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

INFORMS: October 14, 2008

Outline

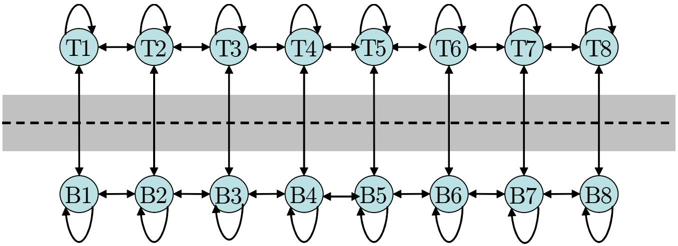
1. Action Graph Games
2. Computing Pure Nash Equilibria
3. Computing Pure Equilibira in Symmetric AGGs
4. Algorithm
5. Conclusions

Outline

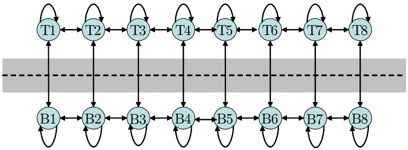
1. Action Graph Games
2. Computing Pure Nash Equilibria
3. Computing Pure Equilibira in Symmetric AGGs
4. Algorithm
5. Conclusions

Example: Location Game

- each of n agents wants to open a business
- actions: choosing locations
- utility: depends on
  - the location chosen
  - number of agents choosing the same location
  - numbers of agents choosing each of the adjacent locations



Game on a graph



- This can be modeled as a game played on a directed graph:
  - each player has a token to put on one of the nodes;
  - each player's utility depends on:
    - the node chosen
    - configuration of tokens over neighboring nodes
- Action Graph Games (Bhat & Leyton-Brown 2004, Jiang & Leyton-Brown 2006)
  - fully expressive, compact representation of games
  - exploits anonymity, context specific independence

Definitions

Definition (action graph)

An action graph is a tuple $(\mathcal{A}, E)$, where $\mathcal{A}$ is a set of nodes corresponding to *distinct actions* and E is a set of directed edges.

- Each agent i's set of available actions: $A_i \subseteq \mathcal{A}$
- Neighborhood of node α: $\nu(\alpha) \equiv \{\alpha' \in \mathcal{A} | (\alpha', \alpha) \in E\}$

Definitions

Definition (action graph)

An **action graph** is a tuple $(\mathcal{A}, E)$, where $\mathcal{A}$ is a set of nodes corresponding to *distinct actions* and E is a set of directed edges.

- Each agent i's set of available actions: $A_i \subseteq \mathcal{A}$
- Neighborhood of node α: $\nu(\alpha) \equiv \{\alpha' \in \mathcal{A} | (\alpha', \alpha) \in E\}$

Definition (configuration)

A **configuration** c is an $|\mathcal{A}|$-tuple of integers $(c[\alpha])_{\alpha \in \mathcal{A}}$. $c[\alpha]$ is the number of agents who chose the action $\alpha \in \mathcal{A}$. For a subset of actions $X \subset \mathcal{A}$, let $c[X]$ denote the restriction of c to X. Let $C[X]$ denote the set of restricted configurations over X.

Action Graph Games

Definition (Action Graph Game (AGG))

An action graph game Γ is a tuple $\langle N, (A_i)_{i\in N}, G, u\rangle$ where

- N is the set of agents
- A_i is agent i's set of actions
- $G = (\mathcal{A}, E)$ is the action graph, where $\mathcal{A} = \bigcup_{i\in N} A_i$ is the set of distinct actions
- $u = (u^\alpha)_{\alpha\in\mathcal{A}}$, where $u^\alpha : C[\nu(\alpha)] \mapsto \mathbb{R}$

Action Graph Games

Definition (Action Graph Game (AGG))

An action graph game Γ is a tuple $\langle N, (A_i)_{i\in N}, G, u\rangle$ where

- N is the set of agents
- A_i is agent i's set of actions
- $G = (\mathcal{A}, E)$ is the action graph, where $\mathcal{A} = \bigcup_{i\in N} A_i$ is the set of distinct actions
- $u = (u^\alpha)_{\alpha\in\mathcal{A}}$, where $u^\alpha : C[\nu(\alpha)] \mapsto \mathbb{R}$

Definition (symmetric AGG)

An AGG is symmetric if all players have identical action sets, i.e. if $A_i = \mathcal{A}$ for all i.

AGG Properties

- AGGs are fully expressive
- Symmetric AGGs can represent arbitrary symmetric games
- Representation size $\|\Gamma\|$ is polynomial if the in-degree $\mathcal{I}$ of G is bounded by a constant
- Any graphical game (Kearns, Littman & Singh 2001) can be encoded as an AGG of the same space complexity.
- AGG can be exponentially smaller than the equivalent graphical game & normal form representations.

Outline

1. Action Graph Games
2. Computing Pure Nash Equilibria
3. Computing Pure Equilibira in Symmetric AGGs
4. Algorithm
5. Conclusions

Pure Nash Equilibria

Action profile: $\mathbf{a} = (a_1, \ldots, a_n)$

Definition (pure Nash equilibrium)

An action profile $\mathbf{a}$ is a pure Nash equilibrium of the game Γ if for all $i \in N$, a_i is a best response to a_{-i} (i.e. for all $a_i' \in A_i$, $u_i(a_i, a_{-i}) \geq u_i(a_i', a_{-i})$).

- not guaranteed to exist
- often more interesting than mixed Nash equilibria

Complexity of Finding Pure Equilibria

Checking every action profile:

- linear time in normal form size
- worst-case exponential time in AGG size

Complexity of Finding Pure Equilibria

Checking every action profile:

- linear time in normal form size
- worst-case exponential time in AGG size

Consider the restriction to symmetric AGGs.

Theorem (Conitzer, personal communication; also proven independently in (Daskalakis *et al.* 2008))

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.

Our Contribution

We provide an algorithm that is tractable for symmetric AGGs with bounded treewidth

- the algorithm can also be applied to other settings

Specifically, we propose a dynamic programming approach:

- partition action graph into subgraphs (via tree decomposition)
- construct equilibria of the game from equilibria of games played on subgraphs

Our Contribution

We provide an algorithm that is tractable for symmetric AGGs with bounded treewidth

- the algorithm can also be applied to other settings

Specifically, we propose a dynamic programming approach:

- partition action graph into subgraphs (via tree decomposition)
- construct equilibria of the game from equilibria of games played on subgraphs

Related Work:

- finding pure equilibria in graphical games
  - (Gottlob, Greco, & Scarcello 2003) and (Daskalakis & Papadimitriou 2006)
- finding pure equilibria in simple congestion games
  - (Ieong, McGrew, Nudelman, Shoham, & Sun 2005)

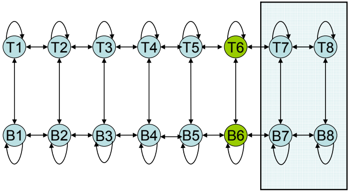
Outline

1. Action Graph Games
2. Computing Pure Nash Equilibria
3. **Computing Pure Equilibira in Symmetric AGGs**
4. Algorithm
5. Conclusions

Restricted Game

To derive an algorithm that builds up from partial solutions, we must define the concept of a restricted game

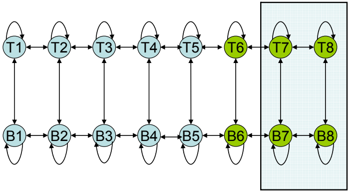
- game played by a subset of players: $n' \leq n$
- actions restricted to $R \subseteq \mathcal{A}$
- utility functions same as in original AGG
  - need to specify configuration of neighboring nodes not in R



- *restricted game* $\Gamma(n', R, c[\nu(R)])$

Partial Solution

We want to use equilibria of restricted games as building blocks



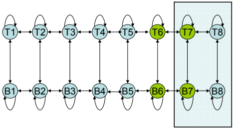
Definition (partial solution)

A partial solution on a restricted game $\Gamma(n', X, c[\nu(X)])$ is a configuration $c[X \cup \nu(X)]$ such that $c[X]$ is a pure NE of Γ.

Extending partial solutions

- Problem: combining two partial solutions on two non-overlapping restricted games does not necessarily produce an equilibrium of the combined game
  - configurations may be inconsistent, or
  - player might profitably deviate from playing in one restricted game to another
- keeping all partial solutions: impractical as sizes of restricted games grow
- we would like sufficient statistics that summarize partial solutions as compactly as possible

Sufficient statistic



Sufficient Statistic: a tuple consisting of

1. configuration over
   - outside neighbours: $\nu(X)$
   - inside nodes that are neighbors of outside nodes: $\nu(\overline{X})$
2. number of agents playing in X
3. U_w, utility of the worst-off player in $X \setminus \nu(\overline{X})$.
4. U_b, best utility an outside player can get by playing in $X \setminus \nu(\overline{X})$.

Number of distinct tuples: polynomial for action graphs of bounded treewidth

Combining sufficient statistics

Given two sets of such tuples, summarizing partial solutions on $X, Y \subset \mathcal{A}$, we can compute the set of sufficient statistics for the combined restricted game $X \cup Y$

- start with all consistent configurations
  - analogous to database join of the two sets of tuples
- discard those with profitable $X \to Y$ deviations (& vice versa)
  - easy: discard when U_w from X is worse than U_b from Y
  - trickier: checking deviations from $X \cap \nu(Y)$ to $\nu(\overline{Y})$
    - utilities in $\nu(\overline{Y})$ change when $c[\nu(Y)]$ changes, so checking these deviations is more costly
    - solution: augment our sufficient statistics to keep track of the configuration of the neighborhood of $\nu(\overline{Y})$, in order to compute these utilities on the fly
    - luckily, for graphs of bounded treewidth, this implies storing a small amount of additional information
  - overall: all profitable deviations can be discarded efficiently

Outline

1. Action Graph Games
2. Computing Pure Nash Equilibria
3. Computing Pure Equilibira in Symmetric AGGs
4. Algorithm
5. Conclusions

Algorithm

1. Construct the primal graph of the action graph.
2. Build a tree decomposition of this primal graph.
3. Partition the AGG according to the tree decomposition.
4. Find all sufficient statistics[1] corresponding to partial solutions of games restricted to each partition.
5. Working up the tree, combine adjacent nodes together.
6. When root is reached, return whether the game has a PSNE.

[1] Augment sufficient statistics to include configurations over additional actions that belong to the decomposition's tree node that is closest to the root.

Algorithm

1. Construct the primal graph of the action graph.
2. Build a tree decomposition of this primal graph.
3. Partition the AGG according to the tree decomposition.
4. Find all sufficient statistics[1] corresponding to partial solutions of games restricted to each partition.
5. Working up the tree, combine adjacent nodes together.
6. When root is reached, return whether the game has a PSNE.

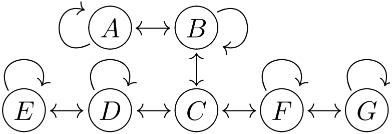
Theorem

For symmetric AGGs with bounded treewidth, our algorithm determines existence of pure Nash equilibria in polynomial time.

Recover a PSNE from the SS's: downwards pass on the tree

[1] Augment sufficient statistics to include configurations over additional actions that belong to the decomposition's tree node that is closest to the root.

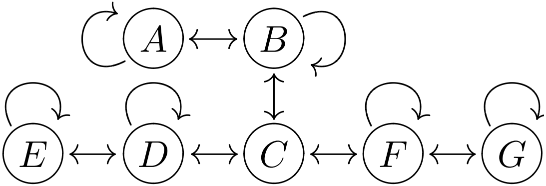
An Example



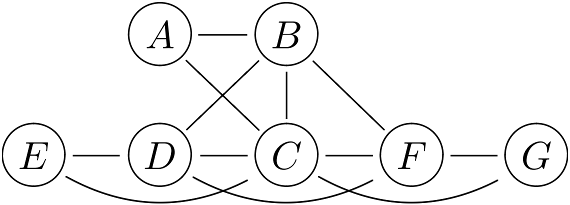
- Two players
- Utility functions:
  - start with payoff of 0
  - $+1$ reward if playing action F or D
  - -2 penalty if another player selected an action with an incoming edge
    - For C, this means a neighboring action (since C does not have a self-edge)
    - Otherwise, this means the same or a neighboring action
- Pure Nash equilibria:
  - One player chooses D, the other chooses F
  - Both players choose C

1. Construct Primal Graph

Action graph:



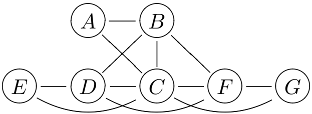
Primal graph: make each neighborhood a clique

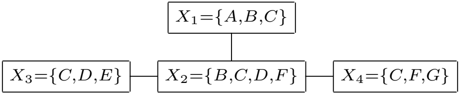


2. Construct Tree Decomposition

A tree where each node is labeled with one or more nodes from the primal graph, where

- every label is used at least once
- for every edge in the primal graph from α_1 to α_2, there is a node in the tree labeled with both α_1 and α_2
- if a label occurs in two nodes x_1, x_2 in the tree, it also occurs on all paths between x_1 and x_2.



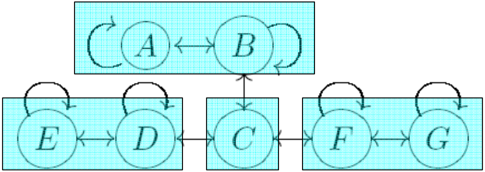


If treewidth of the AGG is bounded by a constant, the primal graph's tree decomposition can be computed in polynomial time.

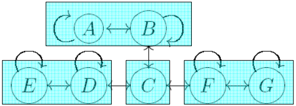
3. Partition the AGG According to the Tree Decomposition

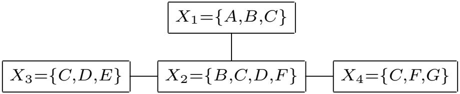
By construction: for each node α in the action graph, there always exists a tree node in the decomposition of the primal graph that contains α and its neighbors in the action graph.

The tree decomposition therefore induces the following partition on the AGG:



4. Compute Sufficient Statistics for Partial Solutions on Each Partition





For restricted game on $\{C\}$:

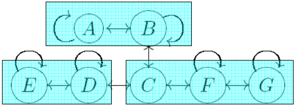
n'	$c[B,C,D,F]$	$U_w(\emptyset)$	$U_b(\emptyset)$
0	0,0,0,0	∞	$-\infty$
0	1,0,0,0	∞	$-\infty$
$\cdots$	$\cdots$	∞	$-\infty$
1	0,1,0,0	∞	$-\infty$
1	1,1,0,0	∞	$-\infty$
$\cdots$	$\cdots$	∞	$-\infty$
2	0,2,0,0	∞	$-\infty$

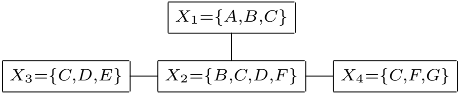
For restricted game on $\{F,G\}$:

n'	$c[C,F,G]$	$U_w(G)$	$U_b(G)$
0	0,0,0	∞	0
0	1,0,0	∞	0
0	2,0,0	∞	0
1	0,1,0	∞	-2
1	1,0,1	0	-2
2	0,1,1	-2	$-\infty$

5. Working up the Tree, Combine Restricted Games

Combine restricted games in bottom-up order: from leaves to root.



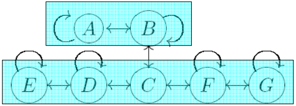


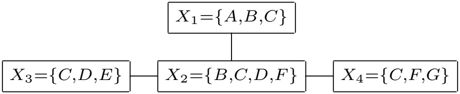
Combine {C} and {F,G} to create table for restricted game on {C,F,G}:

n'	$c[B,C,D,F]$	$U_w(G)$	$U_b(G)$
0	0,0,0,0	∞	0
0	1,0,0,0	∞	0
...	...	∞	0
1	0,0,0,1	∞	-2
1	1,0,0,1	∞	-2
1	0,0,1,1	∞	-2
2	0,1,0,0	0	$-\infty$
2	0,2,0,0	∞	$-\infty$

5. Working up the Tree, Combine Restricted Games

Combine restricted games in bottom-up order: from leaves to root.



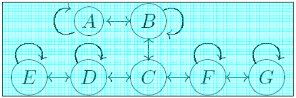


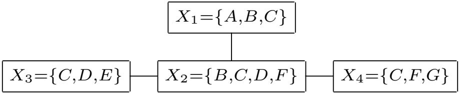
Combine {D,E} and {C,F,G} to create table for {C,D,E,F,G}:

n'	$c[B,C,D,F]$	$U_w(E,G)$	$U_b(E,G)$
0	0,0,0,0	∞	0
0	1,0,0,0	∞	0
0	2,0,0,0	∞	0
1	0,0,1,0	∞	0
1	1,0,1,0	∞	0
1	0,0,0,1	∞	0
1	1,0,0,1	∞	0
2	0,0,1,1	∞	$-\infty$
2	0,2,0,0	∞	$-\infty$

5. Working up the Tree, Combine Restricted Games

Combine restricted games in bottom-up order: from leaves to root.





Combine {A,B} and {C,D,E,F,G}:

n'	$c[A,B,C]$	$U_w(D,E,F,G)$	$U_b(D,E,F,G)$
2	0,0,0	1	$-\infty$
2	0,0,2	∞	$-\infty$

6. Top-Down Pass to Compute PNSE

n'	$c[A,B,C]$	$U_w(D,E,F,G)$	$U_b(D,E,F,G)$
2	0,0,0	1	$-\infty$
2	0,0,2	∞	$-\infty$

To compute a PSNE, start from the root and work down. At each node, pick a row from the table of sufficient statistics that is consistent with earlier picks.

- If we start with row 1, we select an equilibrium in which one player chooses D, one player chooses F
- If we start with row 2, we select an equilibrium in which both players choose C

Outline

1. Action Graph Games
2. Computing Pure Nash Equilibria
3. Computing Pure Equilibira in Symmetric AGGs
4. Algorithm
5. Conclusions

Conclusions & Beyond Symmetric AGGs

- dynamic programming approach for computing pure equilibria in AGGs
- poly-time algorithm for symmetric AGGs with bounded treewidth
- our approach can be extended to general AGGs
  - different set of sufficient statistics
    - when the game is k-symmetric (i.e. has k distinct action sets), use k-configuration (k-tuple of configurations, one for each equivalence class of players), and similarly use k-tuples of U_w, U_b
    - for subgraphs in which only k' of the k classes of players participate, only need to keep track of the sufficient statistics for those k' classes.
  - related algorithms for graphical games (Daskalakis & Papadimitriou 2006) and simple congestion games (Ieong et al 2005) become special cases of our approach