Game Theory

Lecture 13

Graphical Games

Introduction

- Representing multiplayer games with large player populations in the normal form is undesirable for both practical and conceptual reasons.
  - On the practical side, the number of parameters that must be specified grows exponentially with the size of the population.
  - On the conceptual side, the normal form may fail to capture structure that is present in the strategic interaction, and which can aid understanding of the game and computation of its equilibria.
- For this reason, there have been many proposals for parametric multiplayer game representations that are more succinct than the normal form, and attempt to model naturally arising structural properties.
  - Examples include ***congestion*** and ***potential*** games and related models.

Graphical Games: Introduction

- Graphical games are a representation of multiplayer games meant to capture and exploit locality or sparsity of direct influences.
- They are most appropriate for large population games in which the payoffs of each player are determined by the actions of only a small subpopulation.
- As such, they form a natural counterpart to earlier parametric models.
  - Whereas ***congestion games*** and related models implicitly assume a ***large number of weak influences*** on each player, graphical games are suitable when there is a small number of strong influences.

Graphical Games: Introduction

- A graphical game is described at the first level by an undirected graph G in which players are identified with vertices.
  - The payoff to player i is a function only of the actions of i and its neighbors, rather than the actions of the entire population.
  - In the many natural settings where such local neighborhoods are much smaller than the overall population size, the benefits of this parametric specification over the normal form are already considerable.
- In this lecture, we examine the computation of Nash equilibria in graphical games in which the underlying graph G is a tree.
  - Here we will discuss a natural *two-pass algorithm* for computing Nash equilibria requiring only the local exchange of "*conditional equilibrium*" information over the edges of G.

Basic Notations

we use a_i to denote the action chosen by player i.

For simplicity we will assume a binary action space, so $a_i \in \{0, 1\}$. (The generalization of the results examined here to the multiaction setting is straightforward.)

The payoffs to player i are given by a table or matrix M_i, indexed by the joint action $\vec{a} \in \{0, 1\}^n$.

The value $M_i(\vec{a})$, which we assume without loss of generality to lie in the interval [0, 1], is the payoff to player i resulting from the joint action $\vec{a}$.

The actions 0 and 1 are the *pure strategies* of each player, while a *mixed* strategy for player i is given by $p_i \in [0, 1]$ that the player will play 0.

For any joint mixed strategy, given by a product distribution $\vec{p}$, we define the expected payoff to player i as $M_i(\vec{p}) = \mathbf{E}_{\vec{a} \sim \vec{p}}[M_i(\vec{a})]$, where $\vec{a} \sim \vec{p}$ indicates each a_j is 0 with probability p_j and 1 with probability $1 - p_j$ independently.

Basic Notations

We use $\vec{p}[i : p_i']$ to denote the vector (product distribution) which is the same as $\vec{p}$ except in the ith component, where the value has been changed to p_i'.

A *Nash equilibrium (NE)* for the game is a mixed strategy $\vec{p}$ such that for any player i, and for any value $p_i' \in [0, 1]$,

$$M_i(\vec{p}) \geq M_i(\vec{p}[i : p_i']).$$

(We say that p_i is a *best response* to the rest of $\vec{p}$.)

An ϵ-*Nash equilibrium* is a mixed strategy $\vec{p}$ such that for any player i, and for any value $p_i' \in [0, 1]$, $M_i(\vec{p}) + \epsilon \geq M_i(\vec{p}[i : p_i'])$.

(We say that p_i is an ϵ-*best response* to the rest of $\vec{p}$.)

Graphical Game: Definition

In a *graphical game*, each player i is represented by a vertex in an undirected graph G.

We use $N(i) \subseteq \{1, \ldots, n\}$ to denote the *neighborhood* of player i in G – that is, those vertices j such that the edge (i, j) appears in G.

By convention $N(i)$ always includes i itself as well.

If $\vec{a}$ is a joint action, we use $\vec{a}^{\,i}$ to denote the projection of $\vec{a}$ onto just the players in $N(i)$.

Definition A *graphical game* is a pair $(G, \mathcal{M})$, where G is an undirected graph over the vertices $\{1, \ldots, n\}$, and $\mathcal{M}$ is a set of n *local game matrices*.
For any joint action $\vec{a}$, the local game matrix $M_i \in \mathcal{M}$ specifies the payoff $M_i(\vec{a}^{\,i})$ for player i, which depends only on the actions taken by the players in $N(i)$.

Graphical Game: Remarks

- Graphical games are a (potentially) more compact way of representing games than standard normal form.
  - Rather than requiring a number of parameters that is exponential in the number of players n, a graphical game requires a number of parameters that is exponential only in the size d of the largest local neighborhood.
  - Thus if $d \ll n$ – that is, the number of direct influences on any player is much smaller than the overall population size – the graphical game representation is dramatically smaller than the normal form.
- It is also worth noting that although the payoffs to player i are determined only by the actions of the players in $N(i)$, equilibrium still requires global coordination across the player population:
  - if player i is connected to player j who is in turn connected to player k, then i and k indirectly influence each other via their mutual influence on the payoff of j.
  - How local influences propagate to determine global equilibrium outcomes is one of the computational challenges posed by graphical games.

Computing Nash Equilibria in Tree Graphical Games

- We describe the most basic algorithm exploiting the advantages of graphical game representation for the purposes of equilibrium computation.

- We assume that the underlying graph G is a tree.

  - While obviously a strong restriction on the topology, we shall see that this case already presents nontrivial computational challenges.

  - We first describe the algorithm **TreeNash** at a high level, leaving certain important implementation details unspecified, because it is conceptually advantageous to do so.

  - We then describe one instantiation of the missing details, yielding an algorithm that provably computes approximations of all equilibria.

Some notation and concepts

- We begin with some notation and concepts needed for the description of **TreeNash**.
- In order to distinguish ***parents*** from ***children*** in the tree, it will be convenient to treat players/vertices symbolically (such as *U*, V, and *W*) rather than by integer indices, so
  - ➢ we use M_V to denote the ***local game matrix*** for the player identified with player/vertex *V*.
  - ➢ We use capital letters to denote vertex/players to distinguish them from their chosen actions, for which we shall use lower case.
- If *G* is a tree, we choose an arbitrary vertex as the ***root*** (which we visualize as being at the bottom, with the ***leaves*** at the top).
  - ➢ Any vertex on the path from a vertex *V* to the root will be called ***downstream*** from *V*, and
  - ➢ any vertex on a path from *V* to any leaf will be called ***upstream*** from *V* .
  - ➢ Thus, each vertex other than the root has exactly one downstream neighbor (or child), and perhaps many upstream neighbors (or parents).
  - ➢ We use $UP_G(V)$ to denote the set of all vertices in *G* that are upstream from *V* , including *V* by definition.

Some notation and concepts

Suppose that V is the child of U in G.

We let G^U denote the subgraph induced by the vertices in $\mathrm{UP}_G(U)$ (that is, the subtree of G rooted at U.)

If $v \in [0, 1]$ is a mixed strategy for player (vertex) V, $\mathcal{M}^U_{V=v}$ will denote the subset of payoff matrices in $\mathcal{M}$ corresponding to the vertices in $\mathrm{UP}_G(U)$, with the modification that the game matrix M_U is collapsed by one index by fixing $V = v$.

We can think of an NE for the graphical game $(G^U, \mathcal{M}^U_{V=v})$ as a *conditional* equilibrium "upstream" from U (inclusive) – that is, an equilibrium for G^U *given* that V plays v.

- Here we are simply exploiting the fact that since G is a tree, fixing a mixed strategy v for the play of V isolates G^U from the rest of G.

TreeNash Algorithm

Now suppose that vertex V has k parents $U_1, \ldots, U_k$, and the single child W.

We now describe the data structures sent from each U_i to V, and in turn from V to W, on the downstream pass of **TreeNash**.

- Each parent U_i will send to V a binary-valued "table" $T(v, u_i)$.
  - The table is indexed by the *continuum* of possible values for the mixed strategies $v \in [0, 1]$ of V and $u_i \in [0, 1]$ of U_i, $i = 1, \ldots, k$.
  - The semantics of this table will be as follows: for any pair (v, u_i), $T(v, u_i)$ will be 1 if and only if there exists an NE for $(G^{U_i}, \mathcal{M}^{U_i}_{V=v})$ in which $U_i = u_i$.

TreeNash Algorithm: Downstream Pass

- Since v and u_i are continuous variables, it is not obvious that the table $T(v,u_i)$ can be represented compactly, or even finitely, for arbitrary vertices in a tree. For now we will simply assume a finite representation, and shortly discuss how this assumption can be met.
- The initialization of the ***downstream pass*** of the algorithm begins at the leaves of the tree, where the computation of the tables is straightforward:
  - If U is a leaf and V its only child, then $T(v, u) = 1$ if and only if $U = u$ is a best response to $V = v$ (Step (ii) (c) of the Algorithm).
- Assuming for induction that each U_i sends the table $T(v, u_i)$ to V, we now describe how V can compute the table $T(w, v)$ to pass to its child W (Step (ii) (d)2 of the algorithm).
  - For each pair (w, v), $T(w, v)$ is set to 1 if and only if there exists a vector of mixed strategies $\vec{u} = (u_1, \ldots, u_k)$ (called a *witness*) for the parents $\vec{U} = (U_1, \ldots, U_k)$ of V such that

    (i) $T(v, u_i) = 1$ for all $1 \leq i \leq k$; and

    (ii) $V = v$ is a best response to $\vec{U} = \vec{u}$, $W = w$.
- Note that there may be more than one witness for $T(w,v) = 1$.

TreeNash Algorithm: Downstream Pass

- In addition to computing the value $T(w,v)$ on the downstream pass of the algorithm, V will also keep a list of the witnesses $\vec{u}$ for each pair (w,v) for which $T(w,v) = 1$ (Step ii(d)2 of the Algorithm).
  - ➢ These witness lists will be used on the upstream pass.
- To see that the semantics of the tables are preserved by the computation just described, suppose that this computation yields $T(w,v) = 1$ for some pair (w,v), and let $\vec{u}$ be a witness for $T(w,v) = 1$
  - ➢ The fact that $T(v,u_i) = 1$ for all i ensures by induction that:
    - if V plays v, there are upstream NE in which each $U_i = u_i$.
  - ➢ Furthermore, v is a best response to the local settings $U_1 = u_1$, . . . , $U_k = u_k, W = w$.
  - ➢ Therefore, we are in equilibrium upstream from V.
  - ➢ On the other hand, if $T(w,v) = 0$, it is easy to see there can be no equilibrium in which $W = w$, $V = v$. Note that the existence of an NE guarantees that $T(w,v) = 1$ for at least one (w,v) pair.
- The downstream pass of the algorithm terminates at the root Z, which receives tables $T(z,y_i)$ from each parent Y_i .
- Z simply computes a one-dimensional table $T(z)$ such that :
  - ➢ $T(z) = 1$ if and only if for some witness $\vec{y}$, $T(z,yi) = 1$ for all i, and z is a best response to $\vec{y}$.

TreeNash Algorithm: Upstream Pass

- The ***upstream pass*** begins by Z which
  - chooses any z for which $T(z) = 1$, and
  - chooses any witness $(y_1, \ldots, y_k)$ to $T(z) = 1$, and then
    - passing both z and y_i to each parent Y_i.
    - The interpretation is that Z will play z, and is "instructing" Y_i to play y_i .
  - Inductively, if a vertex V receives a value v to play from its downstream neighbor W, and the value w that W will play, then it must be that $T(w,v) = 1$.
    - So V chooses a witness $\vec{u}$ to $T(w,v) = 1$, and passes each parent U_i their value u_i as well as v (Step (iii) of the Algorithm).
    - Note that the semantics of $T(w,v) = 1$ ensure that $V = v$ is a best response to $\vec{U} = \vec{u}, W = w$.

TreeNash Algorithm: Upstream Pass

- We have left the choices of each witness in the upstream pass unspecified or nondeterministic to emphasize that the tables and witness lists computed represent *all* the NE.
  - ➢ The upstream pass can be specialized to find a number of specific NE of interest, including
    - player optimum (NE maximizing expected reward to a chosen player),
    - social optimum (NE maximizing total expected reward, summed over all players), and
    - welfare optimum (NE maximizing expected reward to the player whose expected reward is smallest).

- Modulo the important details regarding the representation of the tables $T(w,v)$, which we discuss next, the arguments provided above establish the following formal result.

Theorem *Let* (G,M) *be any graphical game in which* G *is a tree. Algorithm* **TreeNash** *computes a Nash equilibrium for* (G,M). *Furthermore, the tables and witness lists computed by the algorithm represent all Nash equilibria of* (G,M).

Algorithm **TreeNash**
Inputs: Graphical game $(G, \mathcal{M})$ in which G is a tree.
Output: A Nash equilibrium for $(G, \mathcal{M})$.

(i) Compute a depth-first ordering of the vertices of G.

(ii) **(Downstream Pass)** For each vertex V in depth-first order:

(a) Let vertex W be the child of V (or nil if V is the root).

(b) For all $w, v \in [0, 1]$, initialize $T(w, v)$ to be 0 and the witness list for $T(w, v)$ to be empty.

(c) If V is a leaf (base case):

1. For all $w, v \in [0, 1]$, set $T(w, v)$ to be 1 if and only if $V = v$ is a best response to $W = w$ (as determined by the local game matrix M_V).

(d) Else (inductive case, V is an internal vertex):

1. Let $\vec{U} = (U_1, \ldots, U_k)$ be the parents of V; let $T(v, u_i)$ be the table passed from U_i to V on the downstream pass.

2. For all $w, v \in [0, 1]$ and for all joint mixed strategies $\vec{u} = (u_1, \ldots, u_k)$ for $\vec{U}$: If $V = v$ is a best response to $W = w$, $\vec{U} = \vec{u}$ (as determined by the local game matrix M_V), and $T(v, u_i) = 1$ for $i = 1, \cdots, k$, set $T(w, v)$ to be 1 and add $\vec{u}$ to the witness list for $T(w, v)$.

(e) Pass the table $T(w, v)$ from V to W.

(iii) **(Upstream Pass)** For each vertex V in reverse depth-first ordering (starting at the root):

(a) Let $\vec{U} = (U_1, \ldots, U_k)$ be the parents of V (or the empty list if V is a leaf); let W be the child of V (or nil if V is the root), and (w, v) the values passed from W to V on the upstream pass.

(b) Label V with the value v.

(c) (Non-deterministically) Choose any witness $\vec{u}$ to $T(w, v) = 1$.

(d) For $i = 1, \ldots, k$, pass (v, u_i) from V to U_i.

leaf nodes

upstream

G^{U_i}

U_1 U_i U_K

a parent of V

$T(v, u_i) = 1 \Leftrightarrow$
$\exists\, an\ NE\ for\ (G^{U_i}, \mathcal{M}^{U_i}_{V=v}) \Lambda u_i \in BR_{u_i}(v)$

V

$T(w, v) = 1 \Leftrightarrow$
$v \in BR_V(\vec{u}, w) \Lambda T(v, u_i) = 1$ for all i $= 1, \ldots, K$

Then: $\vec{u} = (u_1, \ldots, u_K)$ is a witness

W

child of V

downstream

Y_1 Y_i $Y_{K'}$

$T(z, y_i) = 1 \Leftrightarrow$
$\exists\, an\ NE\ for\ \left(G^{Y_i}, \mathcal{M}^{Y_i}_{Z=z}\right) \Lambda\, y_i \in BR_{y_i}(z)$

Z

$T(z) = 1 \Leftrightarrow$
$z \in BR_Z(\vec{y}\,) \Lambda T(z, y_i) = 1\ \forall$i $= 1, \ldots, K'$

Then: $\vec{y} = (y_1, \ldots, y_{K'})$ a witness

root

An Approximation Algorithm

- In this section, we sketch one instantiation of the missing details of algorithm **TreeNash** that yields a polynomial-time algorithm for computing *approximate* NE for the tree game (*G*,*M*).
  - The approximation can be made arbitrarily precise with greater computational effort.

Rather than playing an arbitrary mixed strategy in [0, 1], each player will be constrained to play a *discretized* mixed strategy that is a multiple of τ, for some τ to be determined by the analysis.

Thus, player i plays $q_i \in \{0, \tau, 2\tau, \ldots, 1\}$, and the joint strategy $\vec{q}$ falls on the discretized τ-grid $\{0, \tau, 2\tau, \ldots, 1\}^n$.

In algorithm **TreeNash**, this will allow each table $T(v, u)$ (passed from vertex U to child V) to be represented in discretized form as well:

only the $1/\tau^2$ entries corresponding to the possible τ-grid choices for U and V are stored, and all computations of best responses in the algorithm are modified to be approximate best responses.

An Approximation Algorithm

To quantify how the choice of τ will influence the quality of the approximate equilibria found (which in turn will determine the computational efficiency of the approximation algorithm), we appeal to the following lemma.

We note that this result holds for arbitrary graphical games, not only trees.

Lemma *Let G be a graph of maximum degree d, and let $(G, \mathcal{M})$ be a graphical game. Let $\vec{p}$ be a Nash equilibrium for $(G, \mathcal{M})$, and let $\vec{q}$ be the nearest (in L_1 metric) mixed strategy on the τ-grid. Then $\vec{q}$ is a $d\tau$-NE for $(G, \mathcal{M})$.*

- We omit the proof of Lemma, but what matters is that the algorithm remains exponential in *d* simply due to the representational complexity of the local product distributions.
  - The important point is that τ needs to depend only on the local neighborhood size *d*, not the total number of players *n*.

The ApproximateTreeNash Algorithm

- It is now straightforward to describe **ApproximateTreeNash**. This algorithm is identical to algorithm **TreeNash** with the following exceptions:
  - The algorithm now takes an additional input ε.
  - For any vertex U with child V , the table $T(u, v)$ will contain only entries for u and v multiples of τ .
  - All computations of best responses in algorithm **TreeNash** become computations of ε-best responses in algorithm **ApproximateTreeNash**.

For the running time analysis, we simply note that each table has $(1/\tau)^2$ entries and that the computation is dominated by the downstream (Step (ii)(d) of algorithm **TreeNash**).

This requires ranging over all table entries for all k parents, a computation of order $((1/\tau)^2)^k$

By appropriately choosing the value of τ in order to obtain the required ϵ-approximations, we obtain the following theorem.

Theorem *Let $(G, \mathcal{M})$ be a graphical game in which G is a tree with n vertices, and in which every vertex has at most d parents. For any $\epsilon > 0$, let $\tau = O(\epsilon/d)$. Then* **ApproximateTreeNash** *computes an ϵ-Nash equilibrium for $(G, \mathcal{M})$.*