Multiagent Systems (BE4M36MAS)

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

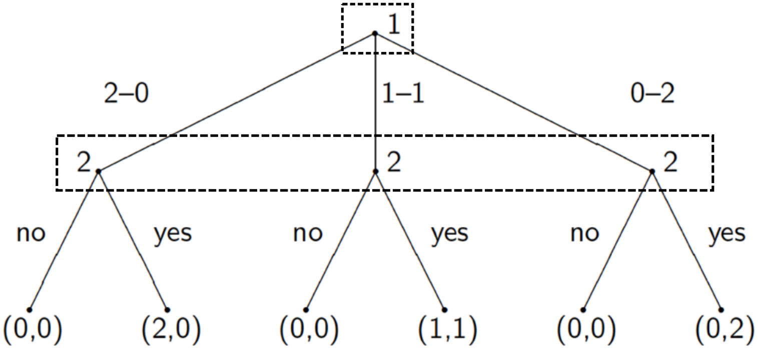
branislav.bosansky@agents.fel.cvut.cz

November 8, 2016

Previously ... on multi-agent systems.

1. Extensive-Form Games
2. Transformations between representations

Imperfect Information Extensive-Form Games



Imperfect Information Extensive-Form Games

Why backward induction does not work?

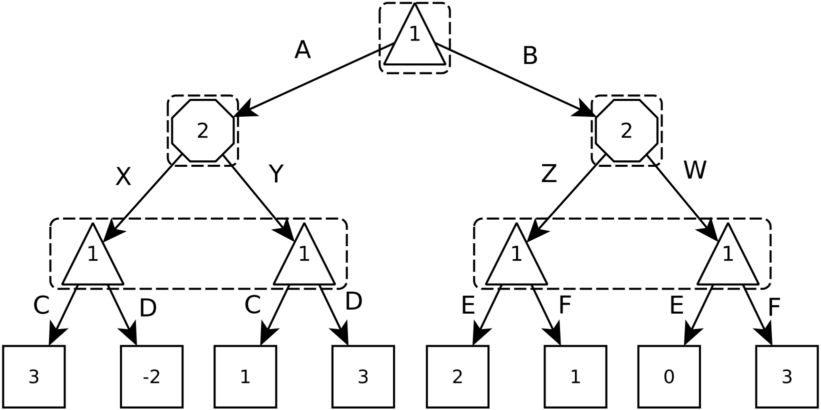
Exact algorithms:

- 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.

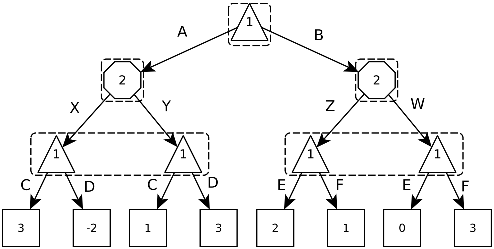
Approximate algorithms:

- Counterfactual Regret Minimization (CFR)
- Excessive Gap Technique (EGT)

Imperfect Information EFG



Induced Normal-Form Game

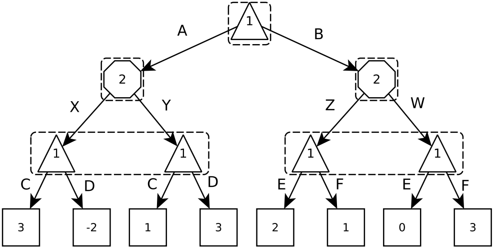


	XZ	XW	YZ	YW
ACE	3	3	1	1
ACF	3	3	1	1
ADE	-2	-2	3	3
ADF	-2	-2	3	3
BCE	2	0	2	0
BCF	1	3	1	3
BDE	2	0	2	0
BDF	1	3	1	3

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

We can avoid it by using sequences.

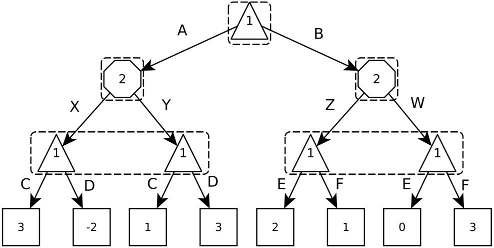
Sequences in Extensive-Form Games



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* σ_i. Set of all possible sequences of player i is denoted Σ_i.

Sequences in Extensive-Form Games

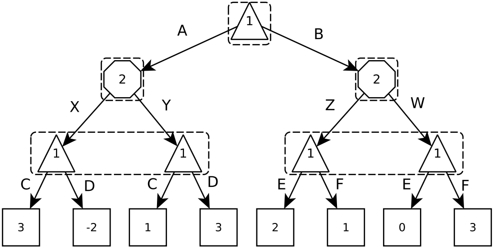


$\triangle(\Sigma_1)$	$\bigcirc(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

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* σ_h. Set of all possible sequences of player i is denoted Σ_i.

Extended Utility Function



$\triangle(\Sigma_1)$	$\bigcirc(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

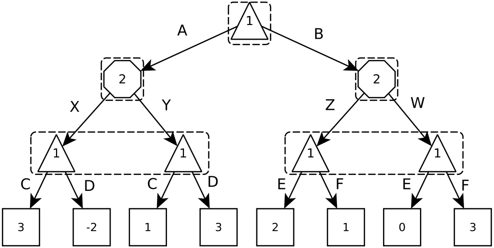
We need to extend the utility function to operate over sequences:

$$g : \Sigma_1 \times \Sigma_2 \to \mathbb{R},$$

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

- $u(z)$ iff z corresponds to history represented by sequences σ_1 and σ_2
- 0 otherwise

Extended Utility Function

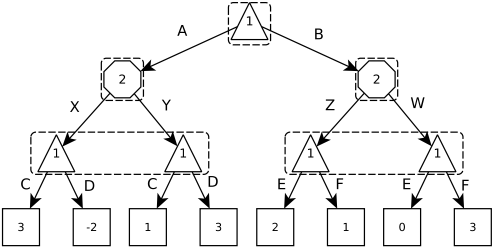


$\triangle(\Sigma_1)$	$\bigcirc(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

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

- $\sum_{z \in \mathcal{Z}'} \mathcal{C}(z)u(z)$ iff $\mathcal{Z}'$ is a set of leafs that correspond to history represented by sequences σ_1 and σ_2, and $\mathcal{C}(z)$ represents the probability of leaf z being reached due to chance
- 0 otherwise

Extended Utility Function

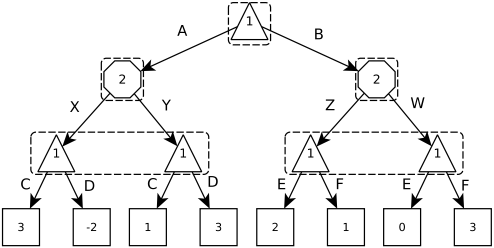


$\triangle(\Sigma_1)$	$\bigcirc(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

Examples:

- $g(\emptyset, W) = 0$
- $g(AC, W) = 0$
- $g(BF, W) = 3$
- $g(A, X) = 0$
- $\ldots$

Realization Plans



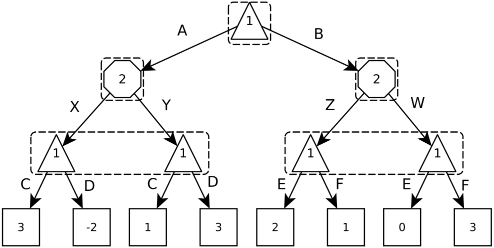
$\triangle(\Sigma_1)$	$\bigcirc(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

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 σ_1 will be played.

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

Realization Plans



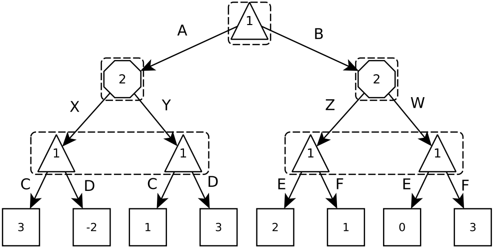
$\triangle(\Sigma_1)$	$\bigcirc(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

Examples:

- $r_1(\emptyset) = 1$
- $r_1(A) + r_1(B) = r_1(\emptyset)$
- $r_1(AC) + r_1(AD) = r_1(A)$
- $r_1(BE) + r_1(BF) = r_1(B)$

- $r_2(\emptyset) = 1$
- $r_2(X) + r_1(Y) = r_2(\emptyset)$
- $r_2(Z) + r_1(W) = r_2(\emptyset)$

Best Response



$\triangle(\Sigma_1)$	$\bigcirc(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

- 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 leafs and (2) information sets that are immediately reached.

Sequence Form Linear Program

We are now ready to state the linear program:

$$\max_{r_1, v} v(root) \tag{1}$$

$$\text{s.t.} \quad r_1(\emptyset) = 1 \tag{2}$$

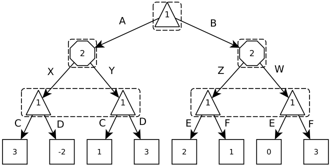
$$0 \leq r_1(\sigma_1) \leq 1 \qquad \forall \sigma_1 \in \Sigma_1 \tag{3}$$

$$\sum_{a \in \mathcal{A}(I_1)} r_1(\sigma_1 a) = r_1(\sigma_1) \quad \forall \sigma_1 \in \Sigma_1, \forall I_1 \in \mathsf{inf}_1(\sigma_1) \tag{4}$$

$$\sum_{I' \in \mathsf{inf}_2(\sigma_2 a)} v(I') + \sum_{\sigma_1 \in \Sigma_1} g(\sigma_1, \sigma_2 a) r_1(\sigma_1) \geq v(I) \qquad \forall I \in \mathcal{I}_2, \sigma_2 = \mathsf{seq}_2(I), \forall a \in \mathcal{A}(I) \tag{5}$$

- $\mathsf{seq}_i(I)$ is a sequence of player i to information set,
- $I \in \mathcal{I}_i$, v_I is an expected utility in an information set,
- $\mathsf{inf}_i(\sigma_i)$ is an information set, where the last action of σ_i has been executed,
- $\sigma_i a$ denotes extending a sequence σ_i with action a

Sequence Form LP - Example



$$\max_{r_1, v} v(\mathsf{inf}_2(X)) + v(\mathsf{inf}_2(Z)) \tag{6}$$

$$r_1(\emptyset) = 1; r_1(A) + r_1(B) = r_1(\emptyset) \tag{7}$$

$$r_1(AC) + r_1(AD) = r_1(A), \tag{8}$$

$$r_1(BE) + r_1(BF) = r_1(B) \tag{9}$$

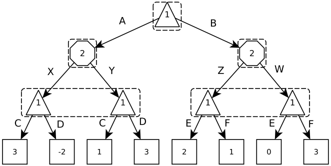
$$v(\mathsf{inf}_2(X)) \leq 0 + g(AC, X)r_1(AC) + g(AD, X)r_1(AD) \tag{10}$$

$$v(\mathsf{inf}_2(Y)) \leq 0 + g(AC, Y)r_1(AC) + g(AD, Y)r_1(AD) \tag{11}$$

$$v(\mathsf{inf}_2(Z)) \leq 0 + g(BE, Z)r_1(BE) + g(BF, Z)r_1(BF) \tag{12}$$

$$v(\mathsf{inf}_2(W)) \leq 0 + g(BE, W)r_1(BE) + g(BF, W)r_1(BF) \tag{13}$$

Sequence Form LP - Example



$$\min_{r_2,v} v(\mathsf{inf}_1(A)) \tag{14}$$

$$r_2(\emptyset) = 1; r_2(X) + r_2(Y) = r_2(\emptyset) \tag{15}$$

$$r_2(Z) + r_2(W) = r_2(\emptyset) \tag{16}$$

$$v(\mathsf{inf}_1(A)) \geq \mathsf{inf}_1(AC),\ v(\mathsf{inf}_1(B)) \geq \mathsf{inf}_1(BE) \tag{17}$$

$$v(\mathsf{inf}_1(AC)) \geq g(AC, X)r_2(X) + g(AC, Y)r_2(Y) \tag{18}$$

$$v(\mathsf{inf}_1(AD)) \geq g(AD, X)r_2(X) + g(AD, Y)r_2(Y) \tag{19}$$

$$v(\mathsf{inf}_1(BE)) \geq g(BE, Z)r_2(Z) + g(BE, W)r_2(W) \tag{20}$$

$$v(\mathsf{inf}_1(BF)) \geq g(BF, Z)r_2(Z) + g(BF, W)r_2(W) \tag{21}$$

Sequence Form Linear Complementarity Program

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 *complementarity constraints*
- best-response inequalities are rewritten using slack variables

$$r_i(\emptyset) = 1 \tag{22}$$

$$0 \le r_i(\sigma_i) \le 1 \tag{23}$$

$$\sum_{a \in \mathcal{A}(I_i)} r(\sigma_i a) = r(\sigma_i) \tag{24}$$

$$\sum_{I' \in \mathsf{inf}_{-i}(\sigma_{-i} a)} v(I') + \sum_{\sigma_i \in \Sigma_i} g(\sigma_i, \sigma_{-i} a) r_i(\sigma_i) + s_{\sigma_{-i} a} = v(I) \tag{25}$$

$$r(\sigma_i) s(\sigma_i) = 0 \tag{26}$$

$$s(\sigma_i) \ge 0 \tag{27}$$

General Sum Extensive-Form Games

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, ...)

Approximate Algorithms for Extensive-Form Games

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)*

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

where

- $\mathcal{Z}_I$ 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 $a : v_i(s_{I\to a}, I)$

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

$$r_i^t(I,a) = v_i(s_{I\to a}, I) - v_i(s_I, I)$$

the algorithm calculates the *cumulative regret*

$$R_i^T = \sum_{t=1}^{T} r_i^t(I,a), \qquad R_i^{T,+}(I,a) = \max\{R_i^T(I,a), 0\}$$

strategy for the next iteration is selected using *regret matching*

$$s_i^{t+1}(I,a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a'\in\mathcal{A}(I)} R_i^{T,+}(I,a')} & \text{if the denominator is positive} \\ \frac{1}{|\mathcal{A}(I)|} & \text{otherwise} \end{cases}$$

Regret and Counterfactual Regret

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