Computing Pure Nash Equilibria in Symmetric Action Graph Games

Albert Xin Jiang **Kevin Leyton-Brown**
Department of Computer Science
University of British Columbia
{jiang;kevinlb}@cs.ubc.ca

Abstract

We analyze the problem of computing pure Nash equilibria in action graph games (AGGs), which are a compact game-theoretic representation. While the problem is NP-complete in general, for certain classes of AGGs there exist polynomial time algorithms. We propose a dynamic-programming approach that constructs equilibria of the game from equilibria of restricted games played on subgraphs of the action graph. In particular, if the game is symmetric and the action graph has bounded treewidth, our algorithm determines the existence of pure Nash equilibrium in polynomial time.

Introduction

Game-theoretic models have recently been very influential in the computer science community. Most of the game theoretic literature presumes that simultaneous-action games will be represented in normal form—i.e., that the game's payoff function is a matrix with one entry for each player's payoff under each combination of all players' actions. This is problematic because quite often games of interest have a large number of players and a large set of action choices, and the size of the normal form representation grows exponentially with the number of players. Fortunately, most large games of any practical interest have highly structured payoff functions. For example, if there exist strict payoff independencies between players, a game can be more compactly written as a graphical game (Kearns, Littman, & Singh 2001). Such a game can be visualized using a graph whose vertices correspond to agents, and whose edges correspond to dependencies between agents' utility functions. If a game's structure takes the form of anonymity or context-specific payoff independencies, it can be more compactly represented as an action graph game (AGG) (Bhat & Leyton-Brown 2004; Jiang & Leyton-Brown 2006). An AGG can be visualized using a graph whose vertices correspond to actions, and whose edges correspond to dependencies between the utilities of agents who take these actions. AGGs are fully expressive, i.e. they can represent arbitrary games. AGGs are always at least as compact as graphical games, and can be exponentially more compact for certain structured games.

Nash equilibrium is the most important solution concept in game theory. Such equilibria come in two varieties. When we allow mixed-strategy equilibria, we know that every finite game has a Nash equilibrium (Nash 1951), and that computing such an equilibrium is PPAD-complete (Chen & Deng 2006). Pure-strategy Nash equilibria are not guaranteed to exist, although they are often more interesting than their mixed-strategy cousins; for example, they can be easier to implement in practice. Various work has considered approaches for finding such equilibria under various game representations (Gottlob, Greco, & Scarcello 2003; Daskalakis & Papadimitriou 2006; Ieong *et al.* 2005; Brandt, Fischer, & Holzer 2007).

In this paper, we analyze the problem of finding pure Nash equilibria in AGGs. While the problem is NP-compete in general, we identify classes of AGGs for which this problem is tractable. We propose a dynamic programming approach that uses tree decomposition techniques to break an action graph into subgraphs, and constructs equilibria of the game from equilibria of restricted games on the subgraphs. In particular, we show that if the AGG is symmetric and the action graph has bounded treewidth, our algorithm determines the existence of pure equilibria in polynomial time. Though space does not permit us to provide the results here, our result can also be extended beyond symmetric games.

Related Work

Gottlob, Greco, & Scarcello (2003) and Daskalakis & Papadimitriou (2006) both analyzed the problem of finding pure equilibria in graphical games, and proposed dynamic programming algorithms based on hypertree decomposition and tree decomposition, respectively. Our dynamic programming approach for AGGs similarly relies on tree decomposition, and indeed simplifies to the equivalent of Daskalakis & Papadimitriou's algorithm on graphical games represented as AGGs. However, on general AGGs we face an additional difficulty, because an agent can deviate from playing an action in one part of the action graph to another.

Ieong *et al.* (2005) proposed a dynamic programming algorithm for finding pure equilibria in *singleton congestion games*. These games can be represented as AGGs with only self edges. Ieong *et al.*'s algorithm builds equilibria from restricted games played on subsets of actions. Our approach deals with agents' deviations (the problem mentioned above) in a way similar to Ieong *et al.*, using the worst current utility and best entrant utility of restricted games.

Action Graph Games

Definition 1. *An action graph game (AGG) is a tuple* $\langle N, \mathbf{S}, (S, E), u \rangle$*, where*

- $N = \{1, \ldots, n\}$ *is the set of agents,*
- $\mathbf{S} = \prod_{i \in N} S_i$ *is the set of action profiles, where* $\prod$ *is the Cartesian product and* S_i *is agent* i*'s set of actions. We denote by* $s_i \in S_i$ *one of agent* i*'s actions, and* $\mathbf{s} \in \mathbf{S}$ *an action profile.*
- *Agents' action sets may partially or completely overlap.* S *is the set of distinct actions. In other words,* $S_i \subseteq S$ *for all* i*, and* $S = \bigcup_{i \in N} S_i$*.*
- $G \equiv (S, E)$ *is the* action graph*, a directed graph with* S *as the set of vertices. We say* s' *is a* neighbor *of* s *if* $(s', s) \in E$*. Let* $\nu(s)$ *denote the set of neighbors of* s*, i.e.* $\nu(s) \equiv \{s' \in S | (s', s) \in E\}$*. Let* Δ *denote the set of* configurations *of agents over actions. A configuration* $D \in \Delta$ *is an* $|S|$*-tuple of integers* $(D[s])_{s \in S}$*, where* $D[s]$ *specifies the number of agents that chose action* $s \in S$*. For a subset of actions* $X \subset S$*, let* $D[X]$ *denote the restriction of* D *over* X*, i.e.* $D[X] = (D[s])_{s \in X}$*. Similarly, let* $\Delta[X]$ *denote the set of restricted configurations over* X*.*
- u *is a* $|S|$*-tuple* $(u^s)_{s \in S}$*, where each* $u^s : \Delta[\nu(s)] \mapsto \mathbb{R}$ *is the* utility function *for* s*. Semantically,* $u^s(D[\nu(s)])$ *is the utility of an agent who chose action* s*, when the configuration over* $\nu(s)$ *is* $D[\nu(s)]$*.*

Let $\mathcal{U}$ be the set of distinct utilities of the game Γ. For notational convenience, let $u_i(\mathbf{s})$ denote agent i's utility under action profile $\mathbf{s}$, i.e. $u_i(\mathbf{s}) = u^{s_i}(D[\nu(s_i)])$ where $\forall x \in \nu(s_i)$, $D[x] = |\{j \in N | s_j = x\}|$. Let s_{-i} denote the tuple of actions for agents other than i.

Intuitively, AGGs capture two types of structure in games:

1. Shared actions capture the game's *anonymity* structure: agent i's utility depends only on her action s_i and the configuration (i.e. number of players that play each action), but not on the identities of the players.
2. The (lack of) edges between nodes in the action graph expresses *context-specific independencies* of utilities of the game: $\forall i \in N$, if i chose action $s \in S$, then i's utility depends only on the configuration over the neighborhood of s. In other words, the configuration over actions not in $\nu(s)$ does not affect i's utility.

Definition 2. *An AGG is* symmetric *if all players have identical action sets, i.e. if* $S_i = S$ *for all* i*.*

Note that in a symmetric AGG, all agents have the same utility functions, i.e., a symmetric AGG represents a *symmetric game*, in which all agents are identical.

Definition 3. *An AGG is* k*-symmetric if there exists a partition* $\{N_1, \ldots, N_k\}$ *of* N *such that for all* $l \in \{1, \ldots, k\}$*, for all* $i, j \in N_l$*,* $S_i = S_j$*.*

Intuitively, k-symmetric AGGs represent games having k classes of agents; agents within each class are identical.

The following are several properties of the AGG representation. Due to space constraints we omit the proofs of these facts and refer the readers to (Jiang & Leyton-Brown 2006).

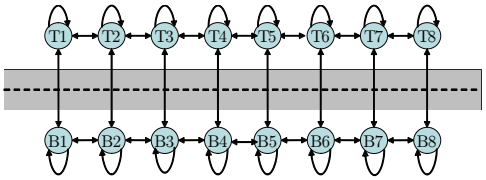

Figure 1: Action graph for the road game with $m = 8$.

- AGGs are fully expressive: any game can be represented as an AGG.
- Symmetric AGGs can represent arbitrary symmetric games.
- As with other game representations, the size of an AGG representation is dominated by the size of its utility functions. For all AGGs Γ, let $||\Gamma|| \equiv \sum_{s \in S} |\Delta[\nu(s)]|$ denote the number of utility values the representation stores, then $||\Gamma|| \leq |S|\binom{n-1+\mathcal{I}}{I} \equiv |S|\frac{(n-1+\mathcal{I})!}{(n-1)!\mathcal{I}!}$, where $\mathcal{I} \equiv \max_{s \in S} |\nu(s)|$ is the maximum in-degree of the action graph G. If $\mathcal{I}$ is bounded by a constant, $||\Gamma|| = O(|S|n^{\mathcal{I}})$.
- Any graphical game can be encoded as an AGG in which all action sets are disjoint. The transformation takes polynomial time and the resulting AGG has the same space complexity as the graphical game. The converse is not true: for certain AGGs, the equivalent graphical games are exponentially larger. In particular, for any symmetric AGG with at least one edge in its action graph, the equivalent graphical game is a clique and its size is no better than the normal form.

Example 1. *Suppose each of* n *agents is interested in opening a business, and can choose to locate in any block along either side of a road of length* m*. Multiple agents can choose the same block. Agent* i*'s payoff depends on the number of agents who chose the same block as he did, as well as the numbers of agents who chose each of the adjacent blocks of land. This game can be compactly represented as a symmetric AGG, whose action graph is illustrated in Figure 1.*

Notice that each node has at most four incoming edges, regardless of the length of the road m. Thus for all m, The AGG representation of a road game with length m stores only $O(|S|n^4) = O(2mn^4)$ payoffs. Also notice that any pair of agents can potentially affect each other's payoffs by choosing adjacent locations. This means that the graphical game representation of this game is a clique, and its space complexity is the same as that of the normal form (exponential in n).

Complexity of Finding Pure Equilibria

Definition 4. *An action profile* $\mathbf{s} \in \mathbf{S}$ *is a* pure Nash equilibrium *of the game* Γ *if for all* $i \in N$*, for all* $s'_i \in S_i$*,* $u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i})$*.*

Intuitively, in a pure Nash equilibrium no agent can profitably deviate from her chosen action. An obvious algorithm for finding pure equilibria of a game is to check every possible action profile. This algorithm runs in linear time in

the normal form representation of the game. However, since AGGs can be exponentially more compact than the normal form, the running time of this algorithm is worst-case exponential in the size of the AGG. Indeed, the problem becomes NP-complete when the input is an AGG.

Theorem 1. *The problem of determining whether a pure Nash equilibrium exists in an AGG is NP-complete.*

Proof Sketch. It is straightforward to see that the problem is in NP, because given a pure strategy profile it takes polynomial time to verify whether it is a Nash equilibrium. NP-hardness follows from the fact that any graphical game can be transformed (in poly-time) to an equivalent AGG of the same space complexity, and the fact that the problem of determining the existence of pure equilibrium in graphical games is NP-hard (Gottlob, Greco, & Scarcello 2003). □

Indeed, the problem remains hard even if we restrict the games to be symmetric. The proof[1] (a reduction from 3SAT) is omitted due to space constraints.

Theorem 2. *The problem of determining whether a pure Nash equilibrium exists in a symmetric AGG is NP-complete, even when the in-degree of the action graph is at most 3.*

Now we look at classes of AGGs in which $|S|$, the number of action nodes, is bounded by some constant. We show that in this case, the problem of finding pure equilibria can be solved in polynomial time. While this is a very restricted class of AGGs, we will use the results of this subsection as building blocks for our dynamic programming approach to solve more complex AGGs.

We first look at symmetric AGGs. The following Lemma allows us to consider only the configurations instead of all the pure strategy profiles.

Lemma 3. *Suppose* Γ *is a symmetric AGG. If* $\mathbf{s}$ *and* $\mathbf{s}'$ *induce the same configuration, then* $\mathbf{s}$ *is a pure equilibrium of* Γ *iff* $\mathbf{s}'$ *is a pure equilibrium of* Γ.

We say a configuration D is a pure equilibrium of Γ if its corresponding pure strategies are pure equilibria. Given a configuration D, we can check whether it is a pure equilibrium in polynomial time.

Theorem 4. *The problem of determining whether a pure Nash equilibrium exists in a symmetric AGG with bounded* $|S|$ *is in* P.

Proof. A polynomial algorithm is to check all configurations. Since $|S|$ is bounded, the number of configurations $\binom{n+|S|-1}{|S|-1} = O(n^{|S|-1})$ is polynomial. □

This can be easily extended to k-symmetric AGGs.

Definition 5. *Suppose* Γ *is a* k*-symmetric AGG with the partition* $\{N_1, \ldots, N_k\}$ *and the corresponding set of distinct action sets* $\{S^1, \ldots, S^k\}$. *Then given a pure strategy profile* $\mathbf{s}$*, its corresponding* k-configuration *is a tuple* $(D_l)_{1 \le l \le k}$ *where* D_l *is the configuration over* S^l *induced by the players in* N_l. *In other words, for all* $s \in S^l$, $D_l[s] = |\{i \in N_l | s_i = s\}|$.

Just as configurations capture all relevant information about pure strategy profiles in symmetric games, k-configurations capture all relevant information about pure strategy profiles in k-symmetric games. Thus we can determine the existence of pure equilibrium by checking all k-configurations. When k is bounded by a constant, there are polynomial number of k-configurations.

Lemma 5. *The problem of determining whether a pure Nash equilibrium exists in a* k*-symmetric AGG with bounded* $|S|$ *and bounded* k *is in* P.

Proof. A polynomial algorithm is to check all k-configurations. Since $|S|$ is bounded, for each $l \in \{1, \ldots, k\}$ the number of distinct D_l is $\binom{|N_l|+|S^l|-1}{|S^l|-1} = O(|N_l|^{|S^l|-1})$. Therefore the number of distinct k-configurations is $O(n^{k(|S|-1)})$, which is polynomial when k is bounded. For each k-configuration, checking whether it is a Nash equilibrium takes polynomial time. Therefore the algorithm runs in polynomial time. □

Now consider the full class of AGGs with bounded $|S|$. Interestingly, our problem is remains easy to solve.

Theorem 6. *The problem of determining whether a pure Nash equilibrium exists in an arbitrary AGG with bounded* $|S|$ *is in* P.

Proof. Any AGG Γ is k-symmetric by definition, where k is the number of distinct action sets. Since $S_i \subseteq S$ for all i, the number of distinct nonempty action sets is at most $2^{|S|} - 2$. Since $|S|$ is bounded by a constant, there are a bounded number of distinct action sets. Thus Γ is k-symmetric with bounded k, and Lemma 5 applies. □

Dynamic Programming

We now consider classes of AGGs in which $|S|$ is not bounded. Whereas enumerating the configurations works well for AGGs with bounded $|S|$, this approach is less effective in the general case with unbounded $|S|$: in a symmetric AGG, the number of configurations over S is $\binom{n+|S|-1}{|S|-1}$, which is superpolynomial in $||\Gamma||$ when $\mathcal{I}$ is bounded.

Our approach is to use dynamic programming to construct pure equilibria of the game from pure equilibria of games restricted to parts of the action graph. While the NP-completeness results from the previous section imply that our approach is unlikely to be tractable for all AGGs, we identify classes of AGGs for which our approach does yield a polynomial algorithm.

For a set of actions $R \subset S$, let G_R be the action graph $G = (S, E)$ restricted to the action nodes R. Formally, $G_R \equiv (R, \{(s,t) \in E | s \in R, t \in R\})$.

For a set of actions $X \subset S$, define $\nu(X) \equiv \{s \in S \setminus X | \exists x \in X$ such that $(s,x) \in E\}$: the set of actions not in X that are neighbors of some action in X. Also define $\nu(\overline{X}) \equiv \{x \in X | \exists s \in S \setminus X$ such that $(x,s) \in E\}$, the set of actions in X that are neighbors of some action not in X.

[1] The proof is based on unpublished personal communications with Vincent Conitzer.

Let $\rho(X) \equiv \nu(X) \cup \nu(\overline{X})$. Given a configuration $D[X]$, let $\#D[X] \equiv \sum_{x \in X} D[x]$.

Given a pure strategy profile $\mathbf{s} = (s_1, \ldots, s_n)$ and a set of actions $R \subset S$, the *restricted strategy profile* $\mathbf{s}|_R$ is a tuple $(N', \mathbf{s}_{N'})$ where $N' = \{i \in N | s_i \in R\}$ is the set of players that chose actions in R and $\mathbf{s}_{N'} = (s_i)_{i \in N'}$ is the tuple of their actions.

Now we introduce the concept of a *restricted game* on $R \subset S$, which intuitively is the game played by a subset $N' \subseteq N$ of players when we "restrict" them to the subgraph G_R, i.e. require them to choose their actions from R. Of course, the utility functions of this restricted game are not defined until we specify a configuration on $\nu(R)$.

Definition 6. *Given an AGG Γ, a set of actions $R \subset S$, a configuration $D[\nu(R)]$ and $N' \subseteq N$, we define the* restricted game $\Gamma(N', R, D[\nu(R)])$ *to be an AGG with the set N' of players and with G_R as the action graph. For each player $i \in N'$, her action set is $S'_i = S_i \cap R$. Each action $s \in R$ has the utility function $u^s|_{D[\nu(R)]}$, which is the same as u^s as defined in Γ except that the configuration of nodes outside R is assigned by $D[\nu(R)]$. Formally,* $\Gamma(N', R, D[\nu(R)]) = \left\langle N', \prod_{i \in N'}(S_i \cap R), G_R, \left(u^s|_{D[\nu(R)]}\right)_{s \in R} \right\rangle$.

It is easy to see that a pure equilibrium on Γ induces a pure equilibrium on the game restricted to G_R.

Lemma 7. *Suppose $\mathbf{s}$ is a pure equilibrium of Γ, and its restricted profile on $R \subset S$ is $\mathbf{s}|_{\mathbf{R}} = (N', \mathbf{s}_{N'})$. Then $\mathbf{s}_{N'}$ is a pure equilibrium of the restricted game $\Gamma(N', R, D[\nu(R)])$, where D is the configuration induced by $\mathbf{s}$.*

We want to use equilibria of restricted games as building blocks to construct equilibria of the entire game. Of course, a restricted game on $R \subset S$ is not well-defined until we specify $D[\nu(R)]$. Thus we define a *partial solution*, which describes a restricted game as well as a pure equilibrium of it, as follows.

Definition 7. *For $R \subset S$, a* partial solution *on R is a restricted strategy profile on $R \cup \nu(R)$, $\mathbf{s}|_{R \cup \nu(R)}$, such that its restriction on R, $\mathbf{s}|_R = (N', \mathbf{s}_{N'})$, is a pure equilibrium of the restricted game $\Gamma(N', R, D[\nu(R)])$.*

We say a partial solution $\mathbf{s}|_{R \cup \nu(R)}$ can be *extended* if there exists a pure strategy profile $\mathbf{s}^*$ such that $\mathbf{s}^*$ is a pure equilibrium of Γ and $\mathbf{s}^*|_{R \cup \nu(R)} = \mathbf{s}|_{R \cup \nu(R)}$.

In order to combine partial solutions to form a partial solution on a larger subgraph, we need to make sure that the result is a valid restricted strategy profile. We say two partial solutions $\mathbf{s}'|_X$ and $\mathbf{s}''|_Y$ are *consistent* if there exists a pure strategy profile $\mathbf{s}$ such that $\mathbf{s}|_X = \mathbf{s}'|_X$ and $\mathbf{s}|_Y = \mathbf{s}''|_Y$. It is straightforward to see that two partial solutions $\mathbf{s}'|_X = (N', \mathbf{s}'_{N'})$ and $\mathbf{s}''|_Y = (N'', \mathbf{s}''_{N''})$ are consistent iff for all $i \in N' \cap N''$, $s'_i = s''_i$.

However, if we simply combine two consistent partial solutions that describe equilibria of restricted games on two disjoint sets $X, Y \in S$, the result would not necessarily induce an equilibrium of the restricted game on $X \cup Y$. This is because an agent who was playing an action in X might profitably deviate by playing an action in Y, and vice versa.

We could deal with this problem by keeping track of all pure equilibria of each restricted game, and determine case-by-case whether two equilibria can be combined (by checking whether agents could profitably deviate from one restricted game to the other). But as we combine the restricted games to form larger restricted games and eventually the unrestricted game on the entire action graph G, the number of equilibria we would have to store could grow exponentially.

Perhaps we don't need to keep track of all partial solutions. Imagine we had a function ch that summarized them, i.e. it mapped each partial solution to a *characteristic* from a finite set $\mathcal{C}$ which is smaller than the set of partial solutions. For this characteristic function to be useful, it need to be *equilibrium-preserving*, defined as follows.

Definition 8. *For $X \subset S$, a function* ch() *that maps partial solutions to their characteristics is* equilibrium-preserving *if for all pairs of partial solutions $\mathbf{s}|_X$, $\mathbf{s}'|_X$, if ch$(\mathbf{s}|_X)$ = ch$(\mathbf{s}'|_X)$ then ($\mathbf{s}|_X$ can be extended) $\Leftrightarrow$ ($\mathbf{s}'|_X$ can be extended).*

Intuitively, an equilibrium-preserving characteristic function ch() induces a partition of the set of partial solutions into equivalence classes. All partial solutions with the same characteristic behave the same way, so we only need to consider the set of all distinct characteristics. For $X \subset S$, we define $A_X \subset \mathcal{C}$ to be the set of characteristics of partial solutions on X. Formally, $A_X = \{\text{ch}(\mathbf{s}|_{X \cup \nu(X)}) \mid \mathbf{s}|_{X \cup \nu(X)} \text{ is a partial solution on } X\}$.

Given such a function ch, a dynamic-programming algorithm for determining the existence of pure equilibria of Γ is:

1. Partition S into $\mathcal{X} = \{X_1, \ldots, X_m\}$ such that the size of each X_i is bounded by a constant.
2. For each $X_i \in \mathcal{X}$, compute A_{X_i}, the set of characteristics of partial solutions on X_i.
3. While $|\mathcal{X}| \geq 2$:
   (a) Take $X, Y \in \mathcal{X}$. Remove them from $\mathcal{X}$.
   (b) Compute $A_{X \cup Y}$ from A_X and A_Y.
   (c) Add $X \cup Y$ to $\mathcal{X}$.
4. Now $\mathcal{X}$ has only one member, S.
5. Return TRUE iff A_S is not empty.

Since a partial solution on S is by definition a pure equilibrium of Γ, there exists a pure equilibrium of Γ if and only if A_S is not empty. For this algorithm to run in polynomial time, the function ch() must satisfy the following properties:

Property 1: At all times during the algorithm, for all $X \in \mathcal{X}$, the size of A_X is polynomial. This is necessary since all restricted strategy profiles could potentially be partial solutions, and so A_X could potentially be the set of all possible characteristics for X.

Property 2: For each X_i of bounded size, A_{X_i} can be computed in polynomial time.

Property 3: $A_{X \cup Y}$ can be computed from A_X and A_Y in polynomial time.

The NP-Completeness results from the previous section imply that we will not find a ch() that satisfies the above for

general AGGs unless P=NP. Nevertheless, in the following sections we show that for certain classes of AGGs there exist ch()'s that do satisfy the above properties, meaning that our dynamic programming algorithm determines the existence of pure Nash equilibrium in polynomial time for those classes of AGGs.

Symmetric AGGs

Now we focus on applying our dynamic programming approach to symmetric AGGs. Since in this case all players have the same action set S, we can identify a symmetric AGG by the tuple $\langle n, G = (S, E), u\rangle$. Similarly, given a symmetric AGG Γ, $X \subset S$, a configuration $D[\nu(X)]$ and $n' \leq n$, we define the *restricted game* $\Gamma(n', X, D[\nu(X)]) = \left\langle n', G_X, \left(u^x|_{D[\nu(X)]}\right)_{x\in X}\right\rangle$. Lemma 3 tells us that we only need to consider configurations instead of strategy profiles. Likewise, for the subgraph restricted to $X \subset S$, instead of restricted strategy profiles we only need to consider restricted configurations $D[X]$. The following lemma is analogous to Lemma 7.

Lemma 8. *If D^* is a pure equilibrium of Γ, then $D^*[X]$ is a pure equilibrium of the restricted game $\Gamma(\#D^*[X], X, D^*[\nu(X)])$.*

We now adapt the relevant concepts introduced in the previous section to symmetric AGGs, so that we use configurations instead of strategy profiles. A partial solution on $X \subseteq S$ is a configuration $D[X \cup \nu(X)]$ such that $D[X]$ is a pure equilibrium of the restricted game $\Gamma(\#D[X], X, D[\nu(X)])$. The following Lemma shows that it is simple to check whether $D[X]$ and $D'[Y]$ are consistent.

Lemma 9. *Given $X, Y \subseteq S$, $D[X]$ is consistent with $D'[Y]$ iff*

1. *for all $s \in X \cap Y$, $D[s] = D'[s]$, and*
2. *Let $n' = \#D[X] + \#D'[Y \setminus X]$, then $n' \leq n$. Furthermore, if $X \cup Y = S$ then $n' = n$.*

For two configurations $D[X], D'[Y]$ that are consistent with each other, we define $D[X] \cup D'[Y]$ to be the (unique) configuration on $X \cup Y$ that is consistent with both $D[X]$ and $D'[Y]$.

Recall that a partial solution on X can be combined with a partial solution on Y to form a partial solution on $X \cup Y$ if they are consistent, and if no player who plays an action in X can profitably deviate to an action in Y and vice versa.

Definition 9. *Given a restricted game Γ' and an equilibrium D^* of Γ', the* worst current utility *WCU(D^*, Γ') is the utility of the worst-off player, or ∞ if Γ' has 0 players. The* best entrance utility *BEU(D^*, Γ') is the best payoff a player outside of Γ' can get by playing an action in Γ', assuming the current players in Γ' play D^*. If Γ' already has all n players, BEU$(D^*, \Gamma') = -\infty$.*

We observe that since all agents in a symmetric game are identical, to check whether agents could profitably deviate from one restricted game Γ' currently in equilibrium D' to another restricted game Γ'' in equilibrium D'', we just need to check whether WCU(D', Γ') is greater than BEU(D'', Γ''). In other words, WCU(D', Γ') and BEU(D', Γ') can be used as sufficient statistics for checking existence of profitable deviations out of and into restricted game Γ'. This allows us to use the following characteristic function.

Lemma 10. *Consider the characteristic function ch that maps a partial solution $D[X \cup \nu(X)]$ to ch$(D[X \cup \nu(X)]) = (D[\rho(X)], \#D[X],$ WCU$(D[X], \Gamma'),$ BEU$(D[X], \Gamma'))$ where $\Gamma' = \Gamma(\#D[X], X, D[\nu(X)])$. Then ch is equilibrium-preserving.*

Intuitively, we need $\#D[X]$ and $D[\nu(X)]$ to identify the restricted game on X, so that we can solve the restricted game in polynomial time when $|X|$ is bounded (Theorem 4). We need $D[\rho(X)]$ and $\#D[X]$ to check if the partial solutions on X with this characteristic are consistent with partial solutions on another subgraph. Finally we need WCU and BEU to check whether agents can profitably deviate into or out of the restricted game Γ'.

The following lemma shows how sets of characteristics from two disjoint subsets of S can be combined together.

Lemma 11. *Suppose X' and X'' are disjoint subsets of S, and $X' \cup X'' = X$. For all $D[\rho(X)]$, $B \leq n$, and $U_c, U_e \in \mathcal{U}$, $(D[\rho(X)], B, U_c, U_e) \in A_X$ iff there exists some $D'[\rho(X')]$, $D''[\rho(X'')]$, $B', B'' \leq B$, and $U'_c, U''_c, U'_e, U''_e \in \mathcal{U}$ such that*

1. *$(D'[\rho(X')], B', U'_c, U'_e) \in A_{X'}$,*
2. *$(D''[\rho(X'')], B'', U''_c, U''_e) \in A_{X''}$,*
3. *$D'[\rho(X')]$ is consistent with $D''[\rho(X'')]$,*
4. *$D[\rho(X)]$ is consistent with $D'[\rho(X')] \cup D''[\rho(X'')]$,*
5. *$B = B' + B''$,*
6. *$U_c = \min\{U'_c, U''_c\}$, and $U_e = \max\{U'_e, U''_e\}$,*
7. *$U'_c \geq U''_e$, and $U''_c \geq U'_e$.*

Let us now consider the size of A_X. Since WCU$(D', \Gamma'),$ BEU$(D', \Gamma') \in \mathcal{U}$ for all D' and Γ', each has at most $|\mathcal{U}| \leq ||\Gamma||$ distinct values. Also $\#D[X] \in \{0, \ldots, n\}$ by definition. Furthermore, $\rho(X) \subseteq X \cup \nu(X)$. So the number of distinct characteristics $(D[\rho(X)], \#D[X],$ WCU$(D[X], \Gamma'),$ BEU$(D[X], \Gamma'))$ can be much smaller than the number of corresponding partial solutions $D[X \cup \nu(X)]$, especially if $|\rho(X)| \ll |X \cup \nu(X)|$. However, as X gets larger $\rho(X)$ could also grow. $|\nu(X)|$ is $|X|\mathcal{I}$ in the worst case, so the number of possible configurations over $\nu(X)$ is superpolynomial in $||\Gamma||$ in the worst case. Since A_X could potentially include every distinct tuple $(D[\rho(X)], B, U_c, U_e)$, the size of A_X is superpolynomial in the worst case. Indeed, Theorem 2 showed that we will not find a poly-time algorithm for general symmetric AGGs unless P = NP. However, if the action graph G has certain structure and we could combine the restricted games in a way such that $|\rho(X)|$ remains small as X grows, then $\forall X$, $|A_X|$ would remain polynomial in $||\Gamma||$, and our algorithm would run in polynomial time.

Action Graphs with Bounded Treewidth

One way to characterize this kind of structure is the concept of treewidth, introduced by Robertson & Seymour (1986).

Given $G = (S, E)$, define $\mathcal{H}(G)$ to be the hypergraph $(S, \mathcal{E})$ with $\mathcal{E} = \{\{s\} \cup \nu(s) | s \in S\}$. In other words, for

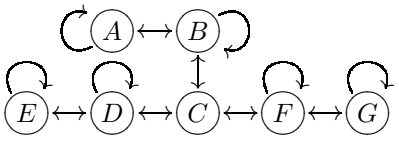

Figure 2: An action graph.

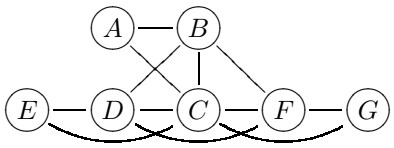

Figure 3: The primal graph.

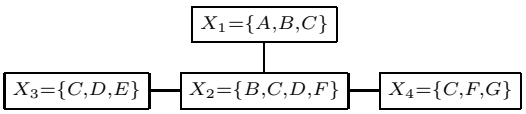

Figure 4: Tree decomposition of Figure 3.

each action $s \in S$, there is a hyperedge containing s and its neighbors. Duplicate hyperedges are removed.

Let G' be the *primal graph* of the hypergraph $\mathcal{H}(G)$. G' is a undirected graph on the same set of vertices, and there is an edge between two nodes if they are in some hyperedge in $\mathcal{H}(G)$. $G' = (S, \{\{u, v\} | \exists h \in \mathcal{E}$ such that $u, v \in h\})$. Thus for each $s \in S$, s and its neighbors in G form a clique in G'. In the Bayes net literature G' is also known as the *moral graph* of G. For example, Figure 2 shows the action graph G of a symmetric AGG. Its hypergraph $\mathcal{H}(G)$ has the same set of vertices and the hyperedges $\{A, B\}$, $\{A, B, C\}$, $\{D, E\}$, $\{C, D, E\}$, $\{F, G\}$, $\{C, F, G\}$, and $\{B, C, D, E\}$. Figure 3 shows G's primal graph G'.

Definition 10. *A* tree decomposition *of an undirected graph $G' = (V, E)$ is a pair $(\mathcal{X}, T)$ with $T = (I, F)$ a tree (where I and F are the nodes and edges of the tree respectively), and $\mathcal{X} = \{X_i | i \in I\}$ a family of subsets of V, one for each node of T, such that*

- $\bigcup_{i \in I} X_i = V$,
- *for all edges $\{v, w\} \in E$ there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$, and*
- *for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The width *of a tree decomposition is* $\max_{i \in I} |X_i| - 1$. *The* treewidth $tw(G')$ *of a graph G' is the minimum width over all tree decompositions of G'.*

Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree decomposition of the primal graph G', with width w. Figure 4 shows a tree decomposition of the primal graph G' from Figure 3. Each node $i \in I$ of the tree is labeled with X_i.

Let the treewidth $tw(\Gamma)$ of an AGG Γ be the treewidth of $und(G)$, the undirected version of its action graph G (excluding self-edges). Then $tw(\Gamma) \leq tw(G')$ because the nodes in the two graphs are the same, and the set of edges of $und(G)$ is a subset of the set of edges of G'. Our algorithm in this subsection is based on a tree decomposition of the primal graph G', and its running time directly depends on $tw(G')$. Nevertheless, in Theorem 14 we will link the complexity of our algorithm with $tw(\Gamma)$.

The following is a well-known property of tree decompositions.

Lemma 12 (e.g. Kloks (1994)). *If X is a clique in G', then $\exists i \in I$ such that $X \subseteq X_i$.*

Since s and its neighbors in G form a clique in G', this implies that for all $s \in S$, $\exists i \in I$ such that $\{s\} \cup \nu(s) \subseteq X_i$. Assign each $s \in S$ to such a node i of the tree. Let R_i be the set of actions assigned to $i \in I$. Then $R_i \cup \nu(R_i) \subseteq X_i$ and $\{R_i | i \in I\}$ is a partition of S. Intuitively, this is why we work with a tree decomposition on the primal graph G' instead of a tree decomposition on the action graph: a tree decomposition on G' guarantees that we are able to partition S into $\{R_i | i \in I\}$ such that for each R_i, all actions that affect the restricted game on R_i are associated with the node i of the tree decomposition. For our tree decomposition in Figure 4, $R_1 = \{A, B\}$, $R_2 = \{C\}$, $R_3 = \{D, E\}$ and $R_4 = \{F, G\}$.

Pick an arbitrary node $r \in I$ to be the root of T. We say node j is a descendant of node i (equivalently i is an ancestor of j) if i is on the path from r to j. Define $Y_i = \{v \in R_j | j = i$ or j is a descendant of $i\}$. Then $Y_r \equiv S$. Intuitively, when we combine the restricted games associated with node i and its descendants in T, we would get a restricted game on Y_i. For each node $i \in I$ with children $c_1, \ldots, c_m \in I$, for each $j \leq m$, define $Z_{i,j} = R_i \cup Y_{c_1} \cup \ldots \cup Y_{c_j}$. This implies that $Z_{i,m} \equiv Y_i$. For our tree decomposition in Figure 4, if we let node 1 to the the root, then $Y_3 = R_3$, $Y_4 = R_4$, $Y_2 = R_2 \cup R_3 \cup R_4 = \{C, D, E, F, G\}$, and $Y_1 = S$. Since node 2 has two children $c_1 = 3$ and $c_2 = 4$, then $Z_{2,1} = R_2 \cup Y_3 = \{C, D, E\}$ and $Z_{2,2} = Y_2$.

Lemma 13. *For all $i \in I$, the following holds in the action graph G: $\rho(Y_i) \subseteq X_i$.*

The fact that $D[X_i]$ contains at least as much information as $D[\rho(Y_i)]$, together with Lemma 10, implies that the characteristic function $\text{ch}(D[Y_i \cup \nu(Y_i)]) = (D[X_i], \#D[Y_i], \text{WCU}(D[Y_i], \Gamma'), \text{BEU}(D[Y_i], \Gamma'))$ is equilibrium-preserving. This is the characteristic function we use. We adapt our dynamic programming algorithm in the previous section so that $\{R_i | i \in I\}$ is the initial partition of S, and the order in which the partitions are combined is "guided" by the tree decomposition, from the leafs to the root.

1. For each R_i, compute A_{R_i}. This can be done by enumerating all possible configurations $D[X_i]$ and keeping ones that constitutes a pure equilibrium of the restricted game on R_i.
2. Initialize the set $Done \subseteq S$ to contain the leaves of the tree T.
3. While $\exists i \in I \setminus Done$ such that $\{j | j$ is a child of $v\} \subseteq Done$:
   (a) Let $A_{Z_{i,0}} := A_{R_i}$
   (b) Let $c_1, \ldots, c_m$ be the children of i.

(c) For $j = 1$ to m , let

$$A_{Z_{i,j}} := \{(D[X_i], B + B', \min\{U_c, U'_c\}, \max\{U_e, U'_e\}) \mid (D[X_i], B, U_c, U_e) \in A_{Z_{i,j-1}}, (D'[X_{c_j}], B', U'_c, U'_e) \in A_{Y_{c_j}}, D[X_i] \text{ and } D'[X_{c_j}] \text{ are consistent}, B + B' \leq n, B + B' = n \text{ if } (i = r \text{ and } j = m) U_c \geq U'_e, U'_c \geq U_e\}$$

(d) $A_{Y_i} := A_{Z_{i,m}}$

(e) Add i to $Done$.

4. Return TRUE iff A_{Y_r} is nonempty.

In each iteration of step 3 of the algorithm, we combine characteristics from restricted games on R_i and $Y_{c_1}, \ldots, Y_{c_m}$ to form a new set of characteristics on the restricted game on Y_i. For the tree decomposition in Figure 4 with node 1 being the root, our algorithm would start from the leaves 3 and 4, then compute $A_{Z_{2,1}} = A_{R_2 \cup Y_3} = A_{\{C,D,E\}}$ by combining A_{R_2} and A_{R_3}, then compute $A_{Y_2} = A_{\{C,D,E,F,G\}}$ by combining $A_{Z_{2,1}}$ and A_{R_4}, and finally compute A_{Y_1} by combining A_{R_1} and A_{Y_2}.

Theorem 14. *Deciding the existence of pure equilibrium in a symmetric AGG with bounded treewidth is in P.*

Proof. Suppose the treewidth of the AGG is bounded by a constant w. Then a tree decomposition of the action graph having width at most w can be constructed in time exponential only in w, i.e. polynomial time (see e.g. (Kloks 1994)). Daskalakis & Papadimitriou (2006) showed that given such a tree decomposition, we can construct a tree decomposition of the primal graph G' having width at most $(w + 1)\mathcal{I} - 1$ in polynomial time.

It is straightforward to check that given a tree decomposition of G', our algorithm above correctly computes A_{Y_i} by applying Lemma 11. Since $Y_r \equiv S$, the algorithm correctly determines the existence of pure equilibrium in Γ. The running time of the algorithm is polynomial in the size of the A_{Y_i}'s. The size of A_{Y_i} is bounded by $n||\Gamma||^2|\Delta[X_i]|$. Since the tree decomposition has width at most $(w + 1)\mathcal{I} - 1$, $|\Delta[X_i]| \leq \binom{n+(w+1)\mathcal{I}}{(w+1)\mathcal{I}}$. The latter is the number of ordered combinatorial compositions of n into $(w+1)\mathcal{I}+1$ nonnegative integers. An equivalent way of counting this number is as follows:

1. break n into $w + 1$ nonnegative integers,
2. then break each of the first w integers into $\mathcal{I}$ nonnegative parts, and the last one into $\mathcal{I} + 1$ nonnegative parts.

There are $\binom{n+w}{w}$ different ways of carrying out step 1. Since each integer in step 2 is at most n, there are at most $\binom{n+\mathcal{I}}{\mathcal{I}}$ ways of breaking each integer. Therefore $\binom{n+(w+1)\mathcal{I}}{(w+1)\mathcal{I}} \leq \binom{n+w}{w}\binom{n+\mathcal{I}}{\mathcal{I}}^{w+1}$. Since w is a constant, this is polynomial in $||\Gamma||$. Hence our algorithm runs in polynomial time. □

Road games (Example 1) have treewidth 2 for all m. Thus by Theorem 14 the existence of pure equilibria can be determined in polynomial time for these games.

Finding All Pure Equilibria

So far we have focused on the problem of deciding the existence of pure equilibria. Our dynamic programming approach can also be used to find these equilibria if they exist. After the bottom-up pass of the tree decomposition as discussed above, a top-down pass would then make sure that each A_{X_i} contains exactly the set of extendable partial solutions. Although the number of pure equilibria of an AGG could be exponential in the representation size $||\Gamma||$, the resulting set of A_{X_i} along with the tree decomposition constitutes a *succinct description* (Daskalakis & Papadimitriou 2006) of the set of pure equilibria of the game. Given a symmetric AGG with bounded treewidth, such a succinct description can be computed in polynomial time. We omit detailed discussion due to space constraints.

Conclusions and Future Work

In this paper we analyzed the problem of computing pure Nash equilibria in AGGs. We proposed a dynamic programming algorithm and showed that for symmetric AGGs with bounded treewidth, our algorithm determines the existence of pure Nash equilibria in polynomial time.

Our approach for symmetric AGGs can be extended to general AGGs. Our approach can also be extended to the computation of the socially optimal equilibrium if one exists, as well as the computation of related solution concepts such as pure-strategy ϵ-Nash equilibrium and strict equilibrium. We will discuss these topics in detail in a future paper.

References

Bhat, N., and Leyton-Brown, K. 2004. Computing Nash equilibria of action-graph games. In *UAI*.

Brandt, F.; Fischer, F.; and Holzer, M. 2007. Symmetries and the complexity of pure Nash equilibrium. In *STACS*.

Chen, X., and Deng, X. 2006. Settling the complexity of 2-player Nash-equilibrium. In *FOCS*.

Daskalakis, C., and Papadimitriou, C. 2006. Computing pure Nash equilibria via Markov random fields. In *ACM-EC*.

Gottlob, G.; Greco, G.; and Scarcello, F. 2003. Pure Nash equilibria: Hard and easy games. In *TARK*.

Ieong, S.; McGrew, R.; Nudelman, E.; Shoham, Y.; and Sun, Q. 2005. Fast and compact: A simple class of congestion games. In *AAAI*.

Jiang, A. X., and Leyton-Brown, K. 2006. A polynomial-time algorithm for Action-Graph Games. In *AAAI*.

Kearns, M.; Littman, M.; and Singh, S. 2001. Graphical models for game theory. In *UAI*.

Kloks, T. 1994. *Treewidth: Computations and Approximations*. Berlin: Springer-Verlag.

Nash, J. 1951. Non-cooperative games. *Annals of Mathematics* 54:286–295.

Robertson, N., and Seymour, P. 1986. Algorithmic aspects of tree-width. *J. Algorithms* 7:309–322.