Abstraction Using Analysis of Subgames

Anjon Basak¹ and Christopher Kiekintveld²

¹ The University of Texas at El Paso
   abasak@miners.utep.edu,
² The University of Texas at El Paso
   cdkiekintveld@utep.edu

Abstract. Normal form games are one of the most familiar representations for modeling interactions among multiple agent. However, modeling many realistic interactions between agents results in games that are extremely large. In these cases computing standard solutions like Nash equilibrium may be intractable. To overcome this issue the idea of abstraction has been investigated, most prominently in research on computer Poker. Solving a game using abstraction requires using some method to simplify the game before it is analyzed. We study a new variation for solving normal form games using abstraction that is based on finding and solving suitable sub games. We compare this method with several variations of a common type of abstraction based on clustering similar strategies.

Keywords: Game theory, abstraction, normal form games, clustering, Nash equilibrium approximation

1 Introduction

Game theory models interactions among self-interested rational agents, where each agent tries to maximize its utility. The most basic representation of a game is the well-known normal form (NFG), which enumerates all possible strategies for each player, and lists the payoffs for the different combinations of strategy choices in a set of matrices. Richer representations such as extensive-form games and Bayesian games can be translated into normal form, though there if often a dramatic increase in the size of the representation. Many different solution concepts have been proposed for analyzing NFG, including Nash equilibrium [16].

One of the key challenges in computational game theory is finding effective ways of analyzing extremely large games. Many real-world situations where we would like to apply game theory naturally have very large strategy spaces or other complexities. However, finding Nash equilibria is known to be a computationally hard problem [4], and most existing algorithms (such as those implemented in the Gambit software package [15]) do not scale well in practice.

To analyze games that are beyond the limits of standard solution algorithms, an increasingly common approach is to apply some form of automated abstraction to simplify the game. The simplified game is then analyzed using an available solver, and the solution is somehow mapped back into the original game. If the simplified game is able to retain the key strategic features of the original game, then in principle the solution
of the simpler game may be a reasonable approximation of the solution to the original game. This general approach has been very successful in developing computer poker agents, and most of the successful players in the annual competition in computer poker over the past years have used some variation of this idea (e.g., [7, 8, 10, 11, 19]).

Many of the recent works on abstraction focus on extensive form games with sequential interactions and uncertainty, in part due to the motivation of poker agents. These include a wide variety of specific methods including both lossless abstractions [9] and lossy abstractions [17]. While many lossy methods do not provide bounds on the error introduced by the abstraction, some recent work has been able to provide theoretical bounds for very general classes of games [13]. Another recent example considers imperfect recall abstractions with earth mover’s distance [6] for a hierarchical abstraction [2] technique.

In this work we focus on the simpler setting of normal form games. Even for this setting abstraction is not completely understood, and there has been less work in this area (with some exceptions, such as the decision-theoretic clustering methods proposed by Bard et al. [1]). We consider several variations of abstractions based on clustering different strategies and somehow merging these strategies to create a simpler game. The idea of clustering strategies (or information sets in games with imperfect information) is central to many of the existing abstraction methods.

The first group of abstractions we consider is based on the common approach of clustering strategies that are similar based on the payoffs they give in response to all possible opponent strategies. We consider different versions of this approach depending on how the abstracted game is analyzed. In particular we introduce an approach that uses additional information from the abstraction process to more directly calculate solutions that are good approximate equilibria in the original game, rather than just finding an equilibrium of the abstracted game. The other methods we consider are motivated by the recursive game reduction technique introduced by Conitzer et al. [3]. This idea leads to a different type of clustering abstraction where clusters form subgames that are solved separately to create the abstracted game. We also introduce a variation of this technique that can improve solution quality by iteratively modifying the subgames to account for abstraction error. We present initial experimental results comparing these different types of clustering abstractions.

### 2 Background

A normal form game captures all possible combinations of strategies for the players in a matrix form. A player can choose either a single “pure” strategy or play a “mixed” strategy that specifies a probability distribution over the pure strategies. The goal for all players is to maximize their expected utility. Formally, a finite, $N$-person normal-form game is described by a tuple $(N, A, u)$, where [18]:

- $N$ is a finite set of $N$ players, indexed by $i$.
- $A = A_1 \times \ldots \times A_n$, where $A_i$ is a finite set of actions available to player $i$. Each vector $a = (a_1, \ldots, a_n) \in A$ is called an action profile. So, $a_i \in A_i$ is pure action in the original game. $a_{i,k}$ is the $k$th action for player $i$. 

– $s_i \in S_i$ is the space of mixed strategies in the game.
– $u = (u_1, \ldots, u_n)$ where $u_i : A \mapsto R$ is a real-valued utility (or payoff) function for player $i$, which is extended to mixed strategies as usual by using expected utility.
– $\pi^i(a_i)$ gives the probability of action $a_i$ for player $i$.
– $A_i(O)$ gives the set of available actions for player $i$ in NFG $O$.

Throughout this paper we will consider abstractions that are also represented as (simpler) normal-form games. We will use the following modified notation to refer to an abstracted game $(\hat{N}, \hat{A}, \hat{u})$:

– $\hat{A} = \hat{A}_1 \times \ldots \times \hat{A}_n$, where $\hat{A}_i$ is a finite set of actions available to player $i$, with action profiles $\hat{a} = (\hat{a}_1, \ldots, \hat{a}_n) \in \hat{A}$. We will use $\hat{a}_{i,k}$ for the $k$th action for player $i$.
– $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_n)$ where $\hat{u}_i : \hat{A} \mapsto \hat{R}$ is a real-valued utility (or payoff) function for player $i$.
– $\hat{a}_i \in \hat{A}_i$ is a pure action in the abstracted game.
– $\hat{s}_i \in \hat{S}_i$ is a mixed strategy in the abstracted game.
– $\hat{A}_i(R)$ is the set of available actions for player $i$.
– $\pi^i(\hat{a}_i)$ gives the probability of action $\hat{a}_i$ for player $i$.

3 Clustering Abstractions

Before giving a formal description of how clustering works, we present a high-level overview of how abstraction can be used in solving a NFG. First, a large NFG is reduced to a smaller one using an abstraction technique. Then we solve the abstracted game using some solution algorithm. Finally we use reverse mapping (we will give a formal definition soon) to map back the NE strategy back to the original game. The quality of the solution is evaluated in the original game based on how close it is to being a Nash equilibrium. We now describe each stage in more detail.

3.1 Clustering of Actions

The basic idea of a clustering abstraction is to find groups of similar actions and to merge them to generate a smaller game. There are many different ways to perform clustering, including different algorithms and different ways to define similarity. We use one
of the most common clustering methods, k-means clustering [14, 5] with a Euclidian distance metric on the payoff vectors for each strategy. The main steps of the method are outlined below; the overall procedure is run many times using random restarts and the best clustering is selected based on the criteria of minimizing the maximum distance between any strategy and the mean of the cluster it is assigned to.

1. For a player each of the k clusters are initialized with a random mean vector. The length of the vector is the number of different payoff values for each strategy against every possible opponent strategy profile (including the payoffs for the opponents).
2. The distances are calculated between each cluster mean and each pure strategy using Euclidian distance. Each action is assigned to the cluster with the closest mean.
3. Cluster means are re-calculated based on the strategies that are currently assigned to the cluster.
4. Repeat Step 2 and Step 3 until the mean values converge.

<table>
<thead>
<tr>
<th>cluster</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 4</td>
</tr>
<tr>
<td>2</td>
<td>1, 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cluster</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3, 4</td>
</tr>
<tr>
<td>2</td>
<td>1, 2</td>
</tr>
</tbody>
</table>

Table 2. Clusters for player 1 and player 2

For example, if we apply k-means clustering to game shown in Table 1 we might find two clusters of strategies for each player, as shown in Table 2. Some additional notations for clusters that we will use are:

- The set of clusters for player $i$ $c_i = \{c_{i,1}, ..., c_{i,m}\}$, where $c_{i,m}$ is the $m$th cluster for player $i$, and $c_{i,m} = \{a_{i,1}, ..., a_{i,k}\}$.
- Every action belongs to exactly one cluster, so $c_{i,1} \cap c_{i,2} \cap ... \cap c_{i,k} = \emptyset$

### 3.2 Specifying the Abstracted Game

We now present a way of building an abstracted game for a given clustering of the strategies (note that this is independent of how the clustering was calculated). Each cluster maps to a single pure strategy in the abstracted game. The outcomes in the abstracted game correspond to each player choosing to play one of the clusters of strategies in the original game. We calculate the payoffs for these outcomes by averaging the payoffs over all of the strategies in the clusters for each player. This is consistent with assuming that both players play a uniform random mixture of the strategies from these clusters in the original game. More formally:

---

3 We experimented with a number of different distance metrics and clustering methods, but none of them gave significantly better results than this one.
Abstraction Using Analysis of Subgames 5

\[ \hat{u}_i(c_{i,m}, c_{j,n}) = \sum_{\forall a_{i,k} \in g(c_{i,m})} \sum_{\forall a_{j,l} \in g(c_{j,n})} u_i(a_{i,k}, a_{j,l}) \quad \forall (i, j) \in N, i \neq j \]  

In equation 1 \( \hat{a}_i = c_{i,m}, \hat{a}_i \in \hat{A}_i(R) \) and \( \hat{a}_j = c_{j,n}, \hat{a}_j \in \hat{A}_j(R) \). \( g \) is the reverse mapping function (section 3.4). An example of an abstracted game calculated from Table 1 using this methods is given in Table 3.

<table>
<thead>
<tr>
<th>Player 1</th>
<th>Player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8, 6</td>
<td>4, 7.5</td>
</tr>
<tr>
<td>5.5, 6</td>
<td>4, 4.5</td>
</tr>
</tbody>
</table>

Table 3. Abstracted game

3.3 Solving The Abstracted Game

Once the abstracted game is constructed, we must choose a solution concept to analyze this simplified game. A natural choice that is often used is to search for a (possibly approximate) Nash equilibrium of the abstracted game. We consider both pure and mixed-strategy Nash equilibrium, as well as a different concept that directly minimizes a bound on the approximation quality in the original game.

**Approximate Pure Strategy Nash Equilibrium** In a pure-strategy Nash equilibrium (PSNE) all players play pure strategies that are mutual best-responses, so no player can gain by changing to a different pure strategy. It is simple to check whether a given outcome is a PSNE by simply checking the possible gains from deviation. However, PSNE are not guaranteed to exist. Therefore, we instead look for the pure-strategy outcome the is the best approximate equilibrium, with the minimum possible gain for any player to change to a different strategy. This is known as an \( \varepsilon \)-Nash equilibrium (in a Nash equilibrium \( \varepsilon = 0 \)). We first calculate the values of deviations for each action \( a_m \) using equation 2, and then select the action profile that minimizes the maximum benefit to deviation, as in equation 3

\[ \varepsilon(a^*_i \in A_i) = \max_{\forall a_i \in A_i} u_i(a_i, a_j) - u_i(a^*_i, a_j) \quad \forall j \in N, i \neq j \]  

\( (a_i \in A_i) = \min_{\forall a_i \in A_i} \varepsilon(a_i) \quad \forall i \in N \)  

**Mixed Strategy Nash Equilibrium** We also calculate a version of mixed-strategy Nash equilibrium using the software package Gambit [15]. There are several different solvers for finding Nash equilibria in this toolkit. We used one based on Quantal response equilibrium (QRE) [12] that is based on a tracing procedure with noisy best-response functions for the players. In this limit as the noise in the best responses goes to zero this method converges to a Nash equilibrium. In practice we have found this to be a relatively reliable way to find a sample Nash equilibrium.
Algorithm 1 Minimum Epsilon Bound Algorithm

```
procedure MIN_EPSILON_BOUND(game, upper_bounded_game)
    1: bestepsilon ← −∞
    2: bestoutcome ← [0, 0]
    3: outcome ← [1, 1]
    4: while outcome ≠ null do
        5: epsilon ← −∞
        6: for player ← 1, n do
            7: payoff ← u(player, outcome)
            8: for action ← 1, game(number_of_actions(player)) do
                9: tmp ← outcome[player]
                10: deviation ← upper_bounded_game.u(outcome, player) − payoff
                11: epsilon ← max(epsilon, deviation)
                12: outcome[player] ← tmp
            13: end for
        14: end for
        15: if epsilon < bestepsilon then
            16: bestepsilon ← epsilon
            17: bestoutcome ← outcome
        18: end if
        19: outcome ← outcome.next
    20: end while
    21: return bestoutcome
end procedure
```

Minimum Epsilon Bounded Equilibrium When solving an abstracted game, it is not clear that the best analysis is simply to find a Nash equilibrium of the abstracted game, since this may not be an equilibrium of the original game. This is the idea for minimum epsilon bounded equilibrium (MEB) which represents an improved version of PSNE that considers additional information about the abstraction in the analysis. Instead of considering deviations to clusters of actions (and the average payoff of the cluster), when considering deviations we use the maximum expected payoff for any of the actions in the original game. This allows for a better estimate of how close the outcome will be to an equilibrium in the original game, not the abstracted game. The difference in comparison with PSNE is in the calculation of $\varepsilon(a_m)$. Equation 4 is used to compute the $\varepsilon$ for MEB.

$$\varepsilon(\hat{a}_i \in \hat{A}_i) = \max_{\hat{a}_i, \hat{a}_j \in \hat{A}_i} \left[ \pi_i(\hat{a}_i, \hat{a}_j) - \hat{u}_i(\hat{a}_i, \hat{a}_j) \right]$$

In the above equation $\pi_i(\hat{a}_i, \hat{a}_j)$ returns a payoff from an upper bound game $\mathcal{R}$. Payoffs for the upper-bounded game $\mathcal{R}$ are computed using equation 5, 6 and 7. The first two equations, 5 and 6, calculate the maximum expected payoff for an abstracted action by reverse mapping to the original actions, calculating the expected payoff for every original action, and selecting the maximum one. Equation 7 iterates over all the actions for every player and calculates the payoffs for the upper-bounded game $\mathcal{R}$. 
Unlike equation 2, equation 4 cannot be used in the original game because we need an upper-bounded game where we use reverse mapping. Unless we have an abstracted game it is not possible to compute an upper-bounded game.

\[
E[a_{i,k}] = \sum_{\forall a_{j,l} \in g(\hat{a}_j)} u_i(a_{i,k}, a_{j,l}) \quad j \in N, i \neq j \tag{5}
\]

\[
E_{\text{max}}[\hat{a}_i] = \max_{\forall a_{i,k} \in g(\hat{a}_i)} E[a_{i,k}] \quad i \in N \tag{6}
\]

\[
\bar{u}_i(\hat{a}_i, \hat{a}_j) = E_{\text{max}}[\hat{a}_i] \quad \forall \hat{a}_i \in \hat{A}_i(R), \forall \hat{a}_j \in \hat{A}_j(R), (i, j) \in N, i \neq j \tag{7}
\]

**Algorithm 2 Upper Bounded Game Algorithm**

1: procedure GETUPPERBOUNDEDGAME(game) \(\triangleright\)
2: upperboundedgame \(\leftarrow\) MatrixGame
3: outcome \(\leftarrow\) [1, 1]
4: while outcome \(\neq\) null do \(\triangleright\) iterate all the outcome
5: payoff \(\leftarrow\) MaxExpectedPayoff(game, outcome)
6: upperboundedgame(outcome).payoff \(\leftarrow\) payoff \(\triangleright\) set the payoff to the upper bounded game’s outcome
7: outcome \(\leftarrow\) outcome.next
8: end while
9: return upperboundedgame \(\triangleright\) return the upper bounded game
10: end procedure

The details are shown in Algorithm 2. Algorithm 1 shows how MEB works. Computation of \(\varepsilon\) can be seen in line 9 to line 14 in Algorithm 1. Algorithm 3 shows how to calculate the maximum expected payoffs.

### 3.4 Reverse Mapping

A reverse mapping is necessary to map solutions found by analyzing the abstracted game back into the original game, since the strategy spaces are not the same. Formally, the reverse mapping function \(g(s_i) \rightarrow s_i\) maps every strategy in the abstracted game into a strategy in the original game. We also use a version of this function for clusters: \(g(c_{i,m}) = \{a_{1,1}, \ldots, a_{i,k}\}\). For a clustering abstraction, the original equilibrium actions obtained by reverse mapping can be obtained using equation 8:

\[
\pi^i(a_{i,k} \in g(c_{i,m})) = \frac{\pi^i(c_{i,m})}{|c_{i,m}|} \tag{8}
\]

In the example (Table 3), the NE for the abstract game R is \{(1, 0), (1, 0)\}. Using the reverse mapping function and equation 8 we found the NE strategy of the original game in Table 1 is \{(0, 0.5, 0, 0.5), (0, 0, 0.5, 0.5)\}.
Algorithm 3 Max Expected Payoff Algorithm

1: procedure MAXEXPECTEDPAYOFF(game, abstractgameoutcome) ▷
2:     maxpayoff ← [−∞, −∞]
3:     for player ← 1..n do ▷ use the reverse
4:         tmpmaxpayoff ← −∞
5:         originalactions1 ← g(abstractgameoutcome[player]) ▷ use the reverse
6:             originalactions2 ← g(abstractgameoutcome[opponentplayer])
7:             for action1 ← originalactions1 do ▷ for every action in originalactions1
8:                 expectedpayoff ← 0
9:                 for action2 ← originalactions2 do ▷ for every action in originalactions2
10:                    outcome ← [action1, action2]
11:                     expectedpayoff ← expectedpayoff + game.u(outcome, player)
12:                 end for
13:             end if
14:         end for
15:         maxpayoff[player] ← tmpmaxpayoff
16:     end for
17: return maxpayoff ▷ return the max expected payoff
18: end procedure

In general, the solution of an abstracted game (e.g., an NE) may not be an exact solution in the original game. We use the same criteria as in ε-Nash equilibrium to evaluate the quality of the solutions once they are mapped back into the original game. The solutions are evaluation based on how close they are to being a Nash equilibrium by calculating the maximum benefit for any player to deviate to another strategy in the original game, which is denoted by ε as before.

4 Subgame Abstraction and Solution Concept

We now introduce a different type of clustering abstraction that is based on the idea of forming subgames of similar strategies using a particular structure. The idea is based on the game reduction technique introduced by Conitzer et al. [3]. The original paper focused mostly on cases where a specific type of structure can be used to solve games exactly using hierarchical reduction, but also notes the possibility of using approximate forms of this structure. We focus on this case, reinterpreting this method as a different type of clustering abstraction and introducing some additional enhancements.

Consider the NFG O shown in Figure 1, which has a structure we will call AIOS (Approximately Identical Outside Subgames). This is a similar structure to ALAGIU (All Lower Actions Give Identical Utility) games [3]. The key idea is to create clusters of strategies for both players that form subgames. Within a subgame, the strategies and payoffs can vary arbitrarily. However, outside of the subgame, the strategies for each player should be a similar as possible in the payoffs for playing against any opponent.
strategy not in the cluster (for now, we will describe our methods for the case of exact equality; later we will loosen this constraint). Note that this is essentially a weakening of the standard clustering condition, which requires that strategies be similar against all opponent strategies.

Suppose the row player is player 1 and column player is player 2. If player 1 decides to play any \(v_i\) strategy, he needs to worry only about the probabilities assigned by player 2 in \(p_j\) strategies, because if player 2 plays from \(q_j\) the payoff is the same for the row player no matter which action he chooses among \(v_i\). We can group \(v_i\) actions into a single cluster \(v\) as shown in Figure 1. Similarly, if player 1 plays any of the \(w_i\) strategies, he needs to worry only about the probabilities assigned in \(q_j\), we can group these actions into a cluster \(w\). By the same reasoning, player 2 can create clusters \(p\) and \(q\) for his strategies.

To build the abstracted game we consider \(v_i\) and \(w_i\) in two clusters, \(c_{1,1}\) and \(c_{1,2}\) respectively. For player 2 \(p_j \in c_{2,1}\) and \(q_j \in c_{2,2}\). The cardinality of the clusters is \(|c_{1,1}| = k, |c_{2,2}| = l, |c_{2,1}| = m, |c_{2,2}| = t\). We also have \(c_1 = \{c_{1,1}, c_{1,2}\}, c_2 = \{c_{2,1}, c_{2,2}\}\). If there are \(n\) clusters for both players there will be \(n\) subgames.

![Fig. 1. A NFG of AIOS class](image)
is different than the uniform mapping used previously. In equation 9 the probabilities $\pi^i(a_{i,k})$ are obtained by solving the subgames.

$$\pi^i(a_{i,k}) = \pi^i(c_{i,m}) \times \pi^i(a_{i,k}) \quad i \in N, \forall a_{i,k} \in g(c_{i,m})$$ (9)

## 5 Iterative Subgame Abstraction and Solution Concept

The AIOS structure introduced is very strict if we require exactly identical payoffs outside of the subgames. Such patterns may be very rare in realistic games. However, it is much more plausible to find approximate forms of this structure. We now introduce formally the idea of a noisy AIOS structure.

We define the $\delta$ parameter to specify how much variation in the payoffs is allowed outside of the subgames. Let $\delta_{i,k}$ be the maximum payoff difference for any pair of actions in cluster $k$ for player $i$ for any strategy of the of the opponent that is not in the same subgame. $\delta_i$ is the maximum of $\delta_{i,k}$ for player $i$, where $k$ can be from 1 to number of clusters.

$$\delta = \max_{i \in N}(\max(\delta_{i,k})), \quad k = 1, ..., |\tilde{A}_i(R)|$$ (10)

$$\delta_{i,k} = \max_{a_{i,m}, a_{i,n} \in g(c_{i,k}), \forall a_{j,t} \neq g(c_{j,k})}(u_i(a_{i,m}, a_{j,t}) - u_i(a_{i,n}, a_{j,t})), \quad (i,j) \in N, i \neq j$$ (11)

Equation 10 picks the maximum $\delta$ considering all of the clusters and players. Equation 11 calculates $\delta$ for one cluster $c_{i,k}$ for a player $i$. For game $O$ in Figure 1 $\delta_{1,1} = \delta_{1,2} = \delta_{2,1} = \delta_{1,2} = 0$. So, $\delta = 0$.

### 5.1 Iterative Subgame Solving

In cases where there are difference outside of the subgames, simply composing the solutions of the subgames may not be an equilibrium of the original game. This is because the solution may occasionally result in play in quadrants of the game that are not on of the subgames solved explicitly, which results in error because the payoffs are not exactly the same for all strategies. Here, we introduce an iterative solution technique that attempts to account for this error. First, we calculate for each strategy in a cluster the expected payoff for this strategy outside of the subgame against the current opponent solution. Then, we modify the subgames using the error offset, and resolve the subgames. This results in a sequence of modified solutions that account for the differences in payoffs outside of the subgames from the previous iteration. We call this methods ISASC for Iterative Subgame Abstraction and Solution Concept.
Consider the subgame $G_1$ in Figure 1. We want to internalize the noise outside of the subgame into the payoffs of subgame. So, before solving $G_1$ we update the payoffs for both player 1 and player 2. For action $v_i$, we calculate the expected utility when player 2 does not play the actions in the subgame. That means that when player 1 plays $v_i$, we calculate the expected utility of $v_i$, let us denote it $\Omega_i$, by considering the probabilities of player 2 playing $q_j$ from previous iteration’s strategy. Then we update the payoffs of $G_1$ for player 1 for action $v_i$ for every $p_j$ by adding the $\Omega_i$. This process is done for all strategies in the game.

$$[u_i(a_i, a_j) = u_i(a_i, a_j) + \Omega_i(a_i)] \quad \forall a_i \in A_i(G), \forall a_j \in A_j(G), \forall (i, j) \in N, i \neq j$$

$$\Omega_i(a_i) = \left[ \sum_{q_j \notin A_j(G)} u_i(a_i, a_j) \times \pi_{T-1}(a_j) \right]$$

The detailed algorithm for updating subgames is shown in 4. Equation 12 is used to update the subgames. Equation 13 is used to compute the expected payoff $\Omega_i$ for an action of player $i$, when player $j$ plays outside of the subgame $G$. $A_i(G)$ gives us the action set for player $i$ in game $G$. $\pi_{T-1}(a_j)$ gives us the probability of action $a_j$ from the equilibrium strategy of iteration $T - 1$.

**Algorithm 4 Update Subgame Algorithm**

1: procedure UPDATESUBGAME(subgame, player) \(\triangleright\)  
2: $S_i \leftarrow \text{actions}(i)$ \(\triangleright\) all the actions of player $i$ in original game  
3: $s_{\text{player}} \leftarrow \text{actions}_{\text{subgame}}(\text{player})$ \(\triangleright\) all the actions of player in subgame  
4: $s_{\text{opponent}} \leftarrow \text{actions}_{\text{subgame}}(\text{opponent})$ \(\triangleright\) all the actions of opponent in subgame  
5: $\pi_{p}(s_{p}) \leftarrow \text{prob}(s_{p})$ \(\triangleright\) probabilities of strategy $s$ of player $p$  
6: for $i \leftarrow 1,$ $s_{\text{player}}.\text{len}$ do \(\triangleright\) for every action of player in subgame  
7: \hspace{1cm} for $j \leftarrow 1,$ $s_{\text{opponent}}.\text{len}$ do \(\triangleright\) for every action of opponent not in subgame  
8: \hspace{2cm} $\omega \leftarrow \omega + u_{\text{player}}(s_{\text{player}}(i), s_{\text{opponent}}(j)) \times \pi_{\text{opponent}}(s_{\text{opponent}}(j))$  
9: \hspace{1cm} end for  
10: end for  
11: for $k \leftarrow 1,$ $s_{\text{opponent}}.\text{len}$ do \(\triangleright\) for every action of opponent in subgame  
12: \hspace{1cm} outcome $\leftarrow [s_{\text{player}}(i), s_{\text{opponent}}(k)]$  
13: \hspace{1cm} subgame(outcome, player) $\leftarrow u_{\text{player}}(outcome) + \omega \triangleright$ update the payoff in subgame for outcome  
14: end for  
15: end for  
16: end procedure

5.2 Variations of ISASC

In this section we present some different versions of ISASC that may reduce the computational burden. We can decide, depending on the support of solution in the abstracted
game, whether to solve a subgame or not. If a subgame does not occur with positive probability in the solution to the abstract game in iteration $T$, it may not need to be re-solved in iteration $T + 1$.

Another criteria that can be used to make different versions of ISASC is to use different solution techniques to solve either the subgames or the abstracted game. For example, when we solve a subgame or abstracted game we can first use the PSNE solution concept to get a NE. Then we calculate the $\epsilon$ for that equilibrium. If the $\epsilon$ is greater than 0 we can use the QRE to get another equilibrium which we can use instead of the equilibrium found using PSNE. Since PSNE is much faster than QRE, this could significantly improve computation time. In our experiments, we test several different variations of the algorithm that differ along these two dimensions, as summarized in Table 5.

<table>
<thead>
<tr>
<th>Support Condition</th>
<th>Subgame solver</th>
<th>Hierarchical game solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISASC-EXP3 $\pi^T_i(\hat{a}_i) &gt; 0$ and $\pi^T_j(\hat{a}_j) &gt; 0$</td>
<td>QRE</td>
<td>QRE</td>
</tr>
<tr>
<td>ISASC-EXP4 $\pi^T_i(\hat{a}_i) &gt; 0$ and $\pi^T_j(\hat{a}_j) &gt; 0$</td>
<td>PSNE+QRE</td>
<td>QRE</td>
</tr>
<tr>
<td>ISASC-EXP5 $\pi^T_i(\hat{a}_i) &gt; 0$ and $\pi^T_j(\hat{a}_j) &gt; 0$</td>
<td>PSNE+QRE</td>
<td>PSNE+QRE</td>
</tr>
<tr>
<td>ISASC-EXP6 $\pi^T_i(\hat{a}_i) &gt; 0$ or $\pi^T_j(\hat{a}_j) &gt; 0$</td>
<td>QRE</td>
<td>QRE</td>
</tr>
<tr>
<td>ISASC-EXP7 $\pi^T_i(\hat{a}_i) &gt; 0$ or $\pi^T_j(\hat{a}_j) &gt; 0$</td>
<td>PSNE+QRE</td>
<td>QRE</td>
</tr>
<tr>
<td>ISASC-EXP8 $\pi^T_i(\hat{a}_i) &gt; 0$ or $\pi^T_j(\hat{a}_j) &gt; 0$</td>
<td>PSNE+QRE</td>
<td>PSNE+QRE</td>
</tr>
</tbody>
</table>

Table 5. Different criterions for different versions of ISASC

6 Experiments

We now present the results of our initial experiments with solving NFG using the different abstraction methods described previously. The main criteria we use is how well the solution methods are able to approximate a Nash equilibria in the original, unobstructed game, as measured by $\epsilon$. We generated test games based on the AIOS model, but with varying levels of $\delta$. That is, we vary the maximum difference between the payoffs outside of the subgame for two strategies in the same cluster. In the extreme case, all of these payoff differences are 0 and solving the subgames results in an exact solution for the original game. As the size of the differences increases, the games are less structured and more like random NFG.

In the experiments we use 100 2-player AIOS games for each $\delta = \{0, 5, 10, 20, 50\}$ with 100 actions for each player (in the original game). We partitioned the strategies of row and column player into 10 clusters with 10 actions each: $|c_1| = |c_2| = 10, |c_{1,m}| = 10, m = 1, 2, ..., 10$. This means that each game has 10 subgames of size $10 \times 10$. In Figure 2 we show the structure of the games used for the experiments. For each of the subgames, all of the payoffs are generated randomly between 0 and 100, so each subgame is a fully random game.

We generated the payoffs outside the subgames in such a way that in every cluster the maximum payoff difference between the payoffs for the actions is $\delta$ for all actions of the opponent that are not part of the subgame. In mathematical terms, $\forall k(\delta_{i,k}) =$
For each cluster we first randomly generate the payoffs for one action against all of the opponent strategies outside of the cluster. Then, for all other actions we generate the payoffs by adding uniform random noise to the payoff for the original seed action in the range \([-\delta/2, \delta/2]\).

Our first experiment focuses on the performance of the ISASC method with varying \(\delta\) and with iteration of the solution technique. In these experiments we solved both the subgames and the abstracted game using QRE. Figure 3 shows how the solution quality (measured by the \(\varepsilon\) in the original game) improves with additional iterations. The solution quality does not improve monotonically, so we remember the best solution found so far. We stop iterating when the strategy does not change from one iteration to the next. To compute the average \(\varepsilon\) for \(\delta\) in each iteration we took the averages of \(\varepsilon\) over 100 games. However, because some games converged early, the later numbers of iterations only average over the games that reached that iteration.

When \(\delta = 0\) the solution can be found exactly in a single iteration because there is no noise. As \(\delta\) increases the error in the solutions increases, as expected. However, we can also see clearly in the Figure 3 that using the iterative improvement scheme can reduce the error in the solution with additional iterations. The first few iterations yield the biggest improvements, with most cases plateauing after the initial improvements. We also note that the biggest improvements come in the cases with the largest values of \(\delta\).
The next experiment compares the solution quality of ISASC with the clustering abstractions using k-means clusters and the three different solution methods applied to the subgames: PSNE, QRE, and MEB. Figure 4 shows the result of the experiment on the same set of games with varying levels of $\delta$. ISASC does very well in cases with low $\delta$, as expected. However, it continues to compare favorably to the other clustering approaches even when the values of $\delta$ are much larger. We also note that MEB generally provides better solution quality than PSNE and QRE.

Finally, we consider different version of ISASC. In Table 5 the support condition column gives the condition for whether to solve a subgame or not. Figure 5 shows the result of this experiment. We can see that when $\delta = 0$ it makes no difference at all. As $\delta$ increases, at first ISASC-EXP8 performs better. Then ISASC-EXP6 did better for higher $\delta$. That means if there is more noise in the game, using QRE for solving both the subgame and abstract game did better. For both ISASC-EXP8 and ISASC-EXP6, the support condition is less strict, so we solve more subgames. Figure 6 shows the solution time of solving subgames for different ISASC. We can see that using PSNE combined with QRE takes less time than using QRE alone as a solver of subgames.
7 Conclusion

Solving large NFG is a fundamental problem in computational game theory, and abstraction is a promising direction for scaling up to solve the largest and most challenging games. However, we still have a very incomplete understanding of different methods for abstraction, and how these interact with solution concepts like Nash equilibrium, even in the relatively simple setting on NFG.

We studied several different approaches that fall into the broad category of clustering abstractions. Even within this type of abstraction there are important variations, including how to do the abstraction, and how to solve the resulting abstractions. The most interesting approach we considered was based on the game reduction methods of Conitzer et al [3]. We showed that this approach is useful as an abstraction method, even when the structure is relaxed to a noisy approximation (which is more likely to exist in real games). We also introduced improvements to this idea including iteratively accounting for the errors outside of the subgames that further improve the solutions.

8 Acknowledgments

This work was supported by the NSF under Grant No. IIS-1253950 and by the Army Research Office under award number W911NF-13-1-0467.
References