy the landmine, but the landmine is destroyed only if both UAVs simultaneously take either *Shoot-left* or *Shoot-right* actions. Unfortunately, if agents miscoordinate and one takes *Shoot-left* and the other *Shoot-right* they incur high negative reward as the landmine is not destroyed but the food-convoy route is damaged. Once the shoot action occurs, the problem is restarted (the UAVs face a landmine the next day).

4. RANDOMIZATION: AGENT TEAMS

Let p_i be the probability of adversary targeting agent i, and $H_W(i)$ be the weighted entropy for agent i's policy. We design an algorithm that maximizes the *multiagent weighted entropy*, given by $\sum_{i=1}^{n} p_i * H_W(i)$, in MTDPs while maintaining the team's expected joint reward above a threshold. Unfortunately, generating optimal policies for decentralized POMDPs is of higher complexity (NEXP-complete) than single agent MDPs and POMDPs [3], i.e., MTDP presents a fundamentally different class where we cannot directly use the single agent randomization techniques.

Hence, to exploit efficiency of algorithms like BRLP or CRLP, we convert the MTDP into a single agent POMDP, but with a method that changes the state space considered. To this end, our new iterative algorithm called RDR (Rolling Down Randomization) iterates through finding the best randomized policy for one agent while fixing the policies for all other agents — we show that such iteration of fixing the randomized policies of all but one agent in the MTDP leads to a single agent problem being solved at each step. Thus, each iteration can be solved via BRLP or CRLP. For a two agent case, we fix the policy of agent i and generate best randomized policy for agent j and then iterate with agent j's policy fixed.

Overall RDR starts with an initial joint deterministic policy calculated in the algorithm as a starting point. Assuming this fixed initial policy as providing peak expected reward, the algorithm then rolls down the reward, randomizing policies turn-by-turn for each agent. Rolling down from such an initial policy allows control of the amount of expected reward loss from the given peak, in service of gaining entropy. *The key contribution of the algorithm is in the rolling down procedure that gains entropy (randomization)*, and this procedure is independent of how the initial policy for peak reward is determined. The initial policy may be computed via algorithms such as [6] that determine a global optimal joint policy (but at a high cost) or from random restarts of algorithms that compute a locally optimal policy [11, 5], that may provide high quality policies at lower cost. The amount of expected reward to be rolled down is input to RDR. RDR then achieves the rolldown in $1/d$ steps where d is an input parameter.

The turn-by-turn nature of RDR suggests some similarities to JESP [11], which also works by fixing the policy of one agent and computing the best-response policy of the second and iterating. However, there are significant differences between RDR and JESP, as outlined below: (i) JESP uses conventional value iteration based techniques whereas RDR creates randomized policies via LP formulations. (ii) RDR defines a new extended state and hence the belief-update, transition and reward functions undergo a major transformation. (iii) The d parameter is newly introduced in RDR to control number of rolldown steps. (iv) RDR climbs down from a given optimal solution rather than JESP's hill-climbing up solution.

RDR Details: For expository purposes, we use a two agent domain, but we can easily generalize to n agents. We fix the policy of one agent (say agent 2), which enables us to create a single agent POMDP if agent 1 uses an extended state, i.e. at each time t, agent 1 uses an extended state $e_1^t = \langle s^t, \mathbf{\Psi_2^t} \rangle$. Here, $\mathbf{\Psi_2^t}$ is as introduced in the previous section. By using e_1^t as agent 1's state at time t, given the fixed policy of agent 2, we can define a single-agent POMDP for agent 1 with transition and observation function as follows.

$$\begin{aligned} P'(e_1^t, a_1^t, e_1^{t+1}) &= P(\langle s^{t+1}, \mathbf{\Psi_2^{t+1}} \rangle | \langle s^t, \mathbf{\Psi_2^t} \rangle, a_1^t) \\ &= P(\omega_2^{t+1} | s^{t+1}, a_2^t, \mathbf{\Psi_2^t}, a_1^t) \\ &\quad \cdot P(s^{t+1} | s^t, a_2^t, \mathbf{\Psi_2^t}, a_1^t) \cdot P(a_2^t | s^t, \mathbf{\Psi_2^t}, a_1^t) \\ &= P(s^t, \langle a_2^t, a_1^t \rangle, s^{t+1}) \cdot O_2(s^{t+1}, \langle a_2^t, a_1^t \rangle, \omega_2^{t+1}) \\ &\quad \cdot P^{\Pi_2}(a_2^t | \mathbf{\Psi_2^t}) \end{aligned} \tag{6}$$

$$\begin{aligned} O'(e_1^{t+1}, a_1^t, \omega_1^{t+1}) &= \Pr(\omega_1^{t+1} | e_1^{t+1}, a_1^t) \\ &= O_1(s^{t+1}, \langle a_2^t, a_1^t \rangle, \omega_1^{t+1}) \end{aligned} \tag{7}$$

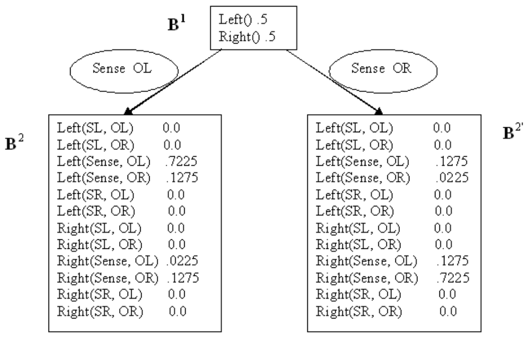


Figure 1: RDR applied to UAV team domain

Thus, we can create a belief state for agent i in the context of j's fixed policy by maintaining a distribution over $e_i^t = \langle s^t, \mathbf{\Psi_j^t} \rangle$. Figure 1 shows three belief states for agent 1 in the UAV domain. For instance B^2 shows probability distributions over e_1^2. In $e_1^2 =$ (Left$\langle Sense, OL \rangle$), Left implies landmine to the left is the current state, Sense is the agent 2's action at time 1, OL (Observe Left) is agent 2's observation at time 2. The belief update rule derived from the transition and observation functions is given in (8), where denominator is the transition probability when action a_1 from belief state B_1^t results in ω_1^{t+1} being observed. Immediate rewards for the belief states are assigned using (9).

$$\begin{aligned} B_1^{t+1}(e_1^{t+1}) = \sum_{s^t} & B_1^t(e_1^t) \cdot P(s^t, (a_1^t, a_2^t), s^{t+1}) \cdot P^{\Pi_2}(a_2^t | \mathbf{\Psi_2^t}) \\ & \cdot O_2(s^{t+1}, (a_1^t, a_2^t), \omega_2^{t+1}) \cdot O_1(s^{t+1}, (a_1^t, a_2^t), \omega_1^{t+1}) \\ & / P(\omega_1^{t+1} | B_1^t, a_1) \end{aligned} \tag{8}$$

$$\Re(a_1^t, B_1^t) = \sum_{e_1^t} B_1^t(e_1^t) \cdot \sum_{a_2^t} R(s^t, (a_1^t, a_2^t)) \cdot P^{\Pi_2}(a_2^t | \mathbf{\Psi_2^t}) \tag{9}$$

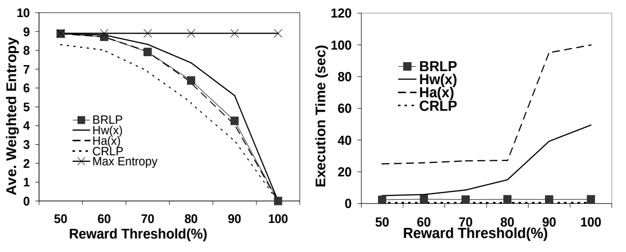
Thus, RDR's policy generation implicitly coordinates the two agents, without communication or a correlational device. Randomized actions of one agent are planned taking into account the impact of randomized actions of its teammate on the joint reward. Algorithm 4 (at the end of paper) presents the pseudo-code for RDR for two agents, and shows how we can use beliefs over extended states e_i^t to construct LPs that maximize entropy while maintaining a certain expected reward threshold. The inputs to this algorithm are the parameters d, *percentdec* and $\bar{x}$. The parameter d specifies the number of iterations taken to solve the problem. It also decides the amount of reward that can be sacrificed at each step of the algorithm for improving entropy. Parameter *percentdec* specifies the percentage of expected reward the agent team forgoes for improving entropy. As with Algorithm 2, parameter $\bar{x}$ provides an input solution with high randomness; and it is obtained using a uniform policy as discussed in Section 2.3. Step 1 of the algorithm uses the Compute joint policy() function which returns a optimal joint deterministic policy from which the individual policies and the expected joint reward are extracted. Obtaining this initial policy is not RDR's main thrust, and could be a global optimal policy obtained via [6] or a very high quality policy from random restarts of local algorithms such as [11, 5]. The input variable *percentdec* varies between 0 to 1 denoting the percentage of the expected reward that can be sacrificed by RDR. The step size as calculated in the algorithm, denotes the amount of reward sacrificed during each iteration. RDR then iterates 1/d times(step 3).

The function GenerateMDP generates all reachable belief states (lines 2 through 6) from a given starting belief state *B* and hence a belief MDP is generated. The number of such belief states is $O(|A_1||\Omega_1|)^{T-1}$ where T is the time horizon. The extended states in each B increases by a factor of $|A_2||\Omega_2|$ with increasing T. Thus the time to calculate B(e) for all extended states e, for all belief states B in agent 1's belief MDP is $O(|S|^2(|A_1||A_2||\Omega_1||\Omega_2|)^{T-1})$. Lines 7 through 12 compute the reward for each belief state. The total computations to calculate the reward is $O(|S||A_1||A_2|(|A_1||A_2||\Omega_1||\Omega_2|)^{T-1})$. The belief MDP generated is denoted by the tuple $\langle B, A, trans, R\rangle$. We reformulate the MDP obtained to problem 4 and use our polynomial BRLP procedure to solve it, using $\bar{x}$ as input. Thus, algorithm RDR is exponentially faster than an exhaustive search of a policy space, and comparable to algorithms that generate locally optimal policies [11].

5. EXPERIMENTAL RESULTS

We present three sets of experimental results. Our first set of experiments examine the tradeoffs in run-time, expected reward and entropy for single-agent problems. Figures 2a and 2b show the results for these experiments based on generation of MDP policies. The results show averages over 10 MDPs where each MDP represents a flight of a UAV, with state space of 28-40 states. The states represent the regions monitored by the UAVs. The transition function assumes that a UAV action can make a transition from a region to one of four other regions, where the transition probabilities were selected at random. The rewards for each MDP were also generated using random number generators. These experiments compare the performance of our four methods of randomization for single-agent policies. In the figures, *CRLP* refers to algorithm 2; *BRLP* refers to algorithm 3; whereas $H_W(x)$ and $H_A(x)$ refer to Algorithm 1 with these objective functions. Figure 2a examines the tradeoff between entropy and expected reward thresholds. It shows the average weighted entropy on the y-axis and reward threshold percent on the x-axis. The average maximally obtainable entropy for these MDPs is 8.89 (shown by line on the top) and three of our four methods (except CRLP) attain it at about 50% threshold, i.e. an agent can attain maximum entropy if it is satisfied with 50% of the maximum expected reward. However, if no reward can be sacrificed (100% threshold) the policy returned is deterministic.

Figure 2b shows the run-times, plotting the execution time in seconds on the y-axis, and expected reward threshold percent on the x-axis. These numbers represent averages over the same 10 MDPs as in Figure 2a. Algorithm CRLP is the fastest and its runtime is very small and remains constant over the whole range of threshold rewards as seen from the plot. Algorithm BRLP also has a fairly constant runtime and is slightly slower than CRLP. Both CRLP and BRLP are based on linear programs and hence their small and fairly constant runtimes. Algorithm 1, for both $H_A(x)$ and $H_W(x)$ objectives, exhibits an increase in the runtime as the expected reward threshold increases. This trend that can be attributed to the fact that maximizing a non-concave objective while simultaneously attaining feasibility becomes more difficult as the feasible region shrinks.



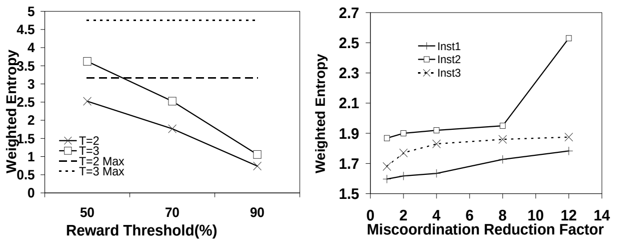
(a) (b)

Figure 2: Comparison of Single Agent Algorithms

We conclude the following from Figure 2: (i) CRLP is the fastest but provides the lowest entropy. (ii) BRLP is significantly faster than Algorithm 1, providing 7-fold speedup on average over the 10 MDPs over the entire range of thresholds. (iii) Algorithm 1 with $H_W(x)$ provides highest entropy among our methods, but the average gain in entropy is only 10% over BRLP. (iv) CRLP provides a 4-fold speedup on an average over BRLP but with a significant entropy loss of about 18%. In fact, CRLP is unable to reach the maximal possible entropy for the threshold range considered in the plot. Thus, BRLP appears to provide the most favorable tradeoff of run-time to entropy for the domain considered, and we use this method for the multiagent case. However, for time critical domains CRLP might be the algorithm of choice and therefore both BRLP and CRLP provide useful tradeoff points.

Reward Threshold	1	.5	.25	.125
90%	.67(.59)	1.73(.74)	3.47(.75)	7.07(.75)
50%	.67(1.53)	1.47(2.52)	3.4(2.62)	7.47(2.66)

Table 2: RDR: Avg. run-time in sec and (Entropy), $T = 2$



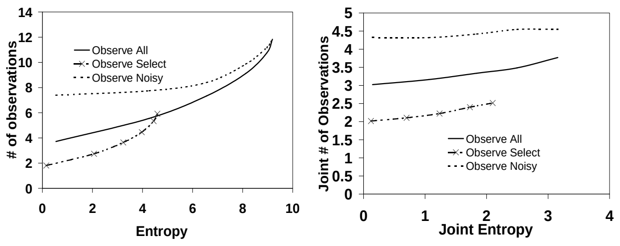
(a) (b)

Figure 3: Results for RDR

Our second set of experiments examine the tradeoffs in run-time, expected joint reward and entropy for the multiagent case. Table 2 shows the runtime results and entropy (in parenthesis) averaged over 10 instances of the UAV team problem based on the original transition and observation functions from [11] and its variations. d, the input parameter controlling the number of rolldown steps of algorithm 4, varies from 1 to 0.125 for two values of percent reward threshold (90% and 50%) and time horizon T=2. We conclude that as d decreases, the run-time increases, but the entropy remains fairly constant for $d \leq .5$. For example, for reward threshold of 50%, for d = 0.5, the runtime is 1.47 secs, but the run-time increases more than 5-fold to 7.47 when d = 0.125; however, entropy only changes from 2.52 to 2.66 with this change in d.

Thus in our next set of graphs, we present results for d = .5, as it provides the most favorable tradeoff, if other parameters remain fixed. Figure 3a plots RDR expected reward threshold percent on the x-axis and weighted entropy on the y-axis averaged over the same 10 UAV-team instances. Thus, if the team needs to obtain 90% of maximum expected joint reward with a time horizon $T = 3$, it gets a weighted entropy of 1.06 only as opposed to 3.62 if it obtains 50% of the expected reward for the same d and T. Similar to the single-agent case, the maximum possible entropy for the multiagent case is also shown by a horizontal line at the top of the graph. Figure 3b studies the effect of changing miscoordination cost on RDR's ability to improve entropy. As explained in Section 3.2, the UAV team incurs a high cost of miscoordination, e.g. if one UAV shoots left and the other shoots right. We now define miscoordination reduction factor (MRF) as the ratio between the original miscoordination cost and a new miscoordination cost. Thus, high MRF implies a new low miscoordination cost, e.g. an MRF of 4 means that the miscoordination cost is cut 4-fold. We plot this MRF on x-axis and entropy on y-axis, with expected joint reward threshold fixed at 70% and the time horizon T at 2. We created 5 reward variations for each of our 10 UAV team instances we used for 3a; only 3 instances are shown, to reduce graph clutter(others are similar). For *instance 2*, the original miscoordination cost provided an entropy of 1.87, but as this cost is scaled down by a factor of 12, the entropy increases to 2.53.

Based on these experiments, we conclude that: (i) Greater tolerance of expected reward loss allows higher entropy; but reaching the maximum entropy is more difficult in multiagent teams — for the reward loss of 50%, in the single agent case, we are able to reach maximum entropy, but we are unable to reach maximum entropy in the multiagent case. (ii) Lower miscoordination costs allow higher entropy for the same expected joint reward thresholds. (iii) Varying d produces only a slight change in entropy; thus we can use d as high as 0.5 to cut down runtimes. (iv) RDR is time efficient because of the underlying polynomial time BRLP algorithm.



(a) (b)

Figure 4: Improved security via randomization

Our third set of experiments examine the tradeoffs in entropy of the agent/agent-team and the total number of observations (probes) the enemy needs to determine the agent/agent-team actions at each state. The primary aim of this experiment is to show that maximizing policy entropy indeed makes it more difficult for the adversary to determine/predict our agents' actions, and thus more difficult for the adversary to cause harm, which was our main goal at the beginning of this paper. Figures 4a and 4b plot the number of observations of enemy as function of entropy of the agent/agent-team. In particular for the experiment we performed, the adversary runs yes-no probes to determine the agent's action at each state, i.e. probes that return an answer yes if the agent is taking the particular action at that state in which case the probing is stopped, and a no otherwise. The average number of yes-no probes at a state is the total number of observations needed by the adversary to determine the correct action taken by the agent in that state. The more deterministic the policy is, the fewer the probes the adversary needs to run; if the policy is completely deterministic, the adversary need not run any probes as it knows the action. Therefore, the aim of the agent/agent-team is to maximize the policy entropy, so that the expected number of probes asked by the adversary is maximized.

In contrast, the adversary minimizes the expected number of probes required to determine the agents' actions. Hence, for any given state s, the adversary uses the Huffman procedure to optimize the number of probes [8], and hence the total number of probes over the entire MDP state space can be expressed as follows. Let $S = \{s_1, s_2, ..., s_m\}$ be the set of MDP states and $A = \{a_1, a_2, ..., a_n\}$ be the action set at each state. Let $p(s, a) = \{p_1, ..., p_n\}$ be the probabilities of taking the action set $\{a'_1, ..., a'_n\}$, $a'_i \in A$ at state s sorted in decreasing order of probability. The number of yes-no probes at state s is denoted by $\zeta_s = p_1 * 1 + ... + p_{n-1} * (n-1) + p_n * (n-1)$. If the weight of the states (see notion of weight introduced in section 2) is $W = \{w_1, w_2, ..., w_m\}$, then the number of observations over the set of states is denoted *Observe-all* $= \sum_{s=1...m}\{w_s * \zeta_s\}$. Setting some weights to zero implies that the adversary was not concerned with those states, and the number of observations in this situation is denoted *Observe-select*. While the number of observations in both Observe-all and Observe-select are obtained assuming the adversary obtains an accurate policy of the agent or agent team, in real situations, an adversary may obtain a noisy policy, and the adversary's number of observations in such a case is denoted Observe-noisy.

Figure 4a demonstrates the effectiveness of entropy maximization using the BRLP method against an adversary using yes-no probes procedure as his probing method for the single agent case. The plot shows the number of observations on y-axis and entropy on the x-axis averaged over the 10 MDPs we used for our single-agent experiment. The plot has 3 lines corresponding to the three adversary procedures namely *Observe-all*, *Observe-select* and *Observe-noisy*. *Observe-all* and *Observe-select* have been plotted to study the effect of entropy on the number of probes the adversary needs. For example, for *Observe-all*, when entropy is 8, the average number of probes needed by the adversary is 9. The purpose of the *Observe-noisy* plot is to show that the number of probes that the adversary requires can only remain same or increase when using a noisy policy, as opposed to using the correct agent policy. The noise in our experiments is that two actions at each state of the MDP have incorrect probabilities. Each data point in the *Observe-noisy* case represents an average of 50 noisy policies, averaging over 5 noisy policies for each reward threshold over each of the 10 MDPs.

Figure 4b plots a similar graph as 4a for the multiagent case, averaged over the 10 UAV-team instances with two UAVs. The plot has three lines namely *Observe-all*, *Observe-select* and *Observe-noisy* with the same definitions as in the single agent case but in a distributed POMDP setting. However, in plot 4b the y-axis represents joint number of yes-no probes and the x-axis represents joint entropy. Both these parameters are calculated as weighted sums of the individual parameters for each UAV, assuming that the adversary assigns equal weight to both the UAVs.

We conclude the following from plots 4a and 4b: (i) The number of observations(yes-no probes) increases as policy entropy increases, whether the adversary monitors the entire state space (*observe-all*) or just a part of it (*observe-select*). (ii) If the adversary obtains a noisy policy (*observe-noisy*), it needs a larger number of observations when compared to the adversary obtaining an accurate policy. (iii) As entropy increases, the agents' policy tends to become more uniform and hence the effect of noise on the number of yes-no probes reduces. In the extreme case where the policy is totally uniform the *Observe-all* and *Observe-noisy* both have same number of probes. This can be observed at the maximal entropy point in plot 4a. The maximal entropy point is not reached in plot 4b as shown in the results for RDR. From the above we conclude that maximizing entropy has indeed made it more difficult for the adversary to determine our agents' actions and cause harm.

6. SUMMARY AND RELATED WORK

This paper focuses on security in multiagent systems where intentional threats are caused by unseen adversaries, whose actions and capabilities are unknown, but the adversaries can exploit any predictability in our agents' policies. Policy randomization for single-agent and decentralized (PO)MDPs, with some guaranteed expected rewards, are critical in such domains. To this end, this paper provides two key contributions: (i) provides novel algorithms, in particular the polynomial-time CRLP and BRLP algorithms, to randomize single-agent MDP and POMDP policies, while attaining a certain level of expected reward; (ii) provides RDR, a new algorithm to generate randomized policies for decentralized POMDPs. RDR can be built on BRLP or CRLP, and thus is able to efficiently provide randomized policies. Finally, while our techniques are applied for analyzing randomization-reward tradeoffs, they could potentially be applied more generally to analyze different tradeoffs between competing objectives in single/decentralized (PO)MDP. More details about this work can be accessed at
http://www-scf.usc.edu/ paruchur/japan.ppt.

Randomization as a goal has received little attention in the literature, and is seen as a means or side-effect in attaining other objectives, e.g., in resource-constrained MDPs [1] or memoryless POMDP policy generators [9] (for breaking loops). In [13] coordination of multiple agents executing randomized policies in a distributed MDP setting is discussed, but there randomization occurs as a side-effect of resource constraints; furthermore, agents communicate to resolve the resulting team miscoordination. In contrast, our work explicitly emphasizes maximizing entropy in policies, and attains implicit coordination (no communication) over randomized policies in decentralized POMDPs. Significant attention has been paid to learning in stochastic games, where agents must learn dominant strategies against explicitly modeled adversaries [10, 7]. Such dominant strategies may lead to randomization, but randomization itself is not the goal. Our work in contrast does not require an explicit model of the adversary and, in this worst case setting, hinders any adversary's actions by increasing the policy's weighted entropy through efficient algorithms, such as BRLP.

Acknowledgements : This research is supported by NSF grant #0208580 & #0222914. It is also supported by the United States Department of Homeland Security through Center for Risk and Economic Analysis of Terrorism Events (CREATE). Sarit Kraus is also affiliated with UMIACS.

7. REFERENCES

[1] E. Altman. *Constrained Markov Decision Process*. Chapman and Hall, 1999.

[2] R. Beard and T. McLain. Multiple uav cooperative search under collision avoidance and limited range communication constraints. In *IEEE CDC*, 2003.

[3] D. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of MDPs. In *UAI*, 2000.

[4] D. Dolgov and E. Durfee. Constructing optimal policies for agents with constrained architectures. Technical report, Univ of Michigan, CSE-TR-476-03, 2003.

[5] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS*, 2004.

[6] E. Hansen, D. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, 2004.

[7] J. Hu and P. Wellman. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, 1998.

[8] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proc. IRE 40*, 1952.

[9] T. Jaakkola, S. Singh, and M. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. *Advances in NIPS*, 7, 1994.

[10] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *ML*, 1994.

[11] R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, 2003.

[12] S. Paquet, L. Tobin, and B. Chaib-draa. An online POMDP algorithm for complex multiagent environments. In *AAMAS*, 2005.

[13] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus. Towards a formalization of teamwork with resource constraints. In *AAMAS*, 2004.

[14] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR*, 16:389–423, 2002.

[15] C. Shannon. A mathematical theory of communication. *The Bell Labs Technical Journal*, pages 379–457,623,656, 1948.

[16] S. Vavasis. Nonlinear optimization: Complexity issues. In *University Press, New York*, 1991.

Algorithm 4 RDR$(d, percentdec, \bar{x})$

```
 1: $\pi_1, \pi_2, Optimalreward \leftarrow Compute\_joint\_policy()$
 2: $stepsize \leftarrow percentdec \cdot Optimalreward \cdot d$
 3: for $i \leftarrow 1$ to $1/d$ do
 4:    $MDP \leftarrow GenerateMDP(b, \Pi_{(i+1)\text{Mod}_2}, T)$
 5:    $Entropy, \Pi_{i\text{Mod}_2} \leftarrow BRLP(Optimalrew - stepsize * i, \bar{x})$

 1: $\mathbf{GenerateMDP}(b, \pi_2, T)$ :
 2: $reachable(0) \leftarrow \{b\}$
 3: for $t \leftarrow 1$ to $T$ do
 4:    for all $B^{t-1} \in reachable(t-1)$ do
 5:       for all $a_1 \in A_1, \omega_1 \in \Omega_1$ do
 6:          $trans, reachable(t) \overset{\cup}{\leftarrow} \text{UPDATE}(B^{t-1}, a_1, \omega_1)$
 7: for $t \leftarrow T$ downto $1$ do
 8:    for all $B^t \in reachable(t)$ do
 9:       for all $a_1 \in A_1$ do
10:          for all $s^t \in S, \psi_2^t \leftarrow \langle a_2^{t-1}, \omega_2^t \rangle$ do
11:             for all $a_2^t$ given $\mathbf{\Psi_2^t}$ do {Equation 9}
12:                $\Re_t^{a_1}(B^t) \overset{+}{\leftarrow} B^t(s^t, \mathbf{\Psi_2^t}) \cdot R\left(s^t, \langle a_1^t, a_2^t \rangle\right) . P(a_2^t | \mathbf{\Psi_2^t})$
13: return $\langle B, A, trans, \Re \rangle$

 1: $\mathbf{UPDATE}(B^t, a_1, \omega_1)$ :
 2: for all $s^{t+1} \in S, \psi_2^t \leftarrow \langle a_2^{t-1}, \omega_2^t \rangle$ do
 3:    for all $s^t \in S$ do {Equation 8}
 4:       $B^{t+1}(s^{t+1}, \mathbf{\Psi_2^{t+1}}) \overset{+}{\leftarrow} B^t(s^t, \mathbf{\Psi_2^t}) \cdot P(s^t, (a_1^t, a_2^t), s^{t+1}) \cdot O_1(s^{t+1}, (a_1^t, a_2^t), \omega_1^{t+1}) \cdot O_2(s^{t+1}, (a_1^t, a_2^t), \omega_2^{t+1}) \cdot P(a_2^t | \mathbf{\Psi_2^t})$
 5:    $trans \leftarrow normalize(B^{t+1}(s^{t+1}, \mathbf{\Psi_2^{t+1}}))$
 6: return $trans, B^{t+1}$
```