Graphical Games

Part 3: [Dealing with arbitrary topology]

Alternative Formulations and Algorithms
Computing NE can be formulated as a search over an uncountably infinite set.

- Want to find a mixed strategy assignment to each player (out of all possible values for mixed strategies) such that each is (approx.) best response to the others.

- Discretizing each player's mixed strategy space

  \[\text{search over finite space of possible values} \]

  \[\text{[cross-product of finite set of possible values for each player's mixed strategies in the discretization]}\]

- Many ways to perform search

- Nashprop is a particular example: exploits local graph topology.

  \[\text{[local search with backtracking]}\]

- Other alternatives (including "brute-force" search).
Node-merging: Another heuristic approach to handle graphical games with graph of arbitrary topology.

**Idea:** Turn graph into a tree by merging nodes

**Ex.**

![Graph before and after node merging](image)

**Alternative: Hyper-tree**

![Hyper-tree example](image)

**Objective:** Want graph whose size of largest "hyper-neighborhood" is smallest.

[A computationally hard problem!]

**Alg:** Adapt Treenash accordingly to work on tree graph with "hyper-nodes"

**Remarks:**

- "Hyper-table" size is exponential on size of largest pair of neighboring "hyper-nodes"
- Running time is exponential on size of largest "hyper-neighborhood"
Compare and contrast: Nashprop vs. Node-merging

- Some graphs can lead to trees with large size "hyper-neighborhoods" even if node-merging is optimal.

  \[ \Rightarrow \text{large amount of computation/storage can be required from the start} \]

  Once, "downstream" (table) pass performed, (possibly hard)
  "upstream" (assignment) pass easy
  (requires no backtracking!)

- Nashprop works directly on the game graph.
  Each round can be done in poly. time/space/time in game representation size (as long as max. neighborhood size doesn't grow to fast with number of players).

  However, assignment-passing phase can take a large \# of rounds (exponential in \# players).
A function minimization approach (VK'02)

Idea: Apply greedy (coordinate-wise) "hill-climbing" method to minimize a particular "cost" function whose global minima correspond exactly to Nash equilibria.

Ex. of a cost function:
Given a joint mixed strategy, the cost function outputs the total sum of each player's regret (the amount a player can gain for itself by unilaterally deviating from its assigned mixed strategy in the given joint mixed strategy).

Defn of gain function:
For a given joint mixed strategy, the gain with a player is the largest reduction in cost achievable by unilaterally deviating from the strategy assigned to the player in the joint mixed strategy.

Note: changing a player's mixed strategy affects both the players' regret and that of its neighbors.
Alg. sketch:

For each round,

let $i_{\text{max}}$ be player with largest gain given the current joint mixed strategy.

If the gain wrt player $i_{\text{max}}$ is positive,

set player $i_{\text{max}}$ current mixed strategy to be that which maximizes the gain wrt player $i_{\text{max}}$.

Remark: "regrets" are continuous but not differentiable, yet...

- For a graphical game,
  both the gain of each player and the mixed strategies that achieve that gain can be computed efficiently by a linear program whose variables "correspond" to the players mixed strategy and its neighbor's expected payoff.

[Intuition: In a graphical game, a player's mixed strategy "affects" only its expected payoff and that of its neighbors in the game graph].

- Only convergence to local optima guaranteed. So it might not find a NE [i.e., incomplete search].
Computational considerations of "hill-climbing" heuristic.

- it might take a long time to converge (find a local minimum), but...

- each iteration (mixed-strategy update) can be done efficiently (poly. time in model size).

- little space considerations.
Computing NE: Constraint Satisfaction Problem (CSP) formulation

• A CSP

Input

• A set \{P_i\} of \(n\) variables
• A set \{D_i\} of \(n\) domains (possible values for vars.)
• A set \{C_j\} of \(m\) constraints (functions from \(D_1 \times \ldots \times D_n\) to \{0,1\})

Output

• An assignment \(\vec{p}^* = (p_1^*, \ldots, p_n^*)\) to the variables \(\vec{p} = (p_1, \ldots, p_n)\) s.t.

\[\begin{align*}
p_i^* & \in D_i \quad \forall i = 1, \ldots, n \\
C_j(\vec{p}) & = 1 \quad \forall j = 1, \ldots, m
\end{align*}\]

Remarks:

• Often the domains are finite sets
• Often the constraints are over a cross-product of a subset of the domains (as opposed to all the domains); each constraint is a function of only some subset of the vars.
For computing a NE.

\( P_i \) corresponds to player \( i \)'s mixed strategy.

\( D_i \) is the mixed strategy space (set of possible values/mixed strategies that player \( i \) can play).

We have one constraint per player.

\( C_i(P_1, \ldots, P_n) \) corresponds to the best-response condition for player \( i \).

**Remarks:**

- If discretization used, \( D_i \) are finite sets (corresponding to the discretized points).

- If approx. NE, \( C_i \) correspond to approx best-response condition for player \( i \).

**Constraint network:** graphical "representation" for a CSP.

![Constraint network diagram]

Each node corresponds to a variable in the CSP.

Each edge \( \Rightarrow \) a constraint that is a function of (at least) those variables.
Example:

\[ G_6 : \]

\begin{align*}
\text{Vars: } & \{ P_1, \ldots, P_8 \} \\
\text{Domains: } & \{ D_1, \ldots, D_8 \} \\
\text{Constraints: } & \{ C_1, \ldots, C_8 \}
\end{align*}

(c)-best-response constraints
for player 1, etc..

\textit{Constraint network:}

\[ \text{VAR, CSP formulations:} \]

\begin{enumerate}
\item \textbf{Hidden-variable} \hspace{1cm} \textit{[constraints are hidden vars; new CSP has only binary constraints]}
\end{enumerate}

Can be simplified by first merging constraints.

\begin{align*}
\text{Vars: } & \{ P_1, \ldots, P_8, C_1, \ldots, C_8 \} \\
\text{Domains: } & \{ D_1, \ldots, D_8, D'_1, \ldots, D'_8 \} \\
\text{Constraints: } & \{ C_{(i,j)} \}, \ C_j \text{ is function of (at least) } P_i \text{ in orig. CSP} \\
\end{align*}

\[ \text{Ex: } \]

\begin{align*}
\text{Dom}(P_1, P_2) = 1 \iff P_i \text{ and } P'_j \text{ consistent (i.e., } P_i = P'_j) \\
\iff C_2(P_1, P_2, P'_2, P'_1) = 1 \text{ and } P'_1 = P_1
\end{align*}
2. **Dual** [vars. are constraints of orig. CSP; new CSP has only binary constraints]

![Diagram of dual CSP network]

\[ \Downarrow \text{Can be simplified to} \]

(by merging constraints)

\[
\begin{align*}
\text{Vars: } & \{ P_{dual}^{1234}, P_{dual}^{235}, P_{dual}^{245}, \ldots \} \\
\text{Domains: } & \{ D_{dual}^{1234}, D_{dual}^{235}, \ldots \} \\
\text{Constraints: } & \{ C_{dual}^{(1234,235)}, \ldots \} \\
\text{Ex. Denote } & P_{1234}^{dual} = (p_1, p_2, p_3, p_4), \quad P_{235}^{dual} = (p_2, p_3, p_5) \\
C_{(1234,235)}(P_{1234}^{dual}, P_{235}^{dual}) & = 1 \text{ iff } P_{234}^{dual} \text{ and } P_{235}^{dual} \text{ consistent (i.e.,} \\
\text{[assignment corresponding to intersection are} & \text{consistent]} \\
\text{iff } & C_1(p_1, p_2) = 1, C_2(p_2, p_3) = 1, C_3(p_2, p_3) = 1 \text{ and} \\
f_1 = f_2^{n}, f_3 = f_3^{n} \]
Clusters of variables:

Ex., a pair of variables \((P_i, P_j) = (P_j, P_i)\) is a new variable if there exists a constraint that is a function of \(P_i, P_j\).

- Rewrite constraints as a function on such pairs of vars.

- So now two nodes in the new CN has an edge if the orig. vars. from which those nodes were formed are vars. of some orig. constraint.

\[ \text{Ex.: Denote } \overrightarrow{P_{23}} = (p_{23}, p_{32}), \overrightarrow{P_{35}} = (p_{35}, p_{53}) \]

\[ \text{Then } C^w(\overrightarrow{P_{23}}, \overrightarrow{P_{25}}, \overrightarrow{P_{35}}) = 1 \text{ iff } C_3(p_{23}, p_{35}, p_{52}) = 1 \text{ and } p_{23} = p_{32}, \]

\[ p_{35} = p_{53}, \quad p_{32} = p_{32} \]

iff \( \overrightarrow{P_{23}}, \overrightarrow{P_{25}}, \overrightarrow{P_{35}} \) consistent among themselves and with constraint.

- Can use clusters of larger or different sizes!
Variable elimination: A general "algorithm"

- Here we apply it to solve CSPs.
- Consider the graphical game (6b) example:

\[ S(\vec{p}) = \max_{i=1,n} L_i(\vec{p}) \]

where \( L_i(\vec{p}) \) is some "loss function" associated with player \( i \).

If we let \( L_i(\vec{p}) = \begin{cases} 0 & \text{if } \vec{p} \text{ is pure} \\ 1 & \text{otherwise} \end{cases} \) then \( \vec{p}^* \) is a best-response condition for player \( i \).

Then, \( \vec{p}^* \) is a K-NE for the game iff \( S(\vec{p}) = 0 \).

To find Nash equilibria in space of mixed strategies, we want to find \( \vec{p}^* \) which globally minimizes the "cost function" \( S \):

\[ \vec{p}^* \in \arg\min_{\vec{p} \in D^n} S(\vec{p}) \]

In the example above, we have:

\[ S(\vec{p}) = \max(L_1(p_1, p_6), L_2(p_2, p_1, p_3, p_4), L_3(p_3, p_2, p_5), L_4(p_4, p_2, p_5), L_5(p_5, p_3, p_4, p_6), L_6(p_6, p_5, p_7, p_8), L_7(p_7, p_6, p_8), L_8(p_8, p_6, p_5)) \]
\[
\min_{\beta} S(\beta) = \min_{\beta} \max_{\hat{P} \in \mathcal{D}} \left[ L_1(\hat{P}, P_2), L_2(P_2, P_3, P_4), L_3(P_3, P_2, P_5), L_4(P_4, P_2, P_5), L_5(P_5, P_3, P_4, P_6), L_6(P_6, P_5, P_7, P_8), L_7(P_7, P_6, P_8), L_8(P_8, P_6, P_7) \right]
\]

("distribute minimization" w.r.t. \(P_8\))

\[
= \min_{(P_1, P_2) \in \mathcal{D}^{n-1}} \max_{(P_2, P_3) \in \mathcal{D}} \left[ L_1(P_1, P_2), L_2(P_2, P_1, P_3, P_4), L_3(P_3, P_2, P_5), L_4(P_4, P_2, P_5), L_5(P_5, P_3, P_4, P_6), L_6(P_6, P_5, P_7, P_8), L_7(P_7, P_6, P_8), L_8(P_8, P_6, P_7) \right]
\]

\[
= \min_{P_2 \in \mathcal{D}} \max_{P_1 \in \mathcal{D}} \left[ L_1(P_1, P_2), L_2(P_2, P_1, P_3, P_4), L_3(P_3, P_2, P_5), L_4(P_4, P_2, P_5), L_5(P_5, P_3, P_4, P_6), L_6(P_6, P_5, P_7, P_8), L_7(P_7, P_6, P_8), L_8(P_8, P_6, P_7) \right]
\]

(continue "distributing minimizations" w.r.t. other vars...)
\[ \min_{\mathbf{p} \in \mathcal{D}} \mathcal{S}(\mathbf{p}) = \min_{\mathbf{p} \in \mathcal{D}} \min_{\mathbf{p}_1} \max_{\mathbf{p}_2} \left[ L_1(\mathbf{p}_1, \mathbf{p}_2), \min_{\mathbf{p}_3} \min_{\mathbf{p}_4} L_2(\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) \right] \]

\[ \max_{\mathbf{p}_5} \min_{\mathbf{p}_3} L_3(\mathbf{p}_3, \mathbf{p}_2, \mathbf{p}_5), \min_{\mathbf{p}_4} L_4(\mathbf{p}_4, \mathbf{p}_2, \mathbf{p}_5) \]

\[ \min_{\mathbf{p}_6} L_5(\mathbf{p}_5, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_6), \min_{\mathbf{p}_7} \min_{\mathbf{p}_8} L_6(\mathbf{p}_6, \mathbf{p}_5, \mathbf{p}_7, \mathbf{p}_8), L_7(\mathbf{p}_7, \mathbf{p}_8), L_8(\mathbf{p}_8, \mathbf{p}_7), \]

\[ L_9(\mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7) \]

\[ L_{10}(\mathbf{p}_5, \mathbf{p}_6) \]

e tc...

**High level picture:**

Can decompose (optimization) problem into many (hopefully) smaller subproblems!

[By exploiting structure in the original problem]
Cost Minimization Problem (CMP)

- Motivation: "Select smallest possible & (regret) achievable for a given discretization size \( \mathcal{T} \)."

- Let \( L_i(\pi) \) = "regret for player i"
  \[
  \max_{\alpha} \{ M_i(\pi|\alpha) - M_i(\pi) \}
  \]

- Same operation as for (\( \epsilon \))best-response CSP (except different "loss functions")

For "loss functions" just defined,

\[
\min_{\pi} S(\pi) = \text{smallest possible regret } \epsilon \text{ achievable for given } \mathcal{T}
\]

\[
= \min_{\pi} \max_{i=1,n} L_i(\pi)
\]

Variable elimination

- Can be seen as operation on same graph as constraint network (CN)

For example,

```
1 2 3 4 5 6 7 8
```

\[
\downarrow \text{"eliminating variable } \pi_8 \text{"}
\]

```
1 2 3 4 5 6 7
```

\[
\downarrow \text{so on.}
\]
In general, "eliminating" one variable requires the following graph operation:

1. For every pair of neighbors of variable to be eliminated, add an edge between those neighbors (if one does not already exist).
2. Remove variable from graph and all edges incident to it.

Perform same operation recursively.

Remarks:

- Graph terminology: A clique is a set of nodes that are "fully connected" in the graph (every pair of nodes in the set has an edge joining them).

- Space & time is exponential in the size of the largest clique found in any of the graphs obtained by the process above (depends on elimination order).

- In example above, size of largest clique found for elimination order is 4, which in this case is equal to the max neighborhood of the graph of the game (so running time/space is poly in model size for this example).

In general, this won't be the case. Consider for example (two-dimensional grid) graph structure:

```
N
```

---
Remarks:

- Gaussian elimination (for solving linear systems of equations) is a special case of variable elimination.

- Tree Nash can also be seen as a special case of variable elimination.

- In particular, for GSs with arbitrary graphs, variable elimination is computationally equivalent to applying Tree Nash to "hyper-tree" resulting from node-merging operation. (not necessarily unique)

- Any resulting "hyper-tree" has a corresponding "associated" elimination order for which var. elim. running time is of the same order as running Tree Nash in the hyper-tree, and vice versa...
Hybrid approaches: (UK'02) [A brief sketch]

- Approx eq. refinement:
  Idea: perform "search" in a small subset of the space; recursively make finer discretization.

  Watch out! Might "lose" eq. in the process.
  (An & NE in some region of the space does not guarantee that & exact NE in that region; finer granularity in the discretization might not lead to Nash equilibrium.)

- Subgame decomposition: An alternative representation/formalization of the CSP (closely connected to the process of variable elimination used to compute a solution)

- Assign a player to exactly one cluster s.t. original neighbors also in cluster [ex. assign 2,3,8,9 to cluster {1,2,3,4,8,9}]

  Idea: "Clamping" mixed strategies for players in
  separator {125} and {234} render eq. in clusters
  conditionally independent eq. in other clusters.

  i.e., values for 8,9 that satisfy eq. constraints
  values for {1,2,3,4} assuming
  values for separators consistent

- Setup a CSP where separators are vars and
  (binary) constraint among them reflect consistency and
  eq. conditions

  Hybrid method results from using
  "hill climbing" within clusters
  (as opposed to using var. elim.).
• Other algorithms exist for solving CSP.

• Constraint propagation algorithms: CP, based on the notion of arc consistency (AC).

• NashProp can be seen as a particular instantiation of a constraint propagation alg. for AC.