Dynamic Programming with Stochastic Opponent Models in Social Games

Tsz-Chiu Au
Department of Computer Science
University of Maryland
College Park, MD 20742, U.S.A.
chiu@cs.umd.edu

Abstract

Policy makers often confront with the following problem: how best their organization can repeatedly interact with other organizations such that the long-term utility of their organization can be maximized? This problem is difficult because policy makers usually know very little about other organizations, and therefore they cannot make perfect predictions about the other organizations' behaviors. In this paper, we formulate this problem as social games in which (1) there are two or more agents interacting with each other; (2) each agent can perform more than one action in each interaction; and (3) the payoff matrix is not fixed; the payoff matrix varies from one situation to another. We devised a dynamic programming algorithm to compute a policy given the model of the other agent's behavior, written in a language called SOMA-programs, a rich language for representing agent's incomplete belief about the other agents' behavior.

Introduction

Policy makers often deal with problems that can be formulated as *social games*, in which agents (such as government agencies, companies, and ethnic groups) interact with other agents for a long period of time, and the success of the agents depends on how well the agents interact with each other. One of the difficulty in these games is that an agent usually know very little about the decision making process of the other agents. But an agent can often build an *incomplete* model of other agents by observing the other agents' past behavior. The challenge, therefore, is how to make decisions based on incomplete models of the other agents' behavior.

This paper proposes a solution to this problem based on an opponent modeling language called *SOMA-programs* (Simari *et al.* 2006). In this language, the other agent's behavior is represented by a set of probabilistic rules called *SOMA-rules*, each of them describes what actions an agent will probably do when certain conditions are satisfied. The key feature of SOMA-rules is that it uses probability intervals to handle missing information about the probability in the rules—even if the modeler does not have the exact probability of the actions that an agent will do when some conditions are satisfied, he can still simply give the upper bound and the lower bound of the probabilities in the SOMA-programs. Unlike other framework such as Markov Decision Process (MDP), SOMA-programs does not require one to give all the transition rules in a transition matrix; SOMA-programs can handle the missing rules nicely using probability intervals. Therefore SOMA-programs is suitable for modeling agents with limited information about their actual behavior.

We focus on the problem that can be modeled as a social game in which the long-term payoff depends on the interaction among the agents. We allow each agent to choose more than one action in each interaction. In each interaction, an agent receives a numeric payoff (such as money), which is determined by the set of actions chose by all agents. The goal is to maximize the lower bound of the expected accumulative payoff of an agent. Our approach is to use dynamic programming to compute a policy that maximize the lower bound of the expected accumulative payoff of an agent, given that there is a SOMA-program for every other agent.

Example

A manufacturer needs to purchase certain amount of raw materials every day. It can choose to purchase from n different suppliers. In order to maintain the daily business of the manufacturer, at least m suppliers must provide the raw materials. But if more than m suppliers provide the raw materials, the extra stock will be wasted.

The suppliers, however, may not be able to provide the requested materials to the manufacturer, due to the limited productivity of their factories that produce the raw materials. Furthermore, it can be beneficial to the suppliers not supplying the requested materials to the manufacturer but to other manufacturer.

Suppose there are three suppliers: Supplier A, Supplier B, and Supplier C. The manufacturer can choose to order raw materials from a subset of them. Simultaneously, the suppliers can choose to produce the raw material for the supplier, even though the suppliers do not know whether the manufacturer would order the raw materials from them. The manufacturer has to make sure that at least one supplier must provide the raw material.

We denote the manufacturer as ψ_1, and the Supplier A, the Supplier B, and the Supplier C as ψ_2, ψ_3, and ψ_4, respectively. We denote the action that the manufacturer orders raw materials from a supplier ψ_k by $\mathsf{order}(\psi_k)$. Likewise, we denote the action that the supplier ψ_k produces raw materials for the manufacturer by $\mathsf{produce}(\psi_k)$.

The daily reward for the manufacturer depends on several factors: (1) the actions it chooses, (2) the actions the suppliers choose, and (3) how many days the manufacturer does not receive sufficient raw materials in the last 7 days. The last factor can be modeled by having a set of states $\{S_1, S_2, \ldots, S_7\}$, such that S_i means in the past 7 days the number of days on which the manufacturer failed to receive sufficient raw materials is i. Let C be the set of actions chosen by the manufacturer and the suppliers. We define a reward function R_1 as follows.

$$R_1(S_i, C) = 10\times i + 3\times|\{\psi_k : \mathsf{order}(\psi_k), \mathsf{produce}(\psi_k) \in C\}|$$

We consider the accumulated reward of a period of 30 days. Suppose the actions of the agents (i.e., the manufacturer and the suppliers) on the i'th day is C_i. The accumulated reward of a period of 30 days would be $\sum_{1\leq i\leq 30} R_1(S_i, C_i)$, where S_i is the state of the world after the the actions before the i'th day are executed. The objective of the manufacturer is to choose a set of order actions on every day so as to maximize its accumulated reward.

The problem, however, is that the manufacturer does not know the exact behavior of the suppliers. Perhaps the behavior is inherently uncertain, since the suppliers' decisions depend on unpredictable factors such as traffics and weathers.

We propose to use a set of rules called SOMA-programs to describe the behavior of the suppliers. The feasibility of SOMA-programs allows us to encode knowledge about the suppliers' behavior that we learnt from the historical records of the interaction between the manufacturer and the suppliers. The objective of the manufacturer is the maximization of the *lower bound* of the expected accumulated reward over a period of 30 days, given the limited information (in form of SOMA-rules) of the behavior the suppliers.

Definition

We assume that states of an agent in a particular time are logically describable using logical formulas whose syntax consists of a finite set of *predicate symbols* (each with an associated arity), a finite set of *constant symbols*, and an infinite set of *variable symbols*. We assume there is no function symbol; thus a *term* can only be either a constant symbol or a variable symbol. An *atom* is $p(t_1, t_2, \ldots t_n)$, where p is an n-ary predicate symbol and $t_1, t_2, \ldots, t_n$ are terms. If $t_1, t_2, \ldots t_n$ are constants, then the atom $p(t_1, t_2, \ldots t_n)$ is a *ground* atom. Let $\mathcal{S}$ be the set of all ground atoms describable in this language. The state of an agent at any given point in time is a set of ground atoms (i.e., a subset of S).

An agent can choose one or more actions and performs them at any point in time. Actions can be expressed by logical formulas that involve a finite set of *action symbols* (each with an associated arity), a finite set of *constant symbols*, and an infinite set of *variable symbols*. The set of constant symbols and variable symbols are the same as those in the logical formulas for states. But the set of action symbols is disjoint from the set of predicate symbols that are used to describe agent's states. An *action atom* is $a(t_1, t_2, \ldots, t_n)$, where a is an action symbols and $t_1, t_2, \ldots, t_n$ are terms as defined above. A *ground* action atom is an action atom whose terms are constants. Let $\mathcal{A}$ be the set of all ground action atoms in this language. A *course of action* (COA) is a finite subset of ground action atoms in $\mathcal{A}$.

Let L be a finite set of literals. Then we denote the set of positive literals in L by L^+, and the set of negative literals by L^-.

Probabilistic Operators

There are several formalizations of probabilistic actions. The most well-known formalizations are (1) probabilistic STRIPS operators (Kushmerick, Hanks, & Weld 1995), (2) factored probabilistic STRIPS operators (Dearden & Boutilier 1997), and (3) Probabilistic PDDL 1.0 for the 2004 International Planning Competition (Younes *et al.* 2005). These formalizations are very expressive; they can be used to describe complicated actions succinctly. Our definition of actions resembles probabilistic STRIPS operators in (Kushmerick, Hanks, & Weld 1995). In this paper, we mathematically define operators as follows.

Definition 1 *An* operator *is a tuple* $(\mathsf{op}, \mathsf{pre}, \mathsf{eff}, \Delta)$, *where*

- op *is an action atom;*
- pre *is a finite set of literals;*
- eff *is a collection of sets of literals; and*
- Δ *is a probability distribution over* eff.

We say op *is the* name *of the operator,* pre *is the* precondition *of the operator,* $e \in \mathsf{eff}$ *is an* effect *of the operator, and* $\Delta(e)$ *is the probability of the effect* $e \in \mathsf{eff}$. *All variables in* pre *and* eff *must appear in* op.

Suppose θ is a substitution of variables in an operator α. We use $\alpha\theta$ to denote the ground operator after the substitution. An action is a ground operator. We denote the precondition and the effect of an action a by $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$, respectively. Each element in $\mathsf{eff}(a)$ is a *possible effect* of a.

A state is a set of atoms. A ground action $a = (\mathsf{op}, \mathsf{pre}, \mathsf{eff}, \Delta)$ is *applicable* in a state S if and only if $\mathsf{pre} \subseteq S$. After the execution of a at S, the next state would be $\mathsf{apply}(S, e) = (S \setminus e^-) \cup e^+$ for some $e \in \mathsf{eff}$ with the probability $\Delta(e)$.

When actions a_1, a_2, $\ldots$, a_n are executed at state S at the same time, then the next state is $\mathsf{apply}(S, \{e_1, e_2, \ldots, e_n\}) = \mathsf{apply}(\mathsf{apply}(\mathsf{apply}(S, e_1), e_2), \ldots, e_n)$, where e_i is a possible effect of a_i.

SOMA-Programs: A Stochastic Language for Modeling Agents

We assume the behavior of an agent can be modeled as a SOMA-program, which consists of a set of rules called SOMA-rules. This section summarizes the definition of SOMA-rules and SOMA-programs in (Simari *et al.* 2006).

An action formula is either an action atom or a formula in one of the following form: $(P \wedge Q)$, $(P \vee Q)$, or $\neg P$, where

P and Q are action formulas. A SOMA-rule is a formula of the form:

$$P : [l, u] \leftarrow B_1 \wedge \ldots \wedge B_n,$$

where $B_1, \ldots, B_n$ are atoms, $0 \leq l \leq u \leq 1$ are probability values, and P is an action formula. If P is an action atom, the above rule is an elementary SOMA-rule. A SOMA-rule is *ground* if and only if P,B_1, B_2, ..., B_n are ground. We denote the set of ground instances of a rule r by $\mathsf{ground}(r)$.

A SOMA-program is a finite set of SOMA-rules. An elementary SOMA-program is a finite set of elementary SOMA-rules.

A SOMA-program syntactically defines our knowledge about the behavior of an agent. The actual behavior of an agent, however, remains unknown. But the SOMA-program can be used to confine the set of possible behaviors that the agent might have. According to this viewpoint, we define the semantics of a SOMA-program as follows.

A course of action (COA) is a finite set of ground action atoms that an agent might take in a given situation. Let $gr(\mathcal{A})$ be the set of all possible ground actions. Then the power set $\mathcal{C} = 2^{gr(\mathcal{A})}$ of $gr(\mathcal{A})$ be the set of all possible COAs. For any course of action $C \in \mathcal{C}$, we denote the probability that the agent performs exactly the actions in C by $I(C)$. More precisely, a *SOMA-interpretation* I is a probability distribution over $\mathcal{C}$—a mapping from $\mathcal{C}$ to $[0, 1]$ such that $\sum_{C \in \mathcal{C}} I(C) = 1$.

A course of action C *satisfies* a ground action atom P if and only if $P \in C$. A course of action C *satisfies* a ground action formula P if and only if either (1) C satisfies P when P is an action atom; (2) C satisfies P_1 and P_2 when P has the form $P_1 \wedge P_2$; (3) C satisfies P_1 or P_2 when P has the form $P_1 \vee P_2$; or (4) C does not satisfy P when P has the form $\neg P$. If C satisfies a ground action formula P, we also say C is *feasible* with respect to P, and write $C \mapsto P$.

A course of action C is *feasible* w.r.t. a state S if and only if all actions in C are applicable in S. We denote this relationship by a boolean function $\phi(C, S)$, such that $\phi(C, S) = \mathsf{True}$ if and only if C is feasible at S. We assume all applicable ground actions in C feasible at S can be executed in parallel at the same time—there is no ground atom B such that there are two actions $\psi_1, \psi_2 \in C$ such that ψ_1 adds B to the state but ψ_2 deletes B from the state. A ground SOMA-rule $r = (P : [l,\ u] \leftarrow B_1 \wedge \ldots \wedge B_n)$ is *applicable* at a state S if and only if $\{B_1, \ldots, B_n\} \subseteq S$.

A SOMA-interpretation I *satisfies* a ground SOMA-rule r w.r.t. a state S applicable to r if and only if either (1) r is applicable at S and the sum of the probabilities of all feasible (w.r.t. both P and S) COAs is between l and u inclusively; or (2) r is not applicable at S. More precisely, I satisfies r w.r.t. S if and only if (1) $l \leq \sum_{\phi(C,S)=\mathsf{True} \wedge (C \mapsto P)} I(C) \leq u$, or (2) $\{B_1, \ldots, B_n\} \not\subseteq S$. A SOMA-interpretation I *satisfies* a SOMA-rule w.r.t. a state S if and only if it satisfies all ground instances of the rule.

A SOMA-program Π is *consistent* w.r.t. a state S if there is at least one SOMA-interpretation that satisfies all rules in Π w.r.t. S. Given a SOMA-program Π and a state S, we define a set of constraints over the probabilities of COAs as follows. Let $P(C)$ be the probability of a COA $C \in \mathcal{C}$. Then the set of constraints $\mathsf{CONS}(\Pi, S)$ are:

1. For all $r \in \Pi$ and for all $r' \in \mathsf{ground}(r)$ such that r' is applicable to S,

$$\left\{ l \leq \sum_{\phi(C,S)=\mathsf{True} \wedge C \mapsto P} P(C) \leq u \right\} \in \mathsf{CONS}(\Pi, S),$$

2.

$$\left\{ \sum_{\phi(C,S)=\mathsf{True}} P(C) = 1 \right\} \in \mathsf{CONS}(\Pi, S).$$

Every solution to the set $\mathsf{CONS}(\Pi, S)$ of constraints is a SOMA-interpretation I that makes Π consistent—I will satisfies all the rules in Π. A theorem in (Simari *et al.* 2006) states that Π is consistent w.r.t. a given state S if and only if $\mathsf{CONS}(\Pi, S)$ has a solution. However, solving the constraint system $\mathsf{CONS}(\Pi, S)$ is NP-hard. But if the set of SOMA-rules are ground, $\mathsf{CONS}(\Pi, S)$ would become linear constraints, and thus solving any linear objective function subject to $\mathsf{CONS}(\Pi, S)$ is a linear programming problem that can be efficiently solved by existing linear programming algorithms such as the Simplex algorithm and the Karmarkar's Interior Point algorithm.

Problem Definition

Suppose there are n agents interacting with each other. The set of agents is $\{\psi_1, \psi_2, \ldots, \psi_m\}$. We consider discrete time points and finite horizon only. Let $\mathsf{Time} = \{t_0, t_1, t_2, \ldots, t_m\}$ be the set of all time points at which agents can interact with each other. At any time point in Time, each agent can choose a finite number of actions and performs them simultaneously. This set of actions constitutes a course of action. Suppose the state of the world at time t_j is S^j. At time t_j, the agents can only chose actions whose precondition was satisfied at S^j. A *COA profile* at time t_j is a vector $\rho^j = \langle C_1^j, C_2^j, \ldots, C_n^j \rangle$, where C_k^j is the COA chosen by the agent ψ_k at t_j. After all agents choose their courses of actions, their actions will be executed at the same time. The state S^j will then advance towards the next state S^{j+1}, which is determined by the previous state S^j and the COA profile at time t_j. We assume that actions in a COA profile can always be executed at the same time without any conflict among the actions.

Agents will receive rewards after they perform their actions. The rewards depend on (1) the actions performed by *all* participating agents, and (2) the state at which the actions are executed. We model the reward of an agent ψ_k by a *reward function* R_k, such that $R_k(S, \rho)$ is the reward (a non-negative number) that the agent ψ_k should receive if the actions performed at state S are the actions in the COA profile ρ. We assume that the agents know this reward function.

The performance of an agent is measured by its *accumulated rewards* in the entire course of interactions. Suppose an agent earns r^j at time t_j. The accumulated reward of the agent is $\sum_{j=0}^{m} r^j$, where m is the total number of iterations. The objective of an agent is to maximize its accumulated reward by choosing COAs at every time point.

Our problem is to compute the optimal sequence of COAs for a particular agent, namely ψ_1, given that ψ_1 initially has (1) the initial state of the world, (2) the set of all actions, and (3) the models of other agents, written in SOMA-programs. We denote the SOMA-program of the agent ψ_k by Π_k, for $2 \leq k \leq n$. Our key assumption is that the agents' behavior always follows the given SOMA-programs; they never change their behavior during the entire course of the interaction.

Given a state S^j at time t_j, there is a set $\mathcal{I}_j^k$ of SOMA-interpretations for each agent ψ_k, for $2 \leq k \leq n$. Each element I_k in $\mathcal{I}_j^k$ is a probability distribution of COAs, such that ψ_k would choose a COA from the set $\mathcal{C}$ of all COAs for ψ_k according to the distribution I_k at time t_j. An *interpretation profile* is a vector $\langle I_2, I_3, \ldots, I_n \rangle$, where $I_k \in \mathcal{I}_j^k$ is a SOMA-interpretation for ψ_k. Notice that there is no interpretation I_1 in an interpretation profile, because ψ_1 is the subject of our problem.

Now we define the *expected accumulated rewards* ψ_1, given that ψ_1 chooses an COA C_1^j at time t_j and the interpretation profile is $I^j = \langle I_2^j, I_3^j, \ldots, I_n^j \rangle$, for $1 \leq j \leq m$. First, we define the probability of COA profiles as follows. Let the set of all COA profiles be $\Omega = \{\langle C_1, C_2, \ldots, C_n \rangle : C_1, C_2, \ldots, C_n \in \mathcal{C}\}$. For any $\rho^j = \langle C_1^j, C_2^j, \ldots, C_n^j \rangle \in \Omega$ chosen at time t_j, we define the *probability of* ρ^j to be $P(\rho^j) = \prod_{k=2}^n I_k^j(C_k^j)$. We can show that $\sum_{\rho^j \in \Omega} P(\rho^j) = 1$.

Then we define the transition probability from one state to another given a COA profile. Let $\mathsf{next}(S^j, \rho^j)$ be the set of all possible next states when all actions in the COA profile $\rho^j = \langle C_1^j, C_2^j, \ldots, C_n^j \rangle$ are executed at S^j. This set is determined by the effects of the actions in ρ^j. Let $\mathsf{eff}_k^j = \{\langle e_1, e_2, \ldots, e_{|\mathsf{eff}(a_l)|} \rangle : e_i \in \mathsf{eff}(a_l), a_l \in C_k^j\}$ be the set of all permutations of possible effects of ψ_k according to C_k^j. We define the probability of $\vec{e}_k \in \mathsf{eff}_k^j$ by $P(\vec{e}_k) = \prod_{e_i \in \vec{e}_k} \Delta_i(e_i)$, where Δ_i is the probability distribution over $\mathsf{eff}(a_l)$ as defined in the action a_l. We can show that $\sum_{\vec{e}_k \in \mathsf{eff}_k^j} P(\vec{e}_k) = 1$. An *effect profile* is a vector $\langle \vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n \rangle$, where $\vec{e}_k \in \mathsf{eff}_k^j$. The probability of an effect profile $E = \langle \vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n \rangle$ is $P(E) = \prod_{\vec{e}_k \in E} P(\vec{e}_k)$. Then we denote the next state by $S_E^{j+1} = \mathsf{apply}(S^j, \cup\{\vec{e}_k : \vec{e}_k \in E\}) \in \mathsf{next}(S^j, \rho^j)$ can be led from S^j by the effect profile E. The probability of reaching S_E^{j+1} from S^j via E of ρ^j is $P(S_E^{j+1}) = P(E)$.

Given an interpretation profile $I^j = \langle I_2^j, I_3^j, \ldots, I_n^j \rangle$ and the COAs $C_1^0, C_1^1, \ldots, C_1^{m-1}$ of agent ψ_1, the expected accumulated reward of ψ_1 starting from state S^j at time t_j, for $0 \leq j < m$, can be defined by the following recursive formula:

$$Expect(S^j) = \sum_{\rho^j \in \Omega} \left\{ P(\rho^j) \times \left[R_1(S^j, \rho^j) + g(\rho^j) \right] \right\}$$

where

$$g(\rho^j) = \sum_{S_E^{j+1} \in \mathsf{next}(S^j, \rho^j)} P(S_E^{j+1}) \times Expect(S_E^{j+1})$$

In addition, the expected accumulated reward of all states at the last interaction is zero. That is, $Expect(S^m) = 0$ for all $S^m \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. If the initial state is S_0^0, then the expected accumulated reward of the agent ψ_1 is $Expect(S_0^0)$, and this can be computed by the above recursive formula.

The expected accumulated reward depends on the interpretation profiles at every time points. However, there can be more than one interpretation profile satisfies with the SOMA-programs. So there can be many different expected accumulated awards. Therefore, we focus on the *lower bound* of the expected accumulated reward. First, a *COA policy* is a mapping from $\pi : \mathcal{S} \times Time \rightarrow \mathcal{C}$, such that $\pi(S^j, t_j)$ is the action ψ_1 would choose at state S^j at time t_j. So given a COA policy π, our question is, if ψ_1 chooses COAs according to π, what is the lowest possible expected accumulated reward ψ_1 could get? Our objective is to find a COA policy such that the lower bound can be maximized.

In summary, our problem is:

Definition 2 *Given (1) the number of agents n, (2) the number of time points m, (3) the initial state S^0, (4) the set $\mathcal{A}$ of all possible ground actions, (5) the reward function R_1 for the agent ψ_1, and (6) the SOMA-programs $\Pi_2, \Pi_3, \ldots, \Pi_m$ of the agent $\psi_2, \psi_3, \ldots, \psi_m$, find a COA policy π for ψ_1 such that the lower bound of the expected accumulated rewards of ψ_1 can be maximized.*

Several special features of this problem are: (1) there can be more than two agents; (2) agents can choose more than one action at a time; (3) the payoff matrix is not fixed; the payoff matrix can change from one state to another; and (4) SOMA-program permits a high degree of ignorance about the other agent's behavior; the agent modeler does not need to give a precise probability distribution of the COAs. We believe that these are important features in many applications.

An Algorithm for Computing the Maximum Lower Bound of the Expected Accumulated Rewards

In this section, we present an algorithm called MLBEAR for computing the maximum lower bound of the expected accumulated rewards. The algorithm is similar to the dynamic programming algorithm such as the value iteration algorithm for MDPs. First, it computes the values of all states at time t_m, and then the values at time t_{m-1}, and so forth, until the values at time t_0 are computed. The algorithm maintains two tables, namely V and π, to record (1) the maximum lower bound of the expected accumulated rewards and (2) the COA policy that produces this maximum lower bound. The pseudo-code of the MLBEAR algorithm is shown in Figure 1.

The idea is to build a $|\mathcal{S}| \times (|\mathsf{Time}| + 1)$ table, where $\mathcal{S}$ is the set of all possible states and Time is the set of all time points. The entry at the i'th row and the j'th column of the table stores (1) the maximum lower bound of the expected accumulated reward of the agent ψ_1 *if* the initial state is S^i and the initial time point is t_j; and (2) the COA that ψ_1

```
Function MLBEAR()
For $i = 0$ to $|\mathcal{S}| - 1$
    $v(S_i^{m+1}) = 0$
For $j = m$ down to $0$
    For $i = 0$ to $|\mathcal{S}| - 1$
        $(v(S_i^j), \pi(S_i^j, t_j)) = \mathsf{update}(i, j)$
return $v(S_0^0)$

Function update$(i, j)$
Let $\mathcal{C}_1$ be the set of all COAs of the agent $\psi_1$ that are applicable to $S_i^j$
For each $C \in \mathcal{C}_1$
    $v(C) = \mathsf{minvalue}(2, \{C\})$
$C_{max} := \arg\max_{C \in \mathcal{C}_1} v(C)$
Return $(v(C_{max}), C_{max})$        // the value of the max node and the COA

Function minvalue$(k, \rho)$
If $k \leq n$, then
    Let $\mathcal{C}_k$ be the set of all COAs of the agent $\psi_l$ that are applicable to $S_i^j$
    For each $C_l \in \mathcal{C}_k$
        $v_l := \mathsf{minvalue}(k + 1, \rho \cup \{C_l\})$
    Minimize $\sum_{1 \leq l \leq |\mathcal{C}_k|} \{v_l \times p_l\}$ subject to the constraints CONS$(\Pi_k, S_i^j)$
    Return the minimum value        // the value of the LB node
Else
    $\mathcal{S}_{next} := \mathsf{next}(S_i^j, \rho)$        // the set of all possible next states after executing $\rho$
    $r := R_1(S_i^j, \rho)$        // the reward earned by $\psi_1$ at time $t_j$ when executing $\rho$ at $t_j$
    return $r + \sum_{S \in \mathcal{S}_{next}} \{P(S) \times v(S)\}$        // the value of the reward node
```

Figure 1: The pseudo-code of MLBEAR algorithm, the dynamic algorithm for computing the maximum lower bound of expected accumulated reward (MLBEAR).

should take at time t_j in order to obtain the maximum lower bound. We denote the maximum lower bound and the COA of the (i, j) entry by $v(S_i^j)$ and $\pi(S_i^j, t_j)$, respectively. We can see that the maximum lower bound of the expected accumulated rewards would be $v(S_0^0)$, where S_0^0 is the initial state. After we build such a table, we can look up the solution $v(S_0^0)$ and the COA policy π from the table. Here we make the assumption of full observability, i.e., ψ_1 knows which state it is in at any time point, such that ψ_1 can use the COA policy π to determine what action it should take at the observed state at different time in order to get the reward $v(S_0^0)$.

The MLBEAR algorithm is a dynamic programming algorithm for computing the tables. Initially, the algorithm sets $v(S_i^{m+1})$ to zero for all $0 \leq i < |\mathcal{S}|$. Then the table is filled out backward in time by a function called update, which computes and returns a pair $(v(S_i^j), \pi(S_i^j, t_j))$. After all entries are filled out, the maximum lower bound of the expected accumulated reward of ψ_1 is $v(S_0^0)$.

The Update Tree

The update function in the MLBEAR algorithm relies on another function called minvalue. The computations of these functions can be illustrated by the *update tree* as shown Figure 2. There are four kinds of nodes in the update tree: (1) the *state nodes* (the circles) correspond to states at different times; (2) the *max nodes* (the upward triangle) are the decision nodes of the agent ψ_1; (3) the *LB nodes* (the downward triangles) are the decision nodes of the other agents ψ_2, ψ_3, ..., ψ_m; and (4) the *reward nodes* (the square nodes) correspond to the COA profiles. The root node of the tree is the state node of the state S_i^j whose entry in the table has yet to be filled out. The leaf nodes of the tree are the state nodes of the states at time t_{j+1} that are possibly reachable by some COAs from S_i^j. Below the root node is the max node, and above the leaf nodes is a layer of reward nodes. Between the max node and the layer of reward nodes are $(n-1)$ layers of LB nodes (LB stands for "lower bound"). There is a layer of triangle nodes (the max node and the LB nodes) for each agent.

Now let us explain the meaning of the edges in the update tree. Let $\mathcal{C}_i$ be the set of all applicable COAs that the agent ψ_i can choose at S_i^j. For each COA $C \in \mathcal{C}_1$, there is an edge emanating from the max node and connecting to a LB node for the agent ψ_2. In Figure 2, there are three edges between the max node and the layer of LB nodes because $|\mathcal{C}_1| = 3$. Likewise, every LB node for the agent ψ_k has $|\mathcal{C}_k|$ child nodes. The COAs on the path from the max node to a reward node σ^{n+1} constitutes a COA profile ρ that is feasible w.r.t. state S_i^j. This construction makes sure that every feasible COA profiles is represented by some path between the max node to the layer of reward nodes. There is an edge between a reward node and a state nodes S_i^{j+1} if S_i^{j+1} a state that is reachable from S_i^j by some effect profile of the COA profile ρ at S_i^j.

Notice that the ordering of the other agents in the construction of the tree does not affect our result; but we believe

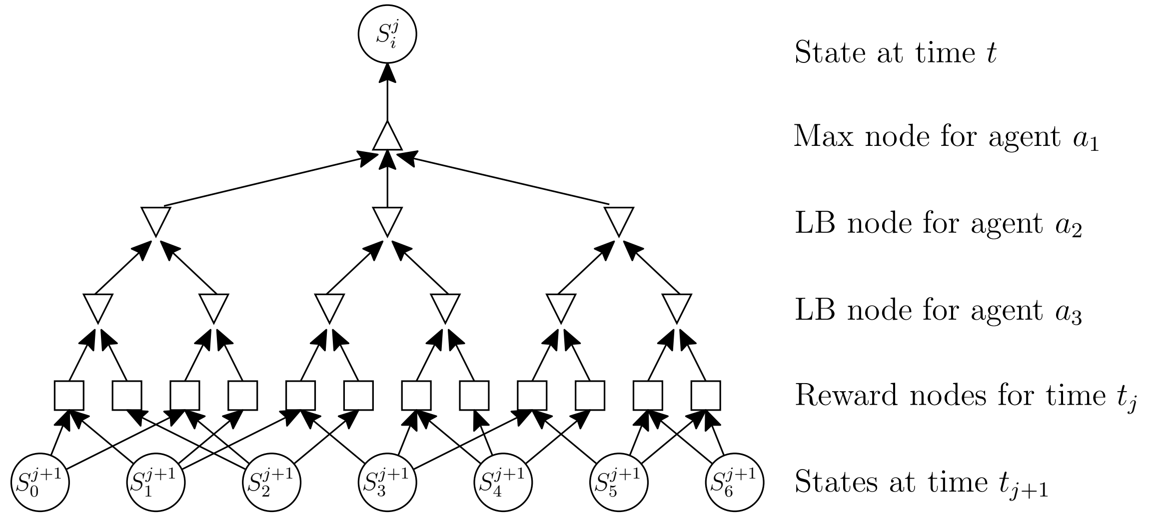

Figure 2: The propagation of values in the update function and the minvalue function. $n = 3$, $|\mathcal{C}_1| = 3$, and $|\mathcal{C}_2| = |\mathcal{C}_3| = 2$.

that our algorithm would run faster if we order the agents according to the number of applicable COAs, such that the LB nodes with large branching factors are at the top of the update tree. The reason is that this ordering can reduce the size of the update tree.

Value Propagation on the Update Tree

The computation of $v(S_i^j)$ in the function update and minvalue is graphically equivalent to the propagation of the values of the state nodes at time t_{j+1} to the root node over the update tree. There is a value associated with each internal node of the tree, and we denote the value of the internal node σ by $v(\sigma)$. $v(\sigma)$ depends only on the value of the child nodes of σ. The algorithm first computes the values of the reward nodes, then the values of the LB nodes, and finally the value of the max node.

Consider a reward node σ^{n+1}. Let ρ be the COA profile corresponding to the path from the max node to σ^{n+1}. The algorithm computes $v(\sigma^{n+1})$ from the value of the states nodes for the states in $\mathsf{next}(S_i^j, \rho)$ by the following equation:

$$v(\sigma^{n+1}) = R_1(S_i^j, \rho) + \sum_{S_E^{j+1} \in \mathsf{next}(S_i^j, \rho)} P(S_E^{j+1}) \times v(S_E^{j+1}),$$

where E is the effect profile and $R_1(S_i^j, \rho)$ is the reward of the agent ψ_1 if the COA profile at S_k^j is ρ. $v(\sigma^{n+1})$ is the lower bound of the expected accumulated reward given that (1) the agents choose COAs in ρ, and (2) the agent ψ_1 choose COAs that yield an maximum lower bound of the expected accumulated reward starting from time t_{j+1}. The pseudo-code of this equation is shown in Line 8–10 of the minvalue function in Figure 1.

The value of LB nodes can be computed as follows. Consider a LB node σ^k for the agent a_k. Let ρ' be the set of COAs corresponding to the path from the max node to σ^k, and let $\mathsf{child}(\sigma^k) = \{\sigma_1^{k+1}, \sigma_2^{k+1}, \ldots, \sigma_{|\mathcal{C}_k|}^{k+1}\}$ be the set of all child nodes of σ^k. The value of the LB node σ^k is the lower bound of the expected accumulated reward given that (1) the agents $\psi_1, \psi_2, \ldots, \psi_{k-1}$ choose the COA in ρ', and (2) the agent ψ_k choose COAs that yield an maximum lower bound of the expected accumulated reward starting from time t_{j+1}. Since we don't know which COA the agent ψ_k would choose, we will rely on the given SOMA-program Π_k of the agent ψ_k to determine the probability distribution of COAs for ψ_k. Consider the set of constraints of $\mathsf{CONS}(\Pi_k, S_i^j)$. A solution of this set of constraints is a probability distribution of the COAs for ψ_k at S_i^j. Let p_l be the variable denoting the probability that the ψ_k would choose the COA C_l and the child LB node is σ_l^{k+1}, for $1 \leq l \leq |\mathcal{C}_k|$. Then consider the optimization problem whose objective is to minimize the value of

$$\sum_{1 \leq l \leq |\mathcal{C}_k|} \left\{ p_l \times v(\sigma_l^{k+1}) \right\},$$

subject to the constraints $\mathsf{CONS}(\Pi_k, S_i^j)$. Then we argue that the minimum value of this optimization problem is the lower bound for the value of σ^k. In short, the reason is that $v(\sigma_l^{k+1})$ is the minimum value, and even if we substitute a larger value for $v(\sigma_l^{k+1})$ in the objective function, the value of the objective function would not decrease; thus $v(\sigma^k)$ is the minimum value. The pseudo-code of this computation is shown in Line 2–6 of the minvalue function in Figure 1.

The max node σ^1 represents the decision node of the agent ψ_1 and thus we should maximize the value of the max node. The update function in Figure 1 assigns $max_{\sigma_l^2 \in \mathsf{child}(\sigma^1)}\{v(\sigma_l^2)\}$ to $v(\sigma^1)$, where $\mathsf{child}(\sigma^1)$ is the set of child nodes of σ^1. The value of $V(S_i^j)$ is $v(\sigma^1)$ and $\pi(S_i^j, t_j)$ is $\arg\max_{\sigma_l^2 \in \mathsf{child}(\sigma^1)}\{v(\sigma_l^2)\}$.

Summary

A common problem that policy makers need to deal with is how their organization can repeatedly interact with other organizations such that the long-term utility of their organization can be maximized, given that the policy makers have little information about the decision making process of other organizations. We formulate this problem as an extension of the conventional repeated game models or Markov Games (Littman 1994) by incorporating several useful features: (1) the use of probability intervals to handle missing information; and (2) the state-dependent payoff matrix—the reward function can change from one state to another.

We model the behavior of other agents by SOMA-programs, a rich language for modeling agents. SOMA-programs permit a high degree of ignorance about the other agent's behavior—the agent modeler does not need to give the exact probability distribution of the courses of action that the other agents would choose given a state. We proposed an algorithm to compute a policy such that the lower bound of the expected accumulated rewards of the agent can be maximized.

In future, we would like to address the following issues:

- Like many dynamic programming algorithms such as the value iteration algorithm for MDPs, our algorithm suffers from the curse of dimensionality—the MLBEAR algorithm enumerates all possible states in order to compute the lower bound of the expected accumulated rewards. The running time of the MLBEAR algorithm can be very large, since the number of states is exponential to the number of atoms, and in practical applications we need a large number of atoms to model a game properly. Recently, there has been work on approximate dynamic programming, which uses various approximation techniques to cope with the curse of dimensionality. In future, we would like to see how to apply these approximation schemes to the MLBEAR algorithm.
- Our key assumption is that the behavior of the other agents would not change during the entire course of the interactions. This assumption holds in some domains (e.g., in the manufacturer-suppliers problem). If this assumption does not hold, we need to update the SOMA-programs during the course of interaction. Therefore, we want to see how to incorporate online SOMA-programs learning method in the MLBEAR algorithm.
- In future, we will implement and evaluate the MLBEAR algorithm.

References

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artif. Intel.* 89(1–2):219–283.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artif. Intel.* 75(1–2):239–286.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *ICML*.

Simari, G.; Sliva, A.; Nau, D.; and Subrahmanian, V. 2006. A stochastic language for modelling opponent agents. In *AAMAS*. To appear.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *JAIR* 24:851–887.