Solving Extensive-Form Games

Branislav Bošanský and Michal Pěchouček

Artificial Intelligence Center,
Department of Computer Science,
Faculty of Electrical Engineering,
Czech Technical University in Prague

branislav.bosansky@agents.fel.cvut.cz

November 8, 2016
Previously ... on multi-agent systems.

1. Extensive-Form Games
2. Transformations between representations
Imperfect Information Extensive-Form Games

Tree diagram with nodes and payoffs:
- Node 1: 2-0, 1-1, 0-2
- Node 2: 2
- Payoffs:
  - (0,0)
  - (2,0)
  - (0,0)
  - (1,1)
  - (0,0)
  - (0,2)
Why backward induction does not work?

Exact algorithms:

- We can solve an EFG as a normal-form game.
- We can use so-called *sequence form* to formulate a linear program that has a linear size in the size of the game.

Approximate algorithms:

- Counterfactual Regret Minimization (CFR)
- Excessive Gap Technique (EGT)
Imperfect Information EFG
Induced Normal-Form Game

Normal form representation is too verbose. The same leaf is stated multiple times in the table.

We can avoid it by using sequences.
Sequences in Extensive-Form Games

Definition

An ordered list of actions of player $i$ executed from the root of the game tree to some node $h \in \mathcal{H}$ is called a sequence $\sigma_i$. Set of all possible sequences of player $i$ is denoted $\Sigma_i$. 
Sequences in Extensive-Form Games

**Definition**

An ordered list of actions of player $i$ executed from the root of the game tree to some node $h \in \mathcal{H}$ is called a *sequence* $\sigma_h$. Set of all possible sequences of player $i$ is denoted $\Sigma_i$. 

\[ \begin{array}{c|c|c} \triangle(\Sigma_1) & \bigcirc(\Sigma_2) \\ \hline \emptyset & \emptyset \\ A & X \\ B & Y \\ AC & Z \\ AD & W \\ BE & \\ BF & \end{array} \]
We need to extend the utility function to operate over sequences:

\[ g : \sum_1 \times \sum_2 \rightarrow \mathbb{R}, \]

where \( g(\sigma_1, \sigma_2) = \)

\[ u(z) \text{ iff } z \text{ corresponds to history represented by sequences } \sigma_1 \text{ and } \sigma_2 \]

\[ 0 \text{ otherwise} \]
Extended Utility Function

In games with chance a combination of sequences can lead to multiple nodes/leafs. $g(\sigma_1, \sigma_2) =$

- $\sum_{z \in Z'} C(z)u(z)$ iff $Z'$ is a set of leafs that correspond to history represented by sequences $\sigma_1$ and $\sigma_2$, and $C(z)$ represents the probability of leaf $z$ being reached due to chance
- 0 otherwise
Extended Utility Function

Examples:

- \( g(\emptyset, W) = 0 \)
- \( g(AC, W) = 0 \)
- \( g(BF, W) = 3 \)
- \( g(A, X) = 0 \)
- \( g(BE, W) = 0 \)
- \( g(BF, W) = 3 \)
We need to express the strategy using sequences. We need to be prepared for all situations.

Let’s assume that the opponent (player 2) will play everything and assign a probability that certain sequence $\sigma_1$ will be played.

A realization plan $(r_i(\sigma_i))$ is a probability that sequence $\sigma_i$ will be played assuming player $-i$ plays such actions that allow actions from $\sigma_i$ to be executed.
Realization Plans

Examples:

- $r_1(\emptyset) = 1$
- $r_1(A) + r_1(B) = r_1(\emptyset)$
- $r_1(AC) + r_1(AD) = r_1(A)$
- $r_1(BE) + r_1(BF) = r_1(B)$
- $r_2(\emptyset) = 1$
- $r_2(X) + r_1(Y) = r_2(\emptyset)$
- $r_2(Z) + r_1(W) = r_2(\emptyset)$
We now have almost everything – a strategy representation and an extended utility function.

We will have a maximization objective and need a best response for the minimizing player.

A player selects the best action (the one that minimizes the expected utility) in each information set.

An expected utility after playing an action in an information set corresponds to a sum of (1) utility values of leaves and (2) information sets that are immediately reached.
We are now ready to state the linear program:

\[
\begin{align*}
\max & \quad v(r_{1},v) \\
\text{s.t.} & \quad r_{1}() = 1 \\
& \quad 0 \leq r_{1}(\sigma) \leq 1 \quad \forall \sigma \in \Sigma_{1} \\
& \quad \sum_{a \in A(I_{1})} r_{1}(\sigma_{1}a) = r_{1}(\sigma_{1}) \quad \forall \sigma_{1} \in \Sigma_{1}, \forall I_{1} \in \inf_{1}(\sigma_{1}) \\
& \quad \sum_{I' \in \inf_{2}(\sigma_{2}a)} v(I') + \sum_{\sigma_{1} \in \Sigma_{1}} g(\sigma_{1}, \sigma_{2}a)r_{1}(\sigma_{1}) \geq v(I) \quad \forall I \in \mathcal{I}_{2}, \sigma_{2} = \text{seq}_{2}(I), \forall a \in A(I) 
\end{align*}
\]

- \(\text{seq}_{i}(I)\) is a sequence of player \(i\) to information set,
- \(I \in \mathcal{I}_{i}, v_{I}\) is an expected utility in an information set,
- \(\inf_{i}(\sigma_{i})\) is an information set, where the last action of \(\sigma_{i}\) has been executed,
- \(\sigma_{i}a\) denotes extending a sequence \(\sigma_{i}\) with action \(a\)
Sequence Form LP - Example

\[
\begin{align*}
\text{max} & \quad v(\inf_2(X)) + v(\inf_2(Z)) \\
& \quad r_1(\emptyset) = 1; \quad r_1(A) + r_1(B) = r_1(\emptyset) \\
& \quad r_1(AC') + r_1(AD) = r_1(A), \\
& \quad r_1(BE) + r_1(BF) = r_1(B) \\
& \quad v(\inf_2(X)) \leq 0 + g(AC, X)r_1(AC) + g(AD, X)r_1(AD) \\
& \quad v(\inf_2(Y)) \leq 0 + g(AC, Y)r_1(AC') + g(AD, Y)r_1(AD) \\
& \quad v(\inf_2(Z)) \leq 0 + g(BE, Z)r_1(BE) + g(BF, Z)r_1(BF) \\
& \quad v(\inf_2(W)) \leq 0 + g(BE, W)r_1(BE) + g(BF, W)r_1(BF)
\end{align*}
\]
\[ \min_{r_2, v} v(\inf_1(A)) \]  
\[ r_2(\emptyset) = 1; r_2(X) + r_2(Y) = r_2(\emptyset) \]  
\[ r_2(Z) + r_2(W) = r_2(\emptyset) \]  
\[ v(\inf_1(A)) \geq \inf_1(AC), \ v(\inf_1(B)) \geq \inf_1(BE) \]  
\[ v(\inf_1(AC)) \geq g(AC, X)r_2(X) + g(AC, Y)r_2(Y) \]  
\[ v(\inf_1(AD)) \geq g(AD, X)r_2(X) + g(AD, Y)r_2(Y) \]  
\[ v(\inf_1(BE)) \geq g(BE, Z)r_2(Z) + g(BE, W)r_2(W) \]  
\[ v(\inf_1(BF)) \geq g(BF, Z)r_2(Z) + g(BF, W)r_2(W) \]
Nash equilibrium of a general-sum game can be (similarly to NFGs) found by solving a sequence form LCP (linear complementarity problem)

- satisfiability program
- realization plans for both players
- connection between realization plans and best responses via \textit{complementarity constraints}
- best-response inequalities are rewritten using slack variables

\[
\begin{align*}
    r_i(\emptyset) &= 1 \\
    0 &\leq r_i(\sigma_i) \leq 1 \\
    \sum_{a \in A(I_i)} r(\sigma_i a) &= r(\sigma_i) \\
    \sum_{I' \in \inf_{-i}(\sigma_{-i} a)} v(I') + \sum_{\sigma_i \in \Sigma_i} g(\sigma_i, \sigma_{-i} a) r_i(\sigma_i) + s_{\sigma_{-i} a} &= v(I) \\
    r(\sigma_i) s(\sigma_i) &= 0 \\
    s(\sigma_i) &\geq 0
\end{align*}
\]
For computing one (any) Nash equilibrium

- Lemke algorithm (Lemke-Howson)

If we want to compute some specific Nash equilibrium (e.g., maximizing welfare, maximizing utility for some player, etc.)

- MILP reformulations (Sandholm et al. 2005, Audet et al. 2009)

- Complementarity constraints can be replaced by using a binary variable that represents whether a sequence is used in a strategy with a non-zero probability

- Big-M notation

- Poor performance ($10^4$ nodes) using state-of-the-art MILP solvers (e.g., IBM CPLEX, ...)
Instead of computing the strategy, we can employ learning algorithms and learn the best strategy via repeated (simulated, or self-) play.

In zero-sum games, no-regret learning techniques are very popular (and useful in practice).

Main idea:
- construct the complete game tree
- in each iteration traverse through the game tree and adapt the strategy in each information set according to the learning rule
- this learning rule minimizes the (counterfactual) regret
- the algorithm minimizes the overall regret in the game
- the average strategy converges to the optimal strategy
Regret and Counterfactual Regret

Player $i$’s regret for *not playing* an action $a'_i$ against opponent’s action $a_{-i}$

$$u_i(a'_i, a_{-i}) - u_i(a_i, a_{-i})$$

In extensive-form games we need to evaluate the value for each action in an information set (*counterfactual value*)

$$v_i(s, I) = \sum_{z \in Z_I} \pi^s_{-i}(z[I]) \pi^s_i(z|z[I]) u_i(z),$$

where

- $Z_I$ are leafs reachable from information set $I$
- $z[I]$ is the history prefix of $z$ in $I$
- $\pi^s_i(h)$ is the probability of player $i$ reaching node $h$ following strategy $s$
Regret and Counterfactual Regret

Counterfactual value for one deviation in information set $I$; strategy $s$ is altered in information set $I$ by playing action $a: v_i(s_{I \rightarrow a}, I)$

at a time step $t$, the algorithm computes counterfactual regret for current strategy

$$r^t_i(I, a) = v_i(s_{I \rightarrow a}, I) - v_i(s_I, I)$$

the algorithm calculates the cumulative regret

$$R^T_i = \sum_{t=1}^{T} r^t_i(I, a), \quad R^{T,+}_i(I, a) = \max\{R^T_i(I, a), 0\}$$

strategy for the next iteration is selected using regret matching

$$s^{t+1}_i(I, a) = \begin{cases} \frac{R^{T,+}_i(I, a)}{\sum_{a' \in A(I)} R^{T,+}_i(I, a')} & \text{if the denominator is positive} \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases}$$
Average cumulative regret converges to zero with iterations and average strategy converges to an optimal strategy.

There are many additional improvements (sampling, MC versions, ...) and modifications of CFR.

CFR+ was used to solve two-player limit poker (Bowling et al. 2015) that uses only positive updates of regret and instead of the average strategy the algorithm uses the immediate (or current) strategy.

The immediate strategy does not (provably) converge to Nash equilibrium.
Comparing SQF and CFR

**Sequence Form**
- the leading exact algorithm (with incremental variants)
- large memory requirements
- incremental variants (or double-oracle) work well on games with small support

**CFR**
- practical optimization algorithm
- memory requirements can be reduced with domain-specific implementation
- converges very slowly if the close approximation is required