Efficient Solving of Multi-Player Game Situations

Martin Hrubý

Brno University of Technology
Brno
Czech Republic

November 23, 2008

Modeling the game situations on computers

Game of N players $Q = \{1, 2, ..., N\}$ is:

$$\Gamma = (S_1, S_2, ..., S_N; U_1, U_2, ..., U_N)$$

- Decision problem. Solution through its model (simulation experiment).
- Designing the strategy sets S_i. $S = S_1 \times S_2 \times ... \times S_N$.
- Computing the utility functions $U_i : S \rightarrow \mathbb{R}$.
- Determining the equilibrium.

Correlated equilibrium

- Equilibrium concept created by prof. Aumann.
- CE is suitable in situations of market competitions when rationality is a common knowledge.
- Leads to a pareto–efficient equilibrium.

Computing CE

- Linear programming task.
- Variables $p = (p_1, p_2, ..., p_{|S|})$ mapped from S.
- $Z(s) = \sum_{i \in Q} w_i U_i(s)$

$$MAXIMIZE : Z = \sum_{j=1}^{|S|} p_j * s_j$$

Constraints:

$$\sum_{j=1}^{|S|} p_j = 1; p_j \in \langle 0, 1 \rangle$$

$$Gp^T \geq 0$$

G-matrix

G-matrix collects all relative preferences of players in Γ game regarding their strategies and can be constructed by following rules:

1. Rows of G-matrix are indexed ijk where i indexes a player, j his strategy and k his alternative strategy. There are $\sum_{i=1}^{N} |S_i|.(|S_i| - 1)$ of rows in the G-matrix.
2. Columns of G are particular strategy profiles of the Γ game. There are $\Pi_{i=1}^{N} |S_i|$ of columns.
3. Cell $g_{ijk,s}$ in the $G-$matrix at ijk row and column $s \in S$:

$$g_{ijk,s} = \begin{cases} U_i(s) - U_i((k, s_{-i})) & s_i = j \\ 0 & otherwise \end{cases}$$

Example

$$\Gamma = (\{a, b\}, \{c, d\}; U_1, U_2)$$

Γ	c	d
a	10,2	8,4
b	5,4	7,3

The G-matrix for Γ:

	ac	ad	bc	bd
$a \longrightarrow b$	5	1		
$b \longrightarrow a$			-5	-1
$c \longrightarrow d$	-2		1	
$d \longrightarrow c$		2		-1

This system will generate a following LP problem (profiles are ordered as (ac, ad, bc, bd)):

$$MAXIMISE : Z = 12p_1 + 12p_2 + 9p_3 + 10p_4 \tag{1}$$

$$p_{1..4} \in \langle 0, 1 \rangle \tag{2}$$

$$\sum_{i=1}^{4} p_i = 1 \tag{3}$$

$$5p_1 + p_2 \geq 0 \tag{4}$$

$$-5p_3 - p_4 \geq 0 \tag{5}$$

$$-2p_1 + p_3 \geq 0 \tag{6}$$

$$2p_2 - p_4 \geq 0 \tag{7}$$

Efficient approach to compute it

Problems:

- Large strategy sets. Many profiles.
- Large LP-task to solve.
- Time & memory limitations.

Solution:

- There are dominant and dominated strategies.
- How to discover dominated strategies?
- Iterative elimination of dominated strategies.

Approach:

- A) State space $\rightarrow$ B) game minimization $\rightarrow$ C) CE.
- Put A) and B) together.

Back to the example

	ac	ad	bc	bd
$a \longrightarrow b$	5	1		
$b \longrightarrow a$			-5	-1
$c \longrightarrow d$	-2		1	
$d \longrightarrow c$		2		-1

$$-5p_3 - p_4 \geq 0$$

	ab	ad
$a \longrightarrow b$	5	1
$c \longrightarrow d$	-2	
$d \longrightarrow c$		2

$\Rightarrow$

	ad
$a \longrightarrow b$	1
$d \longrightarrow c$	2

Gsolve: Parallel algorithm solving CE

1. For all players $i \in Q$:
2. For all $j \in S_i$:
3. For all $k \in S_i, j \neq k$:
4. Compute G_{ijk} in parallel
5. If negative, disable related profiles and propagate backwards in G.

Result:

- Remaining profiles determine $S_i, \forall i \in Q$
- G is already computed
- LP-task can be generated

Data types and structures

- $valid : S \rightarrow Boolean$ is a boolean array displaying what profiles are valid or non-valid (zero/non-zero probability). Initially, all profiles are valid.
- $umap : S \rightarrow \mathbb{R}^N$ is a dynamic dictionary (items can be added and removed during the computation) assigning payoff vectors to particular profiles. Initially, *umap* is empty.
- $dominated : Q \times S_i \rightarrow Boolean$ is a boolean array displaying what strategies are already found to be dominated. Initially, all are set *False*.
- type *profList* is a list of tuples $(S, \mathbb{R})$.
- type $gRow = (pos : profList, neg : profList)$
- $G[i, j, k]$ is a dynamic list of *gRow* indexed by (i, j, k). G represents the G-matrix.

Gsolve – main part

```
for i in Q:
for j in S[i]:
for k in S[i]:
 if (j!=k):
  if (not(dominated[i][j]
  or dominated[i][k])):
     row = solveRow(i,j,k);
     if (row != ([],[])):
        if (row.pos != []):
          G[i,j,k] := row;
        else:
          dominated[i][k] := true
          for (s,r) in row.neg:
            disableProf(i,s);
```

Gsolve – solveRow(i,j,k)

```
pos := []; neg := [];
for s in S: # run this in parallel
  if (s[i]==j and valid[s]):
    s2 = s; s2[i] := k;
    if (valid[s2]):
       uj := getU(s,i);
       uk := getU(s2,i);
       if (uj != Nil and uk != Nil):
          diff := uj - uk;
          if (diff != 0):
             if (diff>0):
                pos.append( (s, diff) );
             else:
                neg.append( (s, diff) );
return (pos, neg);
```

Gsolve – getU(s,i)

```
if (umap[s]==Nil):
  # this is the main computation load
  umap[s] := cellModel(s, C);
return umap[s][i];
```

Gsolve – disableProf(it,s)

```
valid[s] := False
umap[s] := Nil # free the umap item
for i in [1,...,it]:
for j in S[i]:
for k in S[i]:
   row := G[i,j,k]
   row.neg.removeKeyIfPresent(s);
   row.pos.removeKeyIfPresent(s);
   if (row.pos == []):
      for (s2,r) in row.neg:
          disableProf(it, s2)
      G[i,j,k] := Nil
```