Lossy Stochastic Game Abstraction with Bounds*

Tuomas Sandholm
Computer Science Department
Carnegie Mellon University

Satinder Singh
Computer Science & Engineering
University of Michigan

ABSTRACT

Abstraction followed by equilibrium finding has emerged as the leading approach to solving games. Lossless abstraction typically yields games that are still too large to solve, so lossy abstraction is needed. Unfortunately, prior lossy game abstraction algorithms have no guarantees on solution quality. We developed a framework that enables the design of lossy game abstraction algorithms with guarantees on solution quality. It simultaneously handles state and action abstraction. We define a measure of reward approximation error and transition probability error achieved by state and action abstraction in stochastic games such that the regret of the equilibrium found in the abstract game when implemented in the original, unabstracted game is upper-bounded by a function of those measures. We then develop the first lossy game abstraction algorithms with bounds on solution quality. Both of them work level-by-level up from the end of the game. One of the algorithms is greedy and the other is an integer linear program. We also prove that the abstraction problem is NP-complete (even with just action abstraction, 2 agents, and a 1-step game), but point out that this does not mean that the game abstraction problems that occur in practice cannot be solved quickly.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; J.4 [**Social and behavioral sciences**]: Economics; I.2.1 [**Artificial intelligence**]: Applications and Expert Systems: Games

General Terms

Algorithms, Economics, Theory

Keywords

Game theory, game solving, abstraction, game abstraction, equilibrium finding, ϵ-equilibrium

1. INTRODUCTION

Game-theoretic solution (equilibrium) concepts provide a rigorous definition of how rational agents should act in multiagent settings. The ability to actually compute such solutions is a key capability in a variety of applications, for example, auctions, exchanges, negotiation, security games, and recreational games such as poker and billiards. Computational techniques for finding such solutions to games have therefore emerged as a central topic in electronic commerce research (e.g., [19, 18, 10, 16]) and research in the intersection of economics and computer science at large. This research has focused both on developing new game-solving techniques and on using game-solving techniques to answer important questions in a variety of applications.

The following paradigm has emerged as a leading approach to solving large games. First, the game is *abstracted* to form a smaller game. Then the abstracted game is solved with some equilibrium-finding algorithm. Finally, that equilibrium is mapped back into the original, unabstracted game [13, 21, 6].

There are at least three motivations for abstracting games:

(i) The original, unabstracted game can be prohibitively large to be solved with the equilibrium-finding algorithm directly. It may take too much time and/or require too much memory (e.g., [3]).

(ii) The original game might be so complex that it is difficult to even model without making the model be an abstraction of reality (e.g., [26]).

(iii) The original game may not fall into a class for which existence of equilibrium is known or may nor fall into a class for which an equilibrium-finding algorithm exists; abstraction can remedy this (e.g., [2]).

Game abstraction typically takes the form of *state abstraction* and/or *action abstraction*. In state abstraction, states of the game are bundled together so the agent(s) whose turn it is to move cannot distinguish among those states. This means that the abstraction pretends that the agent does not know some of the history that the agent actually knows. Therefore, state abstraction is often also referred to as *information abstraction*. State abstraction means that the agent has to use the same probability mixture over his actions in each of the states in the bundled, abstract state. So, state abstraction amounts to a restriction on the agent's strategy space. Action abstraction means that the actions available to an agent at any point in the abstracted game are not necessarily the same as (typically fewer in number than) in the original, unabstracted game.

*An extended version of this paper has been accepted for publication in the *ACM Conference on Electronic Commerce (EC)*, 2012. Tuomas Sandholm was supported by the National Science Foundation under grants IIS-0964579, IIS-0905390, and CCF-1101668. Satinder Singh was supported by the National Science Foundation under grants IIS-1148668 and IIS-0905146.

In early uses of game abstraction, the abstraction was generated manually using domain knowledge (e.g., [3]). Similarly, when abstraction is used for reason (ii) or (iii) above, it is typically done manually [26, 2]. When state abstraction is done for reason (i), it is nowadays usually done automatically using some abstraction algorithm [10, 22, 7, 8, 9, 12, 27, 13, 11]. Action abstraction is still typically done by hand [13, 21], but that is starting to change [15].

A key question is how good the abstraction is, that is, how good are the equilibrium strategies from the abstracted game when evaluated in the real, original game. A domain-independent lossless state abstraction algorithm was recently developed for a broad class of games [10]. By 'lossless' we mean that the equilibrium from the abstracted game is an exact equilibrium of the original game. For many real games—such as Texas Hold'em poker—the losslessly abstracted game is still too large to solve. Therefore, one needs to employ lossy abstraction. Unfortunately, all prior lossy game abstraction algorithms (e.g., [22, 10, 7, 8, 9, 12, 27, 13, 11, 24, 25]) and manually-generated game abstractions (e.g., [3]) are lossy without bound. That is, they have no guarantee on how good the equilibrium strategies from the abstracted game will be in the original game. An important open question in this field—and arguably in game solving at large—is whether lossy abstraction algorithms with bounds on solution quality can be devised.

In this paper we develop a framework that enables the design of such algorithms. It simultaneously handles state and action abstraction. It is based on bounding the errors in payoffs as well as errors in transition probabilities, caused by abstraction. Specifically, we define a measure of reward approximation error and transition probability approximation error achieved by state and action abstraction such that the regret of the equilibrium found in the abstract game when implemented in the original, unabstracted game is upper-bounded by some function of those measures. The analysis is in some ways similar to that of abstraction in Markov decision processes [14, 20, 23], but for the richer—and much more difficult—setting of games.

We then develop the first lossy game abstraction algorithms with bounds on solution quality. We analyze finite stochastic general-sum games with any number of agents. (However, as we will discuss later, we expect the approach to generalize easily to discounted infinite stochastic games.) We also prove that the abstraction problem is NP-complete even if one looks at one level of the game and does only action abstraction.

The rest of this paper is organized as follows. First, Section 2 illustrates the difficulty of game abstraction: even in a simple game with just action abstraction, a strict refinement of the abstraction can lead to worse solution quality. Then, Section 3 presents our analysis framework. Section 4 proves that state evaluations in the abstract game are near correct in the original game, and Section 5 proves that equilibria from the abstract game have bounded regret in the original game. Section 6 presents abstraction algorithms and analyzes the problem complexity. Finally, Section 7 presents conclusions and discusses future research directions.

2. EXAMPLE OF NON-MONOTONICITY IN ABSTRACTING GAMES

Abstraction in games is a much more difficult problem than abstraction in single-agent settings. In short, the difficulty stems from the fact that the opponent(s) may not honor the abstraction when playing. In this section we will illustrate the resulting non-monotonicity in abstracting games.

It is well known that in extensive form games, an equilibrium strategy derived in a finer-grained abstraction can be more exploitable in the original, unabstracted game than an equilibrium strategy derived in a coarser abstraction. Such abstraction pathologies have been demonstrated also experimentally, in relatively small extensive form (artificial poker) games [24].

The example below shows that abstraction pathologies can occur already in zero-sum two-agent one-step stochastic games (i.e., strategic form games). This is the case even if only action abstraction is used.

EXAMPLE 1. *Consider a game between an attacker and a defender, with two locations, A and B. There is only one state and the agents move simultaneously, after which the game ends. The attacker has two possible actions, A and B, corresponding to which location he attacks. The defender has three actions, A, B, and BETWEEN, corresponding to where he defends. If the agents choose the same location, the defender wins, in which case he gets payoff 1 and the attacker gets payoff -1. If the defender choose A or B and the attacker chooses the opposite location, the attacker wins and gets payoff 1 while the defender gets payoff -1. If the defender choose BETWEEN, the game is a draw: both agents get payoff 0.*

In each equilibrium, the attacker randomizes 50-50 between the locations. The defender plays A with probability p and B with probability p, and BETWEEN with probability $1 - 2p$. *There is an equilibrium for each* $p \in [0, 0.5]$.

Now, consider an abstraction where the attacker only has action A. The defender would choose A, but that is far from equilibrium in the original, unabstracted game, yielding regret 1 because the attacker would choose B.

Now, consider a strictly coarser abstraction where the attacker only has action A and the defender only has actions B and BETWEEN. The defender would choose BETWEEN in the abstracted game. Interestingly, that choice has no regret in the original, unabstracted game because that is an equilibrium strategy in that game!

3. FRAMEWORK

In this section we will present our analysis framework. We will consider stochastic games that have a finite number of agents and a finite set of possible actions at each state for each agent. We will present everything in the finite-horizon undiscounted setting (but we expect the results to extend easily to the infinite-horizon discounted setting, as we will discuss later).

If the stochastic game is given as a directed graph with cycles, then we conceptually attach time to the states so that if a state can be reached via two different-length paths in the graph, we consider that state to be two different states. However, if a state can be reached via different paths of the same length, we do not duplicate the state. (This guarantees that the number of states in the game is linear in the number of steps that the game is played.) As a result, our game representation is a layered directed acyclic graph (LDAG). Layer 1 consists of the start state, layer 2 consists of the

states reachable in one step, and so on. Throughout the paper we will refer to the *height* of a state, which is equivalent to layer, except counting from the other direction, so leaves are at height 1, states just before the leaves are at height 2, and so on. We will use the word *level* as a synonym for height.

Consider two stochastic games: the original, unabstracted one, M, and the abstracted one, M'. Each of the two games has n agents.

We will denote the elements of M as follows. Its state space is S. The set of states at height j is $S_j \subset S$ (note, for all $j \neq k$, $S_j \cap S_k = \phi$). In state $s \in S$ the action space for Agent i is $A_i(s)$ and the joint action space of all n agents is $A(s)$. In state $x \in S_j$, on taking joint action $\mathbf{a} \in A(x)$, the reward to Agent i is $R_i(x, \mathbf{a})$ and the next state is $y \in S_{j-1}$ with probability $T(x, \mathbf{a}, y)$. The action of Agent i in joint action $\mathbf{a}$ is denoted $\mathbf{a}_i$.

The elements of M' are denoted exactly as for M above except that throughout a "prime" superscript is used for all the elements, e.g., the state space of M' is denoted S'.

3.1 State and action abstraction functions

We will define a state abstraction function $h : S \to S'$. Also, for all Agents i and for all $s \in S$, we will define an action abstraction function $g_{s,i} : A_i(s) \to A'_i(h(s))$. We will have these functions be surjections, so $|S'| \leq |S|$ and for all agents i and for all $s \in S$, $|A'_i(h(s))| \leq |A_i(s)|$. The intent is that M' is smaller than M and thus we will refer to M' as the abstract game. When we wish to emphasize this aspect we will attach the prefix "abstract" to the elements associated with M'.

State s in M maps to abstract state $h(s)$ in M'. The set of states that map to abstract state s' is denoted $h^{-1}(s') \subset S$ (i.e., h induces a partition of S).

Action $\mathbf{a}$ in state s in M maps to abstract action $g_s(\mathbf{a})$ in abstract state $h(s)$ in M'. In state s, the set of actions for Agent i that map to abstract action $\mathbf{a}'_i$ is denoted $g^{-1}_{s,i}(\mathbf{a}'_i)$, and the set of joint actions that map to abstract joint action $\mathbf{a}'$ is denoted $g_s^{-1}(\mathbf{a}')$. Note that the action abstraction functions are factored by both state and action, i.e., $g_s^{-1}(\mathbf{a}') = \times_{i=1}^n g^{-1}_{s,i}(\mathbf{a}'_i)$ (where the symbol $\times$ denotes cross-product).

An abstract strategy σ' maps abstract states of M' to abstract joint-actions in M'. The strategy for Agent i is denoted σ'_i.

How does one take a strategy σ' in the abstract game M' and apply it in M? There are many ways of doing this "lifting". That is, for state s the probability $\sigma'_i(\mathbf{a}'_i|h(s))$ of abstract action $\mathbf{a}'_i$ by Agent i can be apportioned arbitrarily among the actions $g^{-1}_{s,i}(\mathbf{a}'_i)$ in the lifted strategy.

DEFINITION 1. *(Strategy Lifting) Given an abstract strategy σ'_i for Agent i in M', a lifted strategy for Agent i in M, denoted $\sigma_i^{\uparrow\sigma'_i}$ is any strategy in M that satisfies the following conditions: for all $s \in S$, for all i, for all $\mathbf{a}'_i \in A'_i$,*

$$\sum_{\mathbf{a}_i \in g^{-1}_{s,i}(\mathbf{a}'_i)} \sigma_i^{\uparrow\sigma'_i}(\mathbf{a}_i|s) = \sigma'_i(\mathbf{a}'_i|h(s)).$$

Furthermore, $\sigma_i^{\uparrow\sigma'}$ is the same as $\sigma_i^{\uparrow\sigma'_i}$, and $\sigma^{\uparrow\sigma'}$ assigns $\sigma_i^{\uparrow\sigma'_i}$ to each i.

Conversely, how does one take a strategy σ defined in M and apply it in M'? One cannot do this in general. However, by construction, the strategy $\sigma^{\uparrow\sigma'}$ defined in M can always be applied in M'.

3.2 Value functions for the original and abstract game

We define value functions for the games as follows. For joint strategy σ in M, for Agent i the value of a non-terminal state $x \in S_{k>1}$ is

$$V_i^\sigma(x) = \sum_{\mathbf{a} \in A(x)} \sigma(\mathbf{a}|x)\Big[R_i(x, \mathbf{a}) + \sum_{y \in S_{k-1}} T(x, \mathbf{a}, y) V_i^\sigma(y)\Big] \quad (1)$$

where for terminal states $x \in S_1$,

$$V_i^\sigma(x) = \sum_{\mathbf{a} \in A(x)} \sigma(\mathbf{a}|x) R_i(x, \mathbf{a}).$$

For joint abstract strategy σ' in M', the value of Agent i in abstract non-terminal state $x' \in S'_{k>1}$ is

$$W_i^{\sigma'}(x') = \sum_{\mathbf{a}' \in A'(x')} \sigma'(\mathbf{a}'|x')\Big[R'_i(x', \mathbf{a}') + \sum_{y' \in S'_{k-1}} T'(x', \mathbf{a}', y') W_i^{\sigma'}(y')\Big] \quad (2)$$

where for terminal states $x' \in S'_1$,

$$W_i^{\sigma'}(x') = \sum_{\mathbf{a}' \in A'(x')} \sigma'(\mathbf{a}'|x') R'_i(x', \mathbf{a}').$$

Note that we will use V's to denote values in M and W's to denote values in M'. Finally, define the largest, over all states in LDAG-level k, value obtained in M' under abstract strategy σ' as

$$\overline{W}_{k,i}^{\sigma'} \stackrel{\text{def}}{=} \max_{x' \in S'_k} W_i^{\sigma'}(x')$$

and define

$$\overline{W}_k^{\sigma'} \stackrel{\text{def}}{=} \max_i \overline{W}_{k,i}^{\sigma'} \quad \text{and} \quad \overline{W}^{\sigma'} \stackrel{\text{def}}{=} \max_k \overline{W}_k^{\sigma'}.$$

3.3 Reward-approximation and transition-approximation error bounds

Next we define reward-approximation and transition-approximation error bounds in the mapping from M to M', as a function of LDAG-level and agent:

$$\epsilon^R_{k,i} \stackrel{\text{def}}{=} \max_{s \in S_k, \mathbf{a} \in A(s)} |R_i(s, \mathbf{a}) - R'_i(h(s), g_s(\mathbf{a}))|, \quad (3)$$

then as a function of just LDAG-level:

$$\epsilon^R_k \stackrel{\text{def}}{=} \max_i [\epsilon^R_{k,i}], \quad (4)$$

and finally as a global bound on the entire game:

$$\epsilon^R = \max_k \epsilon^R_k. \quad (5)$$

Similarly we can define transition-approximation error bounds, first as a function of LDAG-level and next abstract state:

$$\epsilon^T_k(x' \in S'_{k-1}) \stackrel{\text{def}}{=} \max_{s \in S_k, \mathbf{a} \in A(s)} \Big|\sum_{x \in h^{-1}(x')} T(s, \mathbf{a}, x) - T'(h(s), g_s(\mathbf{a}), x')\Big| \quad (6)$$

then as a function of just LDAG-level:

$$\epsilon^T_k \stackrel{\text{def}}{=} \sum_{x' \in S'_{k-1}} \epsilon^T_k(x'), \quad (7)$$

and finally as a global bound on the entire game:

$$\epsilon^T \stackrel{\text{def}}{=} \max_k \epsilon_k^T. \quad (8)$$

4. EVALUATIONS IN THE ABSTRACT GAME ARE NEAR CORRECT IN THE ORIGINAL GAME

Our first result is to show that the evaluation of an abstract strategy in M' is not too far from the evaluation of *any* corresponding lifted strategy in M. The key insight here is that we can state such an approximation result for entire, arbitrary strategy profiles (which contain one strategy for each agent)—rather than, say, studying equilibrium properties or best-response properties of individual strategies or strategy profiles. We will leverage this insight (as embodied in Proposition 1) when we present our game-theoretic results in the rest of the paper.

PROPOSITION 1. $\forall \sigma', \forall s \in S_k, \forall i,$

$$|V_i^{\sigma^\uparrow \sigma'}(s) - W_i^{\sigma'}(h(s))| \le f_{k,i} \stackrel{\text{def}}{=} \sum_{j=1}^{k} \epsilon_{j,i}^R + \sum_{j=1}^{k-1} \overline{W}_{j,i}^{\sigma'} \epsilon_{j+1}^T.$$

PROOF. By induction on the height of the LDAG.

Base case (terminal nodes of the LDAG): $\forall x \in S_1$,

$$\begin{aligned}
V_i^{\sigma^\uparrow \sigma'}(x) &\stackrel{\text{def}}{=} \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) R_i(x, \mathbf{a}) \\
&\le \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \Big[R_i'(h(x), g_x(\mathbf{a})) + \epsilon_{1,i}^R \Big]; \text{ by Eq. 3} \\
&= \epsilon_{1,i}^R + \sum_{\mathbf{a}' \in A'(h(x))} \sum_{\mathbf{a} \in g_x^{-1}(\mathbf{a}')} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) R_i'(h(x), g_x(\mathbf{a})) \\
&= \epsilon_{1,i}^R + \sum_{\mathbf{a}' \in A'(x)} \sigma'(\mathbf{a}'|h(x)) R_i'(h(x), \mathbf{a}'); \text{ by Def. 1} \\
&= \epsilon_{1,i}^R + W_i^{\sigma'}(h(x)) \quad (9)
\end{aligned}$$

$$\begin{aligned}
W_i^{\sigma'}(h(x)) &\stackrel{\text{def}}{=} \sum_{\mathbf{a}' \in A'(h(x))} \sigma'(\mathbf{a}'|h(x)) R_i'(h(x), \mathbf{a}') \\
&= \sum_{\mathbf{a}' \in A'(h(x))} \Big(\sum_{\mathbf{a} \in g_x^{-1}(\mathbf{a}')} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \Big) R_i'(h(x), \mathbf{a}'); \text{ by Def. 1} \\
&\le \sum_{\mathbf{a}' \in A'(h(x))} \sum_{\mathbf{a} \in g_s^{-1}(\mathbf{a}')} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \Big[R_i(x, \mathbf{a}) + \epsilon_{1,i}^R \Big]; \text{ by Eq. 3} \\
&= V_i^{\sigma^\uparrow \sigma'}(x) + \epsilon_{1,i}^R \quad (10)
\end{aligned}$$

Putting the two pieces (Equations 9 and 10) together, we have $W_i^{\sigma'}(h(x)) - \epsilon_{1,i}^R \le V_i^{\sigma^\uparrow \sigma'}(x) \le W_i^{\sigma'}(h(x)) + \epsilon_{1,i}^R$

Step 2: We assume that the proposition holds for $x \in S_{\le k}$.

Step 3: We prove that the assumption from Step 2 implies that the proposition holds for $x \in S_{k+1}$. We will only show one side of the two-sided inequality in the Proposition. The other side is similarly derived.

$$\begin{aligned}
V_i^{\sigma^\uparrow \sigma'}(x) &\stackrel{\text{def}}{=} \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \Big[R_i(x, \mathbf{a}) + \sum_{y \in S_k} T(x, \mathbf{a}, y) V_i^{\sigma^\uparrow \sigma'}(y) \Big] \quad (11) \\
&= \text{term1} + \text{term2}
\end{aligned}$$

$$\begin{aligned}
\text{term1} &\stackrel{\text{def}}{=} \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) R_i(x, \mathbf{a}) \\
&\le \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \Big[R_i'(h(x), \mathbf{a}') + \epsilon_{k+1,i}^R \Big] \\
&= \epsilon_{k+1,i}^R + \sum_{\mathbf{a}' \in A'(h(x))} \sum_{\mathbf{a} \in g_x^{-1}(\mathbf{a}')} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) R_i'(h(x), \mathbf{a}') \\
&= \epsilon_{k+1,i}^R + \sum_{\mathbf{a}' \in A'(h(x))} \sigma'(\mathbf{a}'|h(x)) R_i'(h(x), \mathbf{a}') \quad (12)
\end{aligned}$$

$$\begin{aligned}
\text{term2} &\stackrel{\text{def}}{=} \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \sum_{y \in S_k} T(x, \mathbf{a}, y) V_i^{\sigma^\uparrow \sigma'}(y) \\
&\le \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \sum_{y \in S_k} T(x, \mathbf{a}, y) \Big[W_i^{\sigma'}(h(y)) + f_{k,i} \Big]; \text{ by Step 2} \\
&= f_{k,i} + \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \sum_{y \in S_k} T(x, \mathbf{a}, y) W_i^{\sigma'}(h(y)) \\
&= f_{k,i} + \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \sum_{y' \in S_k'} \sum_{y \in h^{-1}(y')} T(x, \mathbf{a}, y) W_i^{\sigma'}(y') \\
&\le f_{k,i} + \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \sum_{y' \in S_k'} \Big(T'(h(x), g_x(\mathbf{a}), y') + \epsilon_{k+1}^T(y') \Big) W_i^{\sigma'}(y') \\
&= f_{k,i} + \sum_{y' \in S_k'} W_i^{\sigma'}(y') \epsilon_{k+1}^T(y') \\
&\qquad + \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \sum_{y' \in S_k'} T'(h(x), g_x(\mathbf{a}), y') W_i^{\sigma'}(y') \\
&\le f_{k,i} + \overline{W}_{k,i}^{\sigma'} \epsilon_{k+1}^T + \sum_{\mathbf{a} \in A(x)} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) \sum_{y' \in S_k'} T'(h(x), g_x(\mathbf{a}), y') W_i^{\sigma'}(y') \\
&= f_{k,i} + \overline{W}_{k,i}^{\sigma'} \epsilon_{k+1}^T \\
&\quad + \sum_{\mathbf{a}' \in A'(h(x))} \sum_{y' \in S_k'} \sum_{\mathbf{a} \in g_x^{-1}(\mathbf{a}')} \sigma^{\uparrow \sigma'}(\mathbf{a}|x) T'(h(x), \mathbf{a}', y') W_i^{\sigma'}(y') \\
&= f_{k,i} + \overline{W}_{k,i}^{\sigma'} \epsilon_{k+1}^T \\
&\quad + \sum_{\mathbf{a}' \in A'(h(x))} \sigma'(\mathbf{a}'|h(x)) \sum_{y' \in S_k'} T'(h(x), \mathbf{a}', y') W_i^{\sigma'}(y') \quad (13)
\end{aligned}$$

Using Equations 12 and 13 we get the following:

$$\begin{aligned}
V_i^{\sigma^\uparrow \sigma'}(x) &\le f_{k,i} + \epsilon_{k+1,i}^R + \overline{W}_{k,i}^{\sigma'} \epsilon_{k+1}^T \\
&+ \sum_{\mathbf{a}' \in A'(h(x))} \sigma'(\mathbf{a}'|h(x)) \Big[R_i'(h(x), \mathbf{a}') \\
&\qquad + \sum_{y' \in S_k'} T'(h(x), \mathbf{a}', y') W_i^{\sigma'}(y') \Big] \\
&= W_i^{\sigma'}(h(x)) + (\epsilon_{k+1,i}^R + \overline{W}_{k,i}^{\sigma'} \epsilon_{k+1}^T) + f_{k,i}. \quad (14)
\end{aligned}$$

This sets up the recursion

$$f_{k+1,i} = \epsilon_{k+1,i}^R + \overline{W}_{k,i}^{\sigma'} \epsilon_{k+1}^T + f_{k,i}$$

where $f_{0,i} = 0$, and $\epsilon_1^T = 0$, and $\overline{W}_{0,i}^{\sigma'} = 0$. □

Proposition 1 was presented in a manner that allowed the use of level-dependent reward and transition error bounds. By using the largest error bounds across levels, we present a simpler form of the result in the following corollary.

COROLLARY 1. $\forall \sigma', \forall s \in S_k, \forall i,$

$$|V_i^{\sigma^\uparrow \sigma'}(s) - W_i^{\sigma'}(h(s))| \le k \Big[\epsilon^R + \overline{W}_k^{\sigma'} \epsilon^T \Big].$$

Observe that the overall error in value functions is linear in the height of the LDAG (the additional dependence on k comes from the linear dependence on k of $\overline{W}_k^{\sigma'}$).

5. EQUILIBRIA FROM THE ABSTRACT GAME HAVE BOUNDED REGRET IN THE ORIGINAL GAME

In this section we bring in game-theoretic reasoning while leveraging the approximation results from the previous section that apply to all strategy profiles. We will show that any subgame perfect Nash equilibrium (SPNE) in an abstract game M' is an approximate SPNE in the original game M. In particular, we will show how the regret of any SPNE of M' when lifted to M is bounded in terms of the reward and transition abstraction error bounds of the state and action abstraction functions that map M to M'.

THEOREM 1. *For any subgame perfect Nash equilibrium (SPNE) strategy σ'^* in M', any corresponding lifted joint strategy $\sigma^{\uparrow\sigma'^*}$ in M has the property that*

$$\forall i, \forall s \in S_k, \forall \pi_i \in S \rightarrow A_i(S) :$$
$$V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(s) \leq V_i^{\sigma^{\uparrow\sigma'^*}}(s) + 2kf_{k,i} \quad (15)$$

where $\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle$ is the joint strategy in M that results from Agent i unilaterally deviating from $\sigma^{\uparrow\sigma'^*}$ to pure strategy π_i, and $f_{k,i}$ is as defined in Proposition 1.

PROOF. By induction on the height of the LDAG (as in the proof for Proposition 1). Note that unlike strategy $\sigma^{\uparrow\sigma'^*}$, the strategy $\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle$ is not implementable in the abstract game M'. We will also need the following definition of an adaptation of $\sigma^{\uparrow\sigma'^*}$ that is consistent with the action of π_i on state s and hence is constrained to take the same action in all states in $h(s)$ to be implementable in M', i.e., the only change from $\sigma^{\uparrow\sigma'^*}$ is that

$$\sigma_i^{\uparrow\sigma'^*_{\pi_i(s)|s}}(\pi_i(s)|x \in h(s)) = 1,$$

where the corresponding abstract strategy is denoted $\sigma'^*_{\pi_i(s)|s}$.

Base case (terminal nodes of the LDAG): We prove that the theorem holds for $s \in S_1$. Suppose pure strategy π_i is such that for some $s \in S_1$, $V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(s) > V_i^{\sigma^{\uparrow\sigma'^*}}(s) + 2f_{1,i}$. Define $\sigma'^*_{M,\pi_i(s)|s}$ to be the minimal adaptation of σ'^*_M as in the theorem statement. Now,

$$\begin{aligned} V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(s) &= \sum_{\mathbf{a}_{-i} \in A_{-i}} \sigma_{-i}^{\uparrow\sigma'^*}(\mathbf{a}_{-i}|s) R_i(s, \langle \pi_i(s), \mathbf{a}_{-i}\rangle) \\ &= V_i^{\sigma^{\uparrow\sigma'^*_{\pi_i(s)|s}}}(s). \end{aligned} \quad (16)$$

We get a contradiction by the following chain:

$$\begin{aligned} W_i^{\sigma'^*_{\pi_i(s)|s}}(h(s)) &\geq V_i^{\sigma^{\uparrow\sigma'^*_{\pi_i(s)|s}}}(s) - f_{1,i}; \text{ by Proposition 1} \\ &= V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(s) - f_{1,i}; \text{ by Equation 16} \\ &> (V_i^{\sigma^{\uparrow\sigma'^*}}(s) + 2f_{1,i}) - f_{1,i}; \text{ by supposition} \\ &\geq ((W_i^{\sigma'^*}(h(s)) - f_{1,i}) + 2f_{1,i}) - f_{1,i}; \text{ by Prop. 1} \\ &= W_i^{\sigma'^*}(h(s)) \end{aligned}$$

which is a contradiction because σ'^* is an SPNE in M'.

Step 2: We assume the theorem holds for $s \in S_{\leq k}$.

Step 3: We prove that the assumption from Step 2 implies that the theorem holds for $s \in S_{k+1}$.

Suppose π_i is such that for some $s \in S_{k+1}$, $V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(s) > V_i^{\sigma^{\uparrow\sigma'^*}}(s) + 2(k+1)f_{k+1,i}$. Now,

$$\begin{aligned} V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(s) &= \sum_{\mathbf{a}_{-i} \in A_{-i}} \sigma_{-i}^{\uparrow\sigma'^*}(\mathbf{a}_{-i}|s) \Big[R_i(s, \langle \pi_i(s), \mathbf{a}_{-i}\rangle) \\ &\quad + \sum_{x \in S_k} T(s, \langle \pi_i(s), \mathbf{a}_{-i}\rangle, x) V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(x) \Big] \\ &= \sum_{\mathbf{a}_{-i} \in A_{-i}} \sigma_{-i}^{\uparrow\sigma'^*}(\mathbf{a}_{-i}|s) R_i(s, \langle \sigma_i^{\uparrow\sigma'^*_{\pi_i(s)|s}}(s), \mathbf{a}_{-i}\rangle) \\ &\quad + \sum_{\mathbf{a}_{-i} \in A_{-i}} \sigma_{-i}^{\uparrow\sigma'^*}(\mathbf{a}_{-i}|s) \\ &\qquad \sum_{x \in S_k} T(s, \langle \pi_i(s), \mathbf{a}_{-i}\rangle, x) V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(x) \\ &\leq \sum_{\mathbf{a}_{-i} \in A_{-i}} \sigma_{-i}^{\uparrow\sigma'^*}(\mathbf{a}_{-i}|s) R_i(s, \langle \sigma_i^{\uparrow\sigma'^*_{\pi_i(s)|s}}(s), \mathbf{a}_{-i}\rangle) \\ &\quad + \sum_{\mathbf{a}_{-i} \in A_{-i}} \sigma_{-i}^{\uparrow\sigma'^*}(\mathbf{a}_{-i}|s) \sum_{x \in S_k} T(s, \langle \pi_i(s), \mathbf{a}_{-i}\rangle, x) \\ &\qquad \Big[V_i^{\sigma^{\uparrow\sigma'^*_{\pi_i(s)|s}}}(x) + 2kf_{k,i} \Big] \\ &= V_i^{\sigma^{\uparrow\sigma'^*_{\pi_i(s)|s}}}(s) + 2kf_{k,i} \\ &\leq W_i^{\sigma'^*_{\pi_i(s)|s}}(h(s)) + f_{k+1,i} + 2kf_{k,i}. \end{aligned} \quad (17)$$

We get a contradiction by the chain

$$\begin{aligned} W_i^{\sigma'^*_{\pi_i(s)|s}}(h(s)) &\geq V_i^{\langle \pi_i, \sigma_{-i}^{\uparrow\sigma'^*}\rangle}(s) - f_{k+1,i} - 2kf_{k,i}; \text{ by Eq. 17} \\ &> \Big[V_i^{\sigma^{\uparrow\sigma'^*}}(s) + 2(k+1)f_{k+1,i} \Big] - f_{k+1,i} - 2kf_{k,i} \\ &\quad ; \text{ by supposition} \\ &= V_i^{\sigma^{\uparrow\sigma'^*}}(s) + 2k(f_{k+1,i} - f_{k,i}) + f_{k+1,i} \\ &\geq \Big[W_i^{\sigma'^*}(h(s)) - f_{k+1,i} \Big] + 2k(f_{k+1,i} - f_{k,i}) \\ &\quad + f_{k+1,i}; \text{ by Prop. 1} \\ &= W_i^{\sigma'^*}(h(s)) + 2k(f_{k+1,i} - f_{k,i}) \end{aligned} \quad (18)$$

which is a contradiction because by construction in Proposition 1, $f_{k+1,i} \geq f_{k,i}$. □

The regret at height k in the regret bound of Theorem 1 depends cubically on k (there is a dependence on k in Equation 15 and a further multiplicative dependence on k^2 in the $f_{k,i}$ term as defined in the statement of Proposition 1).

5.1 Extension to infinite-horizon discounted stochastic games

In the single-agent Markov Decision Process (MDP) counterpart to stochastic games there is a long history (e.g., [17]) of using finite-horizon analyses as the basis for an analysis of the discounted infinite-horizon MDP. Here we very briefly sketch the same argument-structure applied to our stochastic game setting. Suppose we pick a finite horizon of L and find abstractions as specified in our paper so that the regret at the root node (where the regret is the largest) is ϵ. Now consider any extension of the finite horizon strategy profile to an infinite-horizon strategy profile, e.g., by always selecting the same arbitrary strategy profile at every time step after time L. What is the maximum additional regret at the root node? The additional regret is upper bounded by the fraction $\gamma^L \frac{R_{\max}}{1-\gamma}$, where $R_{\max}$ is the difference between the largest reward and smallest reward. Crucially, this additional regret term decreases exponentially with L and so by making L large enough we can make this additional regret as small as we want (in particular, we can apportion a fraction of the overall error guarantee we seek to the error due to the finite-horizon approximation). We leave the detailed derivation of this result to future work.

6. ABSTRACTION ALGORITHMS

Here we develop algorithms for automatically constructing an abstract game M' from the original game M with the guarantee that an SPNE in M' when lifted and implemented in M will be an approximate SPNE in M (cf. Theorem 1).

6.1 Bottom-up single-pass level-by-level greedy abstraction algorithm

We start by describing a greedy, bottom-up, single-pass algorithm. Given an overall approximation bound on the quality of the SPNE to be found by constructing M', there is a challenging and interesting question of how to apportion the approximation bound into the ϵ's needed for the different levels of the game (see Equations 3,4,6,7) and indeed more finely to the different state and action abstractions. For now, we assume that we have done this apportionment by working backwards from our theory in Section 4. One simple way to do this would be to assume that the ϵ^R's and ϵ^T's are the same for all levels and then solving for these two constants from Equation 15. Regardless of how we obtain them, we will present an algorithm which will take as input ϵ_k^R and ϵ_k^T for every level k separately.

Our algorithm for constructing M' proceeds bottom-up, layer by layer as shown in Algorithm 3. For each layer, first the action-abstractions (g's) are constructed greedily and then the state-abstractions (h's) are constructed greedily. This ordering of actions first and states second was a choice and one could get a similar algorithm by doing this in the reverse order. The size of M' would in general depend on this choice and which choice is better would depend on the details of M.

Action Abstractions.

ALGORITHM 1: Computing Action Abstractions

Input: $s \in S_k, \epsilon, \epsilon', S'_{k-1}, h_{k-1}$

1 **for** $i = 1 \ldots n$ **do**
2 **forall the** $\mathbf{a}_i \in A_i(s)$ **do**
3 $g_{s,i}(\mathbf{a}_i) \leftarrow \mathbf{a}_i$ // default assumption that $\mathbf{a}_i$ will start a new bucket
4 reward-check = false, transition-check = false
5 // next compare $\mathbf{a}_i$ with all actions that come before it in $A_i(s)$ under the ordering of line 2
6 **forall the** $\mathbf{b} \in A_i(s)$ *before* $\mathbf{a}_i$ **do**
7 // check if $\mathbf{a}_i$ is similar in reward to the action that $\mathbf{b}$ maps to (cf. Equation 3)
8 **if** $\max_{\mathbf{a}_{-i} \in A_{-i}(s)} |R_i(s, \langle \mathbf{a}_i, \mathbf{a}_{-i} \rangle) - R_i(s, \langle g_{s,i}(\mathbf{b}), \mathbf{a}_{-i} \rangle)| \leq \epsilon$ **then**
9 reward-check = true
10 **end**
11 // check if $\mathbf{a}_i$ is similar in transitions to the action that $\mathbf{b}$ maps to (cf. Equation 7)
12 **if** $\Big(k == 1\Big)$ *OR* $\Big(\sum_{x' \in S'_{k-1}} \Big\{ \max_{\mathbf{a}_{-i} \in A_{-i}(s)} | \sum_{x \in h_{k-1}^{-1}(x')} T(s, \langle \mathbf{a}_i, \mathbf{a}_{-i} \rangle, x) - \sum_{x \in h_{k-1}^{-1}(x')} T(s, \langle g_{s,i}(\mathbf{b}), \mathbf{a}_{-i} \rangle, x)| \Big\} \leq \epsilon' \Big)$ **then**
13 transition-check = true
14 **end**
15 **if** *(reward-check == true AND transition-check == true)* **then**
16 $g_{s,i}(\mathbf{a}_i) \leftarrow g_{s,i}(\mathbf{b})$ // map a_i to whatever $\mathbf{b}$ maps to
17 break
18 **else**
19 reward-check = false
20 transition-check = false
21 **end**
22 **end**
23 **end**
24 **end**

Output: $g_{s,1}, \ldots, g_{s,n}$

Algorithm 1 shows the procedure for obtaining action abstractions for any specific state given the state abstractions at the level below. It takes a state s as input, cycles through each agent (line 1), for each agent considers every action in $A(s)$ (line 2), and then buckets the actions with a prototype-action for each bucket. Buckets get assigned by checking both that the rewards are similar (lines 8-10) and that transitions are similar (lines 12-14), where similarity is defined by the ϵ and ϵ' input parameters respectively. The function $g_{s,i}(a_i \in A_i(s))$ maps action a_i to the prototype-action for its bucket. Algorithm 1 is called separately for every state (see lines 3-5 of Algorithm 3).

State Abstractions.

Algorithm 2 shows the procedure for obtaining state abstractions for any level given the action abstractions already obtained at that level as well as the state abstractions at the level below. It cycles through every state s at that level (line 1) and compares it to every state x it has already mapped to a bucket before (line 4). The comparisons involve finding a one-to-one correspondence (line 9) if one exists between the abstract actions of s and the abstract actions in the prototype state corresponding to x that satisfies reward similarity (line 10) and satisfies transition similarity (lines 11-12),

ALGORITHM 2: Computing State Abstractions

Input: $\epsilon, \epsilon', g, S'_{k-1}, h_{k-1}$

1 **forall the** $s \in S_k$ **do**
2 $\quad h_k(s) = s$ // default assumption that s will start a new bucket
3 $\quad$ // next compare s with all states that come before it in S_k under the ordering of line 1
4 $\quad$ **forall the** $x \in S_k$ *before* s **do**
5 $\quad\quad$ // First check if the same number of abstract actions are available for each agent in s and $h_k(x)$
6 $\quad\quad$ **if** $\forall i, |range(g_{s,i}(A_i(s)))| == |range(g_{h_k(x),i}(A_i(h_k(x))))|$ **then**
7 $\quad\quad\quad$ // Consider all matchings between the abstract actions of s and $h_k(x)$
8 $\quad\quad\quad$ $\mathcal{Z} \leftarrow$ all one-to-one correspondences between $range(g_s(A(s)))$ and $range(g_{h_k(x)}(A(h_k(x)))$
9 $\quad\quad\quad$ **if** $\exists Z \in \mathcal{Z}$ *such that,*
10 $\quad\quad\quad\quad$ $\Big(\Big[\max_{\mathbf{a}\in A(s)} \max_i |R_i(s,\mathbf{a}) - R_i(h_k(x), Z(g_s(\mathbf{a})))| \leq \epsilon\Big]$ *// reward-check*
11 $\quad\quad\quad\quad$ *AND* $\Big[(k == 1)$ *OR* $\Big(\sum_{x' \in S'_{k-1}} \Big\{ \max_{\mathbf{a}\in A(s)} |\sum_{y \in h^{-1}_{k-1}(x')} T(s,\mathbf{a},y) -$
12 $\quad\quad\quad\quad\quad\quad \sum_{y \in h^{-1}_{k-1}(x')} T(h_k(x), Z(g_s(\mathbf{a})), y)| \Big\} \leq \epsilon'\Big)\Big]\Big)$ *// transition-check*
13 $\quad\quad\quad$ **then**
14 $\quad\quad\quad\quad$ $h_k(s) \leftarrow h_k(x)$
15 $\quad\quad\quad\quad$ break
16 $\quad\quad\quad$ **end**
17 $\quad\quad$ **end**
18 $\quad$ **end**
19 **end**

Output: h_k

ALGORITHM 3: Computing Abstract Game

Input: $\epsilon^R_1, \ldots, \epsilon^R_n, \epsilon^T_1, \ldots, \epsilon^T_n$

1 **for** $k = 1$ **to** *horizon* **do**
2 $\quad$ // compute action abstractions
3 $\quad$ **forall the** $s \in S_k$ **do**
4 $\quad\quad$ $g_s \leftarrow$ ALGORITHM 1($s, \frac{\epsilon^R_k}{2}, \frac{\epsilon^T_k}{2}, S'_{k-1}, h_{k-1}$)
5 $\quad$ **end**
6 $\quad$ // compute state abstractions
7 $\quad$ $h_k \leftarrow$ ALGORITHM 2($\frac{\epsilon^R_k}{2}, g, \frac{\epsilon^T_k}{2}, S'_{k-1}, h_{k-1}$)
8 $\quad$ $S'_k \leftarrow$ range($h_k(S_k)$)
9 **end**

Output: g, h

where similarity is defined by the ϵ and ϵ' input parameters respectively. Such a one-to-one correspondence can be found via a bipartite-matching algorithm and if such a matching exists s is mapped (line 14) to the prototype state $h_k(x)$. Consequently, Algorithm 2 buckets similar states with the h function mapping each state to a prototype-state for its bucket and is called once for each level (see line 7 of Algorithm 3).

Constructing M'.

The bottom-up single-pass Algorithm 3 first calls the action abstraction algorithm (lines 3-5) with approximation parameters $\frac{\epsilon^R_k}{2}$ and $\frac{\epsilon^T_k}{2}$, where the fractions are used to allow for approximation on both sides of the choice of the prototypes in each bucket. Then it calls the state abstraction algorithm (line 7) again with fractional approximation parameters for the same reason as for the action abstraction algorithm. It returns the action-abstraction function g as well as the state-abstraction function h. From these the transition probabilities and the reward functions of M' can be defined by using the transition probabilities and the reward functions in M for the prototype states and actions defined in the buckets in h and g.

6.1.1 Comments on the greedy abstraction algorithm

Algorithm 3 is correct by construction; it yields an M' that satisfies the transition and reward approximation bounds specified as inputs to the algorithm. Recall that these reward and transition bounds were chosen to guarantee the upper-bound on the approximation quality of the SPNE found in M' when implemented in M.

If ϵ^R_k and ϵ^T_k is set equal to zero for all levels k, any abstractions found by Algorithm 3 would be provably lossless.

If only action abstraction were desired, this can be obtained easily by simply not calling the state abstraction algorithm (i.e., by deleting line 7 and setting the h function to the identity function). If, on the other hand, only state abstraction were desired, this can be obtained just as easily by not calling the action abstraction algorithm (i.e., by deleting lines 3 to 5 and setting the g function to the identity function).

Finally, we expect the actual regret in M from using the lifted strategies corresponding to the SPNE strategies found in M' would typically be far smaller than that implied by the theoretical worst-case upper bounds. We plan to evaluate this empirically in future work.

6.2 Bottom-up single-pass level-by-level integer linear program (ILP) for abstraction

One can ask whether it is possible to write a mathematical program to optimize over entire abstractions. Unfortunately, the constraints in the program would not be linear. (Problems where the constraints and objective are linear tend to be dramatically faster to solve.) For example, the error at level 2 would involve a product of reward error in level 1 and transition probability error in level 2. In general, with a k-level LDAG, the constraints in the mathematical program would be kth-order polynomials.

Therefore, in this section we develop an integer linear program (ILP) that optimizes the action and state abstraction within a level, given the abstraction at levels below, and given an error bound allowed for this level. Note that this approach might not yield the best (coarsest in some sense) abstraction, for example, because we do not optimize all levels simultaneously and because we are assuming that the allowable error for each level is given. Nevertheless, as we will see, this approach avoids the need to exogenously make many of the decision choices used in the greedy algorithm of the previous section. The ILP is used the same way as the greedy algorithm: bottom level first, then the level above it, and so on.

This ILP chooses prototypical actions (one for each action bucket for each state and agent in the abstraction) and prototypical states (one for each state bucket in the abstraction)—rather than generating new actions or new

states to be the prototypes.

By construction, the ILP will generate an abstraction that honors the bounds.

We will present the ILP for optimizing an interior level k. The bottom level is simpler because there are no transitions left; we will not present the ILP for that.

The following constraint allows the ILP to split the overall allowed error for this level, ϵ_k (which is given as an input to the ILP) between reward error and transition error. (The greedy algorithm of the previous section requires both of these numbers as input. In other words, it does not handle this splitting.)

$$\epsilon_k^R + \epsilon_k^T \overline{W_k} \leq \epsilon_k$$

Here, $\overline{W_k} = \max_{\sigma'} \overline{W}_k^{\sigma'}$. (Note that this is a conservative multiplier; by using a less conservative—i.e., less aggregated—multiplier, it might be possible to get coarser abstractions while still satisfying the overall error bound.)

The following constraints allow the ILP to split the error between action abstraction and state abstraction. (In contrast, the greedy algorithm of the previous section hardwires these splits 50-50.)

$$\epsilon_k^{state,R} + \epsilon_k^{action,R} \leq \epsilon_k^R \;\; and \;\; \epsilon_k^{state,T} + \epsilon_k^{action,T} \leq \epsilon_k^T$$

Now, we define the following helpful auxiliary variables:

$$\begin{aligned} \alpha_s &= 1 \text{ if state } s \text{ is selected as a state prototype,} \\ &\quad \text{and 0 otherwise.} \\ \beta_{s,a,i} &= 1 \text{ if action } a \text{ is selected as an action prototype} \\ &\quad \text{for agent } i \text{ in state } s, \text{ and 0 otherwise.} \end{aligned}$$

The objective in the ILP can be any linear function of the variables. Our goal is to generate a coarse abstraction, so the equilibrium finding in the abstracted game would be scalable. However, what kind of abstraction is good for scalability may depend on the problem (e.g., whether the game is zero-sum or not) and on the equilibrium-finding algorithm. One family of proxies for this is the size of the resulting abstraction as measured by some linear combination of the number of states and actions in the abstracted game:

$$\min \sum_s \alpha_s + \rho \sum_{s,a,i} \beta_{s,a,i}, \text{ for any constant } \rho$$

(Note that we could also have different ρ's for different agents at different states. Also, we could use a different ρ when running the ILP at different levels of the LDAG.)

Next, we tie the elements to the prototypes using additional auxiliary variables:

$$\begin{aligned} H_{s,t} &= 1 \text{ if state } s \text{ maps to prototype } t, \text{ and 0 otherwise.} \\ G_{a,b,s,i} &= 1 \text{ if action } a \text{ maps to action prototype } b \\ &\quad \text{in state } s \text{ for agent } i, \text{ and 0 otherwise.} \end{aligned}$$

We do the tying in the ILP using the following constraints:

$$\forall s, \; \textstyle\sum_t H_{s,t} = 1$$

// Each state maps to exactly one prototype state (possibly itself).

$$\forall s, i, a \in A_i(s), \textstyle\sum_b G_{a,b,s,i} = 1$$

// Each action maps to exactly one prototype action.

Next, we tie the above mappings to the α's and β's:

$$\forall t, \; \textstyle\sum_s H_{s,t} \leq \alpha_t$$

// Only selected state prototypes can be mapped to.

$$\forall s, i, b \in A_i(s), \; \textstyle\sum_a G_{a,b,s,i} \leq \beta_{s,b,i}$$

// Only selected action prototypes can be mapped to.

Then, we have to make sure that in the abstracted game, all actions within an information set have the same actions. In other words, states can be bundled only if they have the same abstracted action set:

$$\forall s, t, \; \big(H_{s,t} = 1 \Rightarrow \forall i, a \in A_i(s), b \in A_i(s), \; G_{a,b,s,i} = G_{a,b,t,i}\big).$$

This can be written in the ILP, for example, using the following two sets of constraints:

$$\begin{aligned} \forall s, t, i, a \in A_i(s), b \in A_i(s), \; G_{a,b,s,i} &\leq G_{a,b,t,i} + (1 - H_{s,t}) \\ \forall s, t, i, a \in A_i(s), b \in A_i(s), \; G_{a,b,t,i} &\leq G_{a,b,s,i} + (1 - H_{s,t}). \end{aligned}$$

Like in the greedy algorithm in the previous section, the action abstraction reward checks are

$$\begin{aligned} &\forall s, i, a \in A_i(s), b \in A_i(s), \\ &G_{a,b,s,i} = 1 \quad \Rightarrow \\ &\max_{\mathbf{a}_{-i} \in A_{-i}(s)} |R_i(s, \langle a, \mathbf{a}_{-i} \rangle) - R_i(s, \langle b, \mathbf{a}_{-i} \rangle)| \leq \epsilon_k^{action,R}. \end{aligned}$$

(Note that the only variable on the right hand side of the implication is $\epsilon_k^{action,R}$.) These conditions can be written as ILP constraints, for example, as follows:

$$\begin{aligned} &\forall s, i, a \in A_i(s), b \in A_i(s), \\ &\max_{\mathbf{a}_{-i} \in A_{-i}(s)} |R_i(s, \langle a, \mathbf{a}_{-i} \rangle) - R_i(s, \langle b, \mathbf{a}_{-i} \rangle)| \\ &\leq \epsilon_k^{action,R} + M(1 - G_{a,b,s,i}), \end{aligned}$$

where M is a large number (as is standard in integer programming).

Similarly, the action abstraction transition error checks can be written as ILP constraints as follows:

$$\begin{aligned} &\forall s, i, a \in A_i(s), b \in A_i(s), \\ &\sum_{x' \in S'_{k-1}} \Big\{ \max_{\mathbf{a}_{-i} \in A_{-i}(s)} \Big| \sum_{s' \in h_{k-1}^{-1}(x')} T(s, \langle a, \mathbf{a}_{-i} \rangle, s') \\ &- \sum_{s' \in h_{k-1}^{-1}(x')} T(s, \langle b, \mathbf{a}_{-i} \rangle, s') \Big| \Big\} \\ &\leq \epsilon_k^{action,T} + M(1 - G_{a,b,s,i}). \end{aligned}$$

In principle, the state abstraction reward error checks are as follows, for each state s and the prototype state t of the state bucket that s belongs to:

$$\max_{\mathbf{a} \in A(s)} \max_i |R_i(s, \mathbf{a}) - R_i(t, g_s(\mathbf{a}))| \leq \epsilon_k^{state,R}.$$

However, we have to be careful because the action abstraction function g is not a constant now, but endogenous to the ILP. (The helpful side of this is that—unlike in the greedy state abstraction algorithm in the previous section (ALGORITHM 2)—we do not have to have a Z-function that studies the one-to-one correspondences between action in s and t for agent i: the ILP will automatically generate a g where the actions match.) We can write these checks as ILP constraints as follows. The idea is that we write the constraints

down for all possible abstractions, and then relax them (using the big-M's) for the abstractions that the ILP does not choose:

$$\forall s, t, i, \mathbf{a} \in \mathbf{A}(s), \mathbf{b} \in \mathbf{A}(s),$$
$$|R_i(s, \mathbf{a}) - R_i(t, \mathbf{b})|$$
$$\leq \epsilon_k^{state,R} + M(1 - H_{s,t}) + \sum_j M(1 - G_{\mathbf{a}_j, \mathbf{b}_j, s, j}).$$

Finally, the state abstraction transition error checks are conceptually as follows. They check whether the transitions are similar for s and the prototype state t of the state bucket that s belongs to:

$$\sum_{x' \in S'_{k-1}} \Big\{ \max_{\mathbf{a} \in A(s)} | \sum_{s' \in h_{k-1}^{-1}(x')} T(s, \mathbf{a}, s') - \sum_{s' \in h_{k-1}^{-1}(x')} T(t, g_s(\mathbf{a}), s')| \Big\} \leq \epsilon_k^{state,T}.$$

Now, again, g is endogenous to the ILP. We can write the above conditions as ILP constraints as follows:

$$\forall s, t, \mathbf{a} \in \mathbf{A}(s), \mathbf{b} \in \mathbf{A}(s),$$
$$\sum_{x' \in S'_{k-1}} \Big\{ \max_{\mathbf{a} \in A(s)} | \sum_{s' \in h_{k-1}^{-1}(x')} T(s, \mathbf{a}, s') - \sum_{s' \in h_{k-1}^{-1}(x')} T(t, \mathbf{b}, s')| \Big\}$$
$$\leq \epsilon_k^{state,T} + M(1 - H_{s,t}) + \sum_j M(1 - G_{\mathbf{a}_j, \mathbf{b}_j, s, j}).$$

This completes the construction of the ILP. Note that the ILP handles action and state abstraction simultaneously. This is in contrast to the greedy algorithm of the previous section that does one of these first without looking at the other, and then does the other.

6.3 Discussion of the abstraction algorithms and problem complexity

Both the greedy algorithm and the ILP satisfy the accuracy bounds by construction. For any constant number of agents, the ILP is of polynomial size. However, the within-level abstraction problem that both algorithms are trying to solve is hard in the worst case:

PROPOSITION 2. *The (decision version of the) within-level abstraction optimization problem that the ILP and the greedy algorithm are trying to solve is NP-complete. This is the case even if we only want to do action abstraction (i.e., no state abstraction), even if there are only two agents and one state in the game, and the game is only played for one step.*

PROOF. It is easy to check the solution, so the problem is in NP. What remains to be shown is that it is NP-hard. We reduce from SET COVER, which is NP-complete. There we are given a set $\mathcal{S} = (S_1, \ldots, S_r)$ and a ground set of elements $E = \{1, \ldots, q\}$. Each set S_i contains some subset of E. The question is whether all the items in E can be covered using only k sets from $\mathcal{S}$.

Now, we can solve this problem by considering the abstraction problem of a 2-agent game where there is only one state and one step. The two agents are an attacker and a defender. The attacker's action set is $E = \{1, \ldots, q\}$. The defender's action set is $\mathcal{S} = (S_1, \ldots, S_r)$. The payoffs in this game are such that the attacker gets M if he attacks a location that the defender does not defend, and $-M$ otherwise (here, M is very large). The defender gets 1 if he defends successfully and -1 otherwise.

We set the target accuracy in the abstraction algorithm so that the defender cannot lose for sure. (This is possible since the abstraction algorithm will try to remove the defender's actions first before any of the attacker's actions because the attacker's payoffs dominate.) We ask the abstraction algorithm whether such an abstraction exists with k actions for the defender. This is exactly SET COVER. □

The problem being hard in the worst case does not mean that the ILP will be slow in practice. In fact, it has been shown experimentally that within-level abstraction optimization ILPs run fast (in less than a second) even in the large (in a different type of game state abstraction problem) [9]. How fast the ILP will run in practice will depend on the game.

Running a polynomial-time abstraction algorithm (such as our greedy algorithm), or an abstraction algorithm that runs fast in practice though not being polynomial in the worst case, can be very helpful as a preprocessor. These algorithms run in polynomial time in the entire stochastic game, while the (Nash) equilibrium-finding algorithm can take exponential time just to solve one stage game—because solving a normal form game is PPAD-complete (even with two agents and all payoffs in $\{0, 1\}$) [5, 4, 1].

7. CONCLUSIONS AND FUTURE RESEARCH

Abstraction followed by equilibrium finding has emerged as the leading approach to solving games. Lossless abstraction typically yields games that are still too large to solve, so lossy abstraction is needed. Unfortunately, prior lossy game abstraction algorithms have no guarantees on solution quality.

In this paper we developed a framework that enables the design of lossy game abstraction algorithms with guarantees on solution quality. It simultaneously handles state and action abstraction. It is based on bounding the errors in payoffs as well as errors in transition probabilities, caused by abstraction. Specifically, we defined a measure of reward approximation error and transition probability error achieved by state and action abstraction in stochastic games such that the regret of the equilibrium found in the abstract game when implemented in the original, unabstracted game is upper-bounded by a function of those measures.

We then developed the first lossy game abstraction algorithms with bounds on solution quality. Both of them work level-by-level up from the end of the game. One of the algorithms is greedy and the other is an integer linear program. We also proved that the abstraction problem is NP-complete (even with just action abstraction, 2 agents, and a 1-step game), but want to point out that this does not mean that the game abstraction problems that occur in practice cannot be solved quickly.

The obvious stream of applications of this work is to develop better (faster and/or higher-accuracy) game-solving algorithms by having better automated abstraction algorithms. However, the work has an additional important potential application stream in modeling. All models are abstractions of reality. As one models a real-world situation

formally as a game, one faces modeling choices. What aspects of the real world situation should be captured in the model? How detailed should one make the signal space? For example, how should one discretize state if the game solver requires discrete states? How should one discretize the action space when there are too many (and possibly infinitely many) actions in reality but one wants to use a scalable game solver that assumes a finite action space? Such modeling questions are really questions about abstraction, and the work in this paper will help pave the way to answering those questions in a rigorous way, whether the modeling is done by human or by machine.

There is ample scope for future research. Both of our algorithms obtain action and state abstraction by removing some actions and states and keeping others. Our theoretical results do not assume this form of abstraction and other more general forms of abstraction (with automated state and action generation) may yield smaller abstract games. We also believe that our results will extend in a straightforward way to sequential games, at least as long as the information sets form a layered directed acyclic graph. This means that when it is an agent's turn to move, the agent knows how many moves have been made so far (but not necessarily what agents have been given turns to move nor how they moved). This condition is satisfied in many common games. However, it would be interesting to see whether our techniques could be extended to games where that condition does not hold.

References

[1] Abbott, T., Kane, D., and Valiant, P. 2005. On the complexity of two-player win-lose games. In *FOCS*.

[2] Archibald, C. and Shoham, Y. 2009. Modeling billiards games. In *AAMAS*.

[3] Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., and Szafron, D. 2003. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI*.

[4] Chen, X., Deng, X., and Teng, S.-H. 2009. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM 56,* 3.

[5] Daskalakis, C., Goldberg, P., and Papadimitriou, C. 2009. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing 39,* 1, 195–259.

[6] Ganzfried, S., Sandholm, T., and Waugh, K. 2012. Strategy purification and thresholding: Effective non-equilibrium approaches for playing large games. *AAMAS*.

[7] Gilpin, A. and Sandholm, T. 2006a. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *AAAI*.

[8] Gilpin, A. and Sandholm, T. 2006b. A Texas Hold'em poker player based on automated abstraction and real-time equilibrium computation. In *AAMAS*. Demo track.

[9] Gilpin, A. and Sandholm, T. 2007a. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *AAMAS*.

[10] Gilpin, A. and Sandholm, T. 2007b. Lossless abstraction of imperfect information games. *JACM 54,* 5.

[11] Gilpin, A. and Sandholm, T. 2008. Expectation-based versus potential-aware automated abstraction in imperfect information games: An experimental comparison using poker. In *AAAI*. Short paper.

[12] Gilpin, A., Sandholm, T., and Sørensen, T. B. 2007. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *AAAI*.

[13] Gilpin, A., Sandholm, T., and Sørensen, T. B. 2008. A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. In *AAMAS*.

[14] Givan, R., Dean, T., and Greig, M. 2003. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence 147,* 1-2, 163–223.

[15] Hawkin, J., Holte, R., and Szafron, D. 2011. Automated action abstraction of imperfect information extensive-form games. In *AAAI*.

[16] Jiang, A. X. and Leyton-Brown, K. 2011. Polynomial-time computation of exact correlated equilibrium in compact games. In *EC*.

[17] Kearns, M., Mansour, Y., and Ng, A. 1999. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. In *IJCAI*.

[18] Lipton, R., Markakis, E., and Mehta, A. 2003. Playing large games using simple strategies. In *EC*.

[19] Littman, M. and Stone, P. 2003. A polynomial-time Nash equilibrium algorithm for repeated games. In *EC*.

[20] Ravindran, B. 2004. An algebraic approach to abstraction in reinforcement learning. Ph.D. thesis, UMass.

[21] Schnizlein, D., Bowling, M., and Szafron, D. 2009. Probabilistic state translation in extensive games with large action sets. In *IJCAI*.

[22] Shi, J. and Littman, M. 2002. Abstraction methods for game theoretic poker. *Revised Papers from the 2nd Internat. Conference on Computers and Games*. Springer.

[23] Sorg, J. and Singh, S. 2009. Transfer via soft homomorphisms. In *AAMAS*. 741–748.

[24] Waugh, K., Schnizlein, D., Bowling, M., and Szafron, D. 2009a. Abstraction pathologies in extensive games. In *AAMAS*.

[25] Waugh, K., Zinkevich, M., Johanson, M., Kan, M., Schnizlein, D., and Bowling, M. 2009b. A practical use of imperfect recall. In *SARA*.

[26] Wellman, M. 2006. Methods for empirical game-theoretic analysis (extended abstract). In *AAAI*.

[27] Zinkevich, M., Bowling, M., Johanson, M., and Piccione, C. 2007. Regret minimization in games with incomplete information. In *NIPS*.