Algorithms for Solving Dynamic Games with Imperfect Information

Branislav Bošanský

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

bosansky@fel.cvut.cz

October 29, 2018

Dynamic Games with Imperfect Information

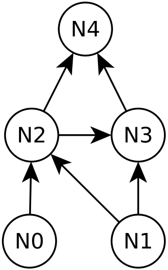
- Why do we need dynamic games?
- How does imperfect information complicate solving dynamic games?
- What are (some of) the algorithms for solving these dynamic games?

Possible Applications of Dynamic Games

Attackers execute their attacks step by step, leaving traces that can be spotted.

Dynamic examples of Adversarial Machine Learning problems (boiling frog, red herring).

Simple Network Security Scenario – Flip-It Game



Flip-it Game in a network

- players aim to gain control over the hosts in the network
- the defender initially controls all hosts
- both players choose which node to attack/protect simultaneously (in case of a tie, the control of the node does not change)
- players only observe the result of their last move
- there are different rewards/costs for each node

Simple Network Security Scenario – Flip-It Game

How can we solve this game?

- Repeat the same strategy
  - In a structured environment, there is a clear dependency on the history (if a defender is unsuccessful in gaining control of a node N4, that means that one of nodes N2, N3 is compromised).
  - If a reaction to an opponent's move is not part of the game-theoretic reasoning, it can be easily exploited by the opponent.
- Solve as a dynamic game:
  - with a fixed number of turns – Extensive-Form Games (EFGs)
  - without a fixed number of turns – Partially Observable Stochastic Games (POSGs)
- Dynamic games inherently model **uncertainty in outcomes, rewards**, or **deception**.

Outline

- Essentials
  - Quick Introduction to Dynamic Games
  - Baseline Algorithm for Solving Extensive-Form Games
- State-of-the-art algorithms for Solving Extensive-Form Games
- Solving Partially Observable Stochastic Games
- Challenges and Vision

Not covered by this tutorial:

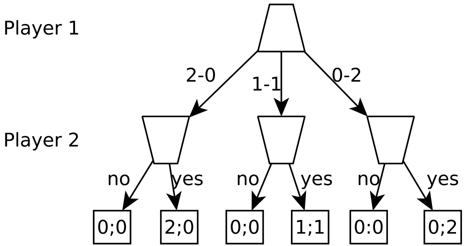
- Continuous games
- Purely heuristic algorithms with no guarantees (e.g., like IS-MCTS [Ciancarini and Favini, 2010])
- General-sum games and recent results in Stackelberg equilibrium computation [Cerny et al., 2018]

Extensive-Form Representation

Extensive-form games (EFGs) provide a compact representation (compared to normal-form games) to model games with finite and fixed number of turns.

They are visualized as **game trees**:

- **nodes** correspond to game states
- **edges** correspond to actions performed by a player in a state



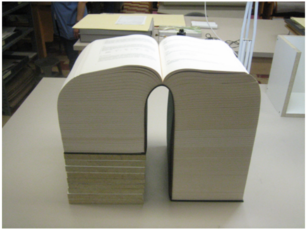
Extensive-Form Games (EFGs)

Formal Definition [Shoham and Leyton-Brown, 2009]:

- players $\mathcal{N} = \{1, 2, \ldots, n\}$
- actions $\mathcal{A}$
- choice nodes (histories) $\mathcal{H}$
- action function $\chi : \mathcal{H} \to 2^{\mathcal{A}}$
- player function $\rho : \mathcal{H} \to \mathcal{N}$
- terminal nodes $\mathcal{Z}$
- successor function $\varphi : \mathcal{H} \times \mathcal{A} \to \mathcal{H} \cup \mathcal{Z}$
- utility function $u = (u_1, u_2, \ldots, u_n)$; $u_i : \mathcal{Z} \to \mathbb{R}$

Strategies in Extensive-Form Games

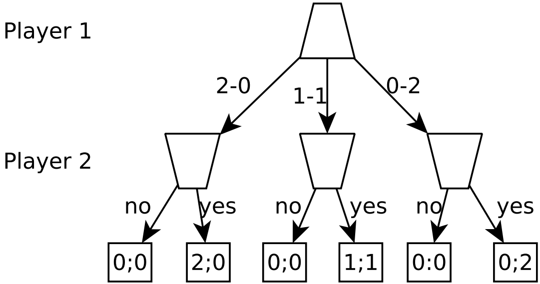
A strategy in dynamic games has to reflect all possible situations an agent can encounter in a game (due to moves of the opponent and/or stochastic events). Strategy prescribes which action should be played in any situation that can arise.



A pure strategy of player i in an EFG is an assignment of an action for each state where player i acts

$$S_i := \prod_{h \in \mathcal{H}, \rho(h)=i} \chi(h)$$

Strategies in EFGs



What are actions and strategies in this game?

$$\mathcal{A}_1 = \{2-0, 1-1, 0-2\};\ \mathcal{S}_1 = \{2-0, 1-1, 0-2\}$$

$$\mathcal{A}_2 = \{no, yes\};\ \mathcal{S}_2 = \{(no, no, no), (no, no, yes), \ldots, (yes, yes, yes)\}$$

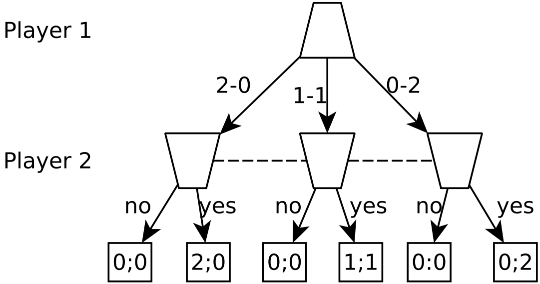
EFGs with Imperfect Information

When players are not able to observe the state of the game perfectly, we talk about *imperfect information games*. The states that are not distinguishable to a player belong to a single *information set*.

Formal Definition [Shoham and Leyton-Brown, 2009]:

- $\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{H}, \mathcal{Z}, \chi, \rho, \varphi, \gamma, u)$ is a perfect-information EFG.
- $\mathcal{I} = (\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_n)$ where $\mathcal{I}_i$ is a set of equivalence classes on choice nodes of a player i with the property that $\rho(h) = \rho(h') = i$ and $\chi(h) = \chi(h')$, whenever $h, h' \in I$ for some information set $I \in \mathcal{I}_i$
- we can use $\chi(I)$ instead of $\chi(h)$ for some $h \in I$

Strategies in EFGs with Imperfect Information



What are actions and strategies in this game?

$$\mathcal{A}_1 = \{2-0, 1-1, 0-2\};\ \mathcal{S}_1 = \{2-0, 1-1, 0-2\}$$

$$\mathcal{A}_2 = \{no, yes\};\ \mathcal{S}_2 = \{no, yes\}$$

Strategies in EFGs with Imperfect Information

Mixed strategies are defined as for normal-form games – a probability distribution over pure strategies.

In EFGs, *behavioral strategies* are more common:

- A *behavioral strategy* of player i is a product of probability distributions over actions in each information set

$$\beta_i : \prod_{I \in \mathcal{I}_I} \Delta(\chi(I))$$

There is an important class of imperfect-information games in which the expressiveness of mixed and behavioral strategies coincide – *perfect recall games.* Informally, in games with perfect recall no player forgets any information she previously knew.

Perfect Recall in EFGs

Definition

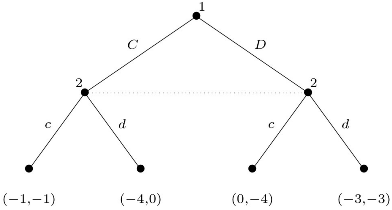
Player i has perfect recall in an imperfect-information game G if for any two nodes h, h' that are in the same information set for player i, for any path consisting of decisions of player i, $h_0, a_0, \ldots, h_n, a_n, h$ from the root of the game tree to h and for any path $h_0, a'_0, \ldots, h'_m, a'_m, h'$ from the root to h', it must be the case that:

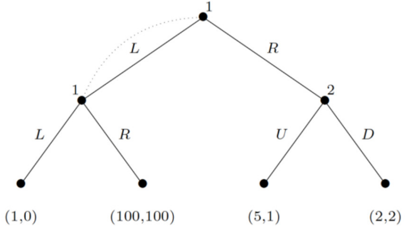
1. $n = m$
2. for all $0 \leq j \leq n$, h_j and h'_j are in the same equivalence class for player i, and $a_j = a'_j$

Definition

We say that an EFG has a *perfect recall* if all players have perfect recall. Otherwise we say that the game has an *imperfect recall*.

Perfect vs. Imperfect Recall





Conditioning on a complete history induces exponentially large strategies.[1]

They are easier to solve.

Strategies can be compactly represented.

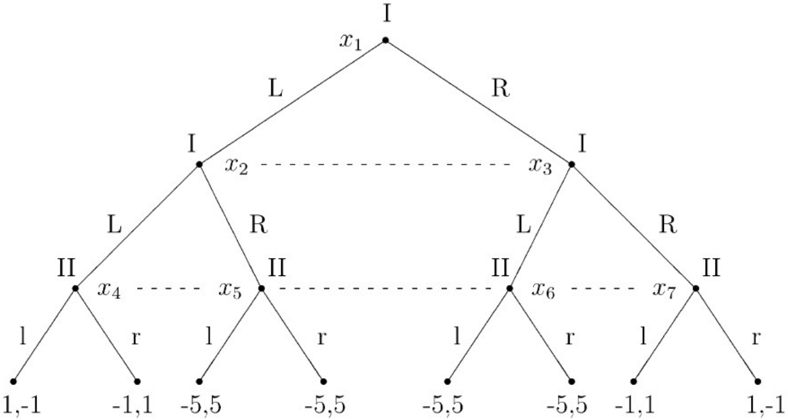
Unnecessary information can be forgotten; hence, the strategies can be (exponentially) smaller.

Much harder to solve (e.g., see [Koller and Megiddo, 1992, Cermak et al., 2018]).

Nash equilibrium (in behavioral strategies) might not exist.

[1] Figures are from [Shoham and Leyton-Brown, 2009].

Imperfect Recall Game with no NE



[2]

[2] Figure from [Wichardt, 2008].

Solving Imperfect Information Extensive-Form Games

Backward induction does not work, there is a dependence between the information sets.

The algorithms (typically) need to consider the game as a whole:

- 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.

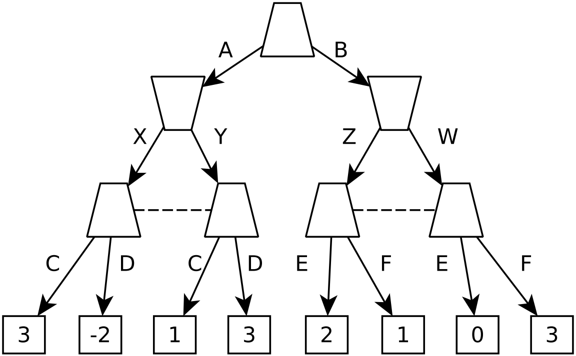
State-of-the-art algorithms:

- Double Oracle for Extensive-Form Games (DOEFG) [Bosansky et al., 2014]
- Counterfactual Regret Minimization (CFR) [Zinkevich et al., 2008, Tammelin, O. 2014]
- Excessive Gap Technique (EGT) [Hoda et al., 2010, Kroer et al., 2018]

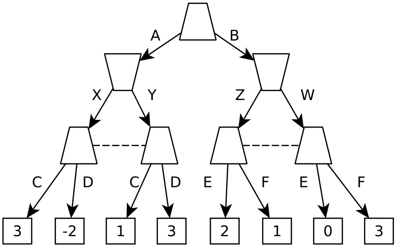
LP Algorithms for Extensive-Form Games

Algorithms based on *linear programming*

Imperfect Information EFG



Induced Normal-Form Game

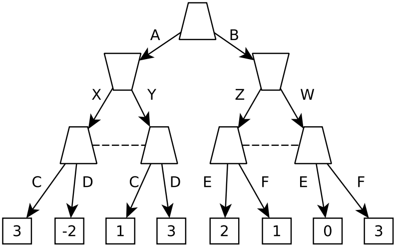


	XZ	XW	YZ	YW
ACE	3	3	1	1
ACF	3	3	1	1
ADE	-2	-2	3	3
ADF	-2	-2	3	3
BCE	2	0	2	0
BCF	1	3	1	3
BDE	2	0	2	0
BDF	1	3	1	3

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

We can avoid it by using sequences.

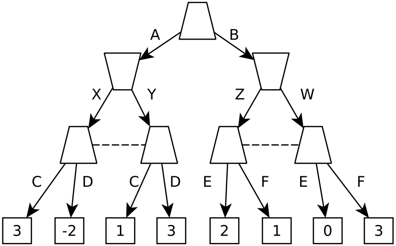
Sequences in Extensive-Form Games



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

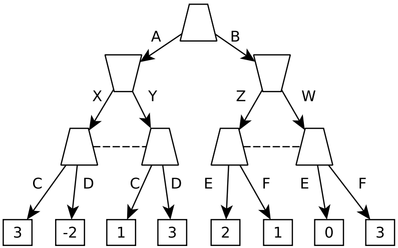


$\triangle(\Sigma_1)$	$\bigtriangledown(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

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_i. Set of all possible sequences of player i is denoted Σ_i.

Extended Utility Function



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

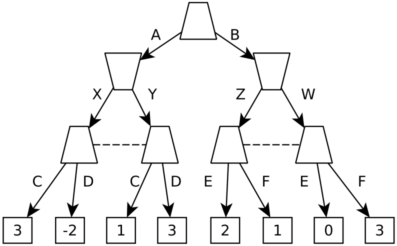
We need to extend the utility function to operate over sequences:

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

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

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

Extended Utility Function

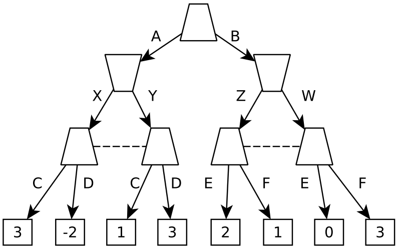


$\triangle(\Sigma_1)$	$\triangledown(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

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

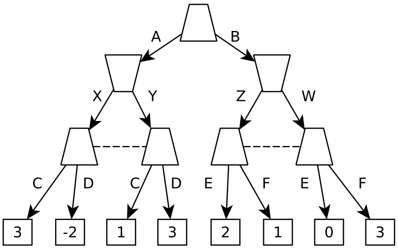


$\triangle(\Sigma_1)$	$\triangledown(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

Examples:

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

Realization Plans



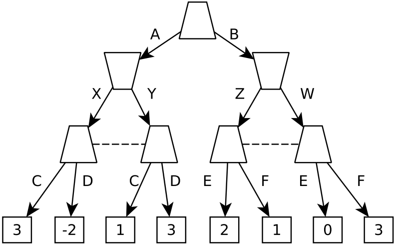
$\triangle(\Sigma_1)$	$\nabla(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

We need to express a mixed 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



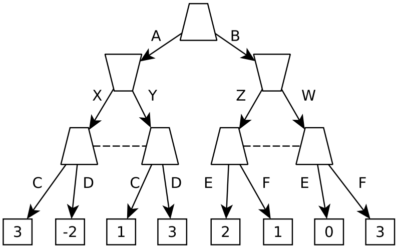
$\triangle(\Sigma_1)$	$\triangledown(\Sigma_2)$
$\emptyset$	$\emptyset$
A	X
B	Y
AC	Z
AD	W
BE	
BF	

Examples:

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

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

Best Response



$\triangle(\Sigma_1)$	$\bigtriangledown(\Sigma_2)$
$\emptyset$	$\emptyset$
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 (SQF)

We are now ready to state the linear program:

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

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

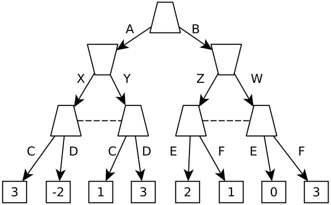
$$0 \leq r_1(\sigma_1) \leq 1 \qquad \forall \sigma_1 \in \Sigma_1 \tag{3}$$

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

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

- $\mathsf{seq}_i(I)$ 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 an extension of a sequence σ_i with action a

Sequence Form LP - Example



$$\max_{r_1, v} v(\inf_2(X)) + v(\inf_2(Z)) \tag{6}$$

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

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

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

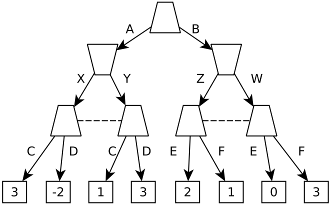
$$v(\inf_2(X)) \leq 0 + g(AC, X)r_1(AC) + g(AD, X)r_1(AD) \tag{10}$$

$$v(\inf_2(Y)) \leq 0 + g(AC, Y)r_1(AC) + g(AD, Y)r_1(AD) \tag{11}$$

$$v(\inf_2(Z)) \leq 0 + g(BE, Z)r_1(BE) + g(BF, Z)r_1(BF) \tag{12}$$

$$v(\inf_2(W)) \leq 0 + g(BE, W)r_1(BE) + g(BF, W)r_1(BF) \tag{13}$$

Sequence Form LP - Example



$$\min_{r_2, v} v(\inf_1(A)) \tag{14}$$

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

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

$$v(\inf_1(A)) \geq v(\inf_1(AC)), \; v(\inf_1(B)) \geq v(\inf_1(BE)) \tag{17}$$

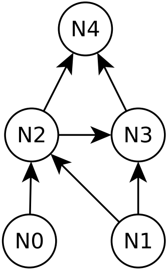
$$v(\inf_1(AC)) \geq g(AC, X) r_2(X) + g(AC, Y) r_2(Y) \tag{18}$$

$$v(\inf_1(AD)) \geq g(AD, X) r_2(X) + g(AD, Y) r_2(Y) \tag{19}$$

$$v(\inf_1(BE)) \geq g(BE, Z) r_2(Z) + g(BE, W) r_2(W) \tag{20}$$

$$v(\inf_1(BF)) \geq g(BF, Z) r_2(Z) + g(BF, W) r_2(W) \tag{21}$$

Simple Network Security Scenario – Flip-It Game



SQF for Flip-it Game in a network

Depth	Size (# Nodes)	Time [s]	LP Time [s]
3	15,685	1	1
4	495,205	23	8
5	16,715,941	–	–

Advantages/Disadvantages of SQF

- **(+)** the fastest exact algorithm (if the LP fits into memory)
- **(+)** quite easy to implement

- **(−)** scales poorly due to memory limitations
- **(−)** very difficult to make it domain-specific

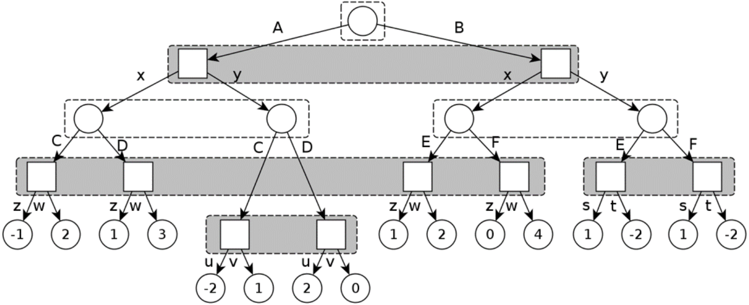
Incremental Strategy Generation

Large linear programs can be solved by an incremental construction of the LP. In game theory, the method has been known as *double-oracle algorithm*. There are 4 steps that repeat until convergence [Bosansky et al., 2014]:

1. **create a restricted game** – a simplified game where the players are allowed to choose only from a limited set of sequences of actions,
2. **solve the restricted game** – formalize the restricted game as a sequence-form LP and solve it,
3. **compute the best response** – each player computes a best response in the **original game** to the strategy from the **restricted game**,
4. **expand the restricted game** – if the best responses strictly improve the expected value, they are added as possible actions into the restricted game.

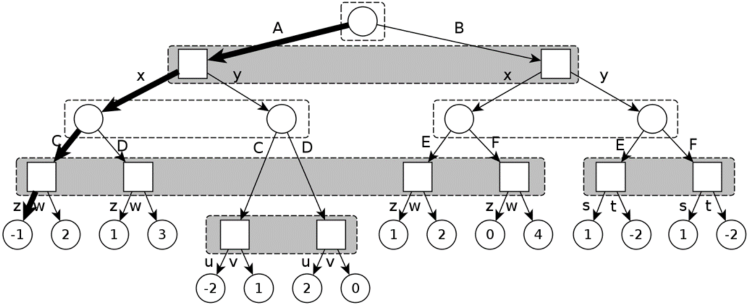
Double Oracle Algorithm for EFGs

The original game. Sequences that form the restricted game will be highlighted.



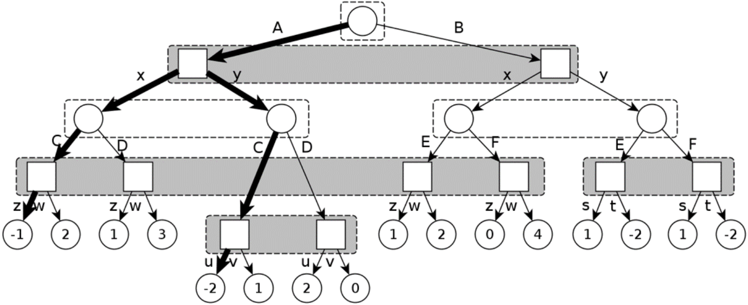
Double Oracle Algorithm for EFGs

Sequences AC and xz are added to the restricted game (as default sequences of actions).



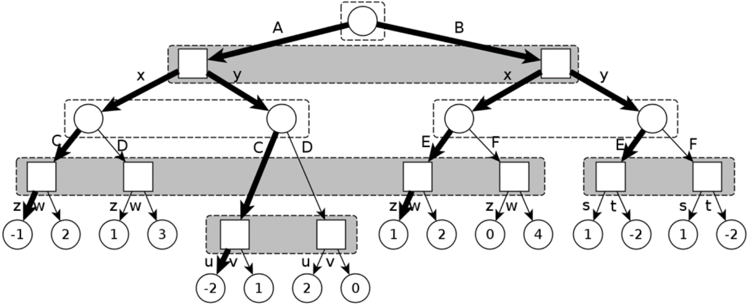
Double Oracle Algorithm for EFGs

Sequence *yu* is added to the restricted game as a best response of the minimizing player.



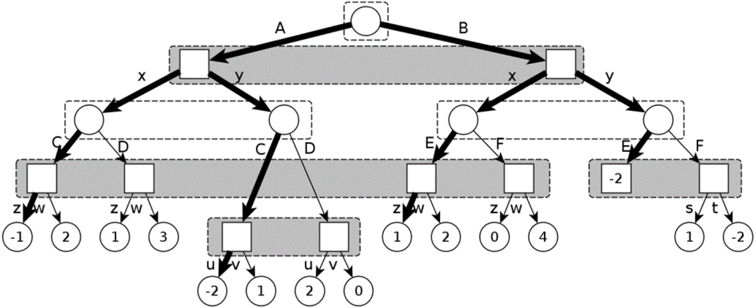
Double Oracle Algorithm for EFGs

Sequence *BE* is added to the restricted game as a best response of the maximizing player.



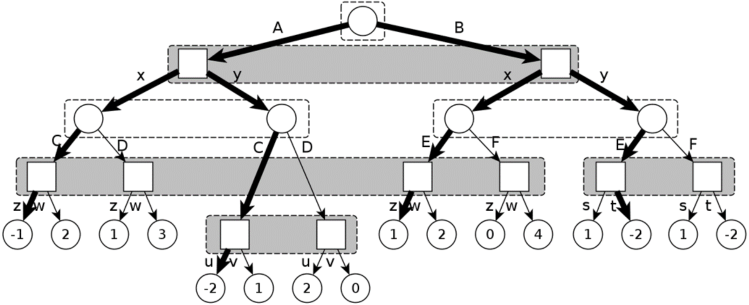
Double Oracle Algorithm for EFGs

There is no action defined for the node with history *ByE*. The algorithm turns that node into a temporary leaf and assigns a temporary utility value for that leaf.



Double Oracle Algorithm for EFGs

The algorithm turns the temporary leaf into a node when an action *s* or *t* is added into the restricted game.



Characteristics of DOEFG

Generalization of the double oracle principle to structured strategy spaces (such as sequences/realization plans).

Creating a valid restricted game is more complicated than adding a single strategy (one may need to create temporary leaves).

DOEFG converges in at most linear number of iterations in the size of the game tree (compared to the exponential number of iterations when using strategies).

Simple Network Security Scenario – Flip-It Game

DOEFG for Flip-it Game in a network

Depth	# Nodes	SQF [s]	SQF LP [s]	DOEFG [s]
3	15,685	1	1	1
4	495,205	23	8	9
5	16,715,941	–	–	508

Advantages/Disadvantages of DOEFG

- (+) can solve much larger domains compared to SQF
- (+) in a domain-independent way, the algorithm identifies necessary strategies to consider in a large EFG
- (+) best-response algorithms can be significantly improved for specific domains/problems

- (−) not that easy to implement
- (−) the sequence-form linear program of the restricted game can be a bottleneck

Simple Network Security Scenario – Flip-It Game

DOEFG with ordered moves for BR algorithm for Flip-it Game in a network

Depth	# Nodes	SQF [s]	SQF LP [s]	DOEFG [s]	DOEFG ordered [s]
3	15,685	1	1	1	1
4	495,205	23	8	9	5
5	16,715,941	–	–	508	168

For depth 6 (size $\approx 4x10^9$ nodes), DOEFG with ordered moves for BR reached error 0.1 in 2 hours.

Algorithms based on *Counterfactual Regret Minimization*

Approximate Algorithms for Extensive-Form Games

Instead of computing the optimal strategy directly, one can employ learning algorithms and learn the strategy via repeated (simulated, or self-) play.

The algorithm minimizes so called *regret* and these algorithms are also known as *no-regret learning* algorithms.

Main idea:

- 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*

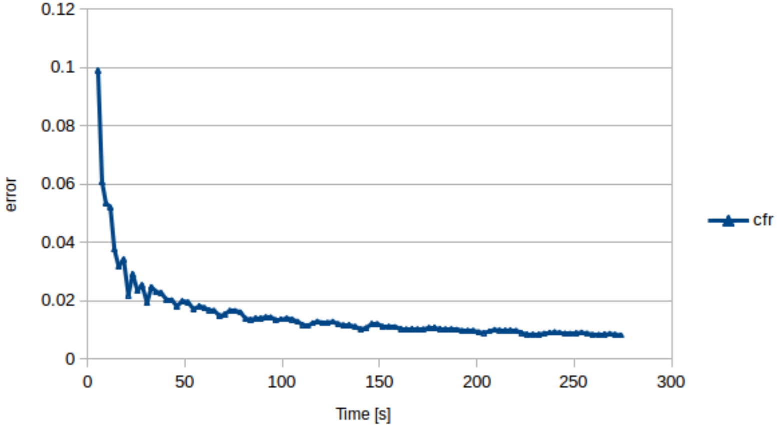
$$R_i^T = \sum_{t=1}^{T} r_i^t(I,a), \qquad R_i^{T,+}(I,a) = \max\{R_i^T(I,a), 0\}$$

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

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

Simple Network Security Scenario – Flip-It Game

CFR for Flip-it Game in a network[3]



[3] With the game tree pre-built in memory (took 1088s).

Extensions of Counterfactual Regret Minimization

There are **many** variants of the vanilla CFR algorithm:

- **MCCFR** – CFR updates are not performed in the complete game, but using outcome sampling (faster iterations) [Lanctot, 2013, Brown and Sandholm, 2016]
- **CFR-BR** – the second player performs a best-response (BR) update instead of a CFR update (ideal for games where a domain-specific BR algorithm is available) [Johanson et al., 2011]
- **CFR-D** – decomposition of CFR updates by subgames (helpful if the game is too large to keep all information sets in memory) [Burch et al., 2014]
- **CFR+** – main modification of the baseline CFR algorithm that significantly improves convergence [Tammelin, O. 2014]

Extensions of Counterfactual Regret Minimization (CFR+)

CFR+ differs from CFR in three aspects:

- only positive regrets are kept in cumulative regrets R_i^T
- players are alternating in the updates
- in the computation of the average strategy, first d iterations are ignored, later iterations are more important compared to first iterations

Sometimes, even the current strategy reaches low exploitability.

Extensions of Counterfactual Regret Minimization (CFR+)

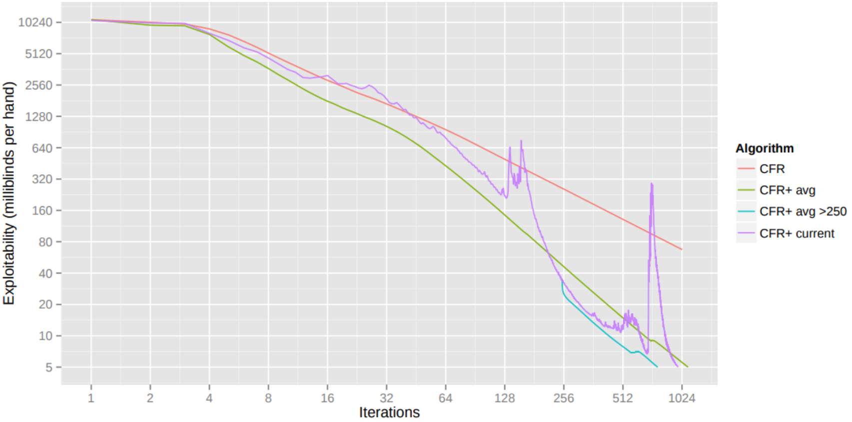


Figure 2: No Limit Texas Hold'em flop subgame

[4]

---

[4]Figure from [Tammelin, O. 2014].

Advantages/Disadvantages of CFR

- $(+)$ practical optimization algorithm
- $(+)$ easy to implement [Lanctot, 2013, p.22]
- $(+)$ memory requirements can be reduced with domain-specific implementation (or CFR-D)

- $(-)$ CFR converges very slowly if a close approximation is required (CFR$+$ is better)
- $(-)$ performance in other domains than poker is largely unknown (in some cases slower than DOEFG)

Continual Resolving and Deepstack

Is there no hope for a provably algorithm that behaves similarly to perfect information games?

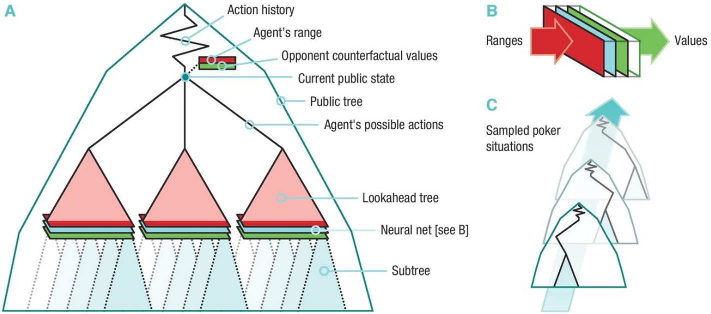
Recently, new methods that allow limited-lookahead algorithm for imperfect information games for poker [Moravcik et al., 2017, Brown and Sandholm, 2017].

Key properties:

- Use (a more complex) heuristic function to evaluate positions at the end of the depth-limited game tree
- Solve an EFG with a limited lookahead (e.g., using CFR or other algorithm)
- Use a specific gadget construction when advancing to next turn of the game.

One cannot assign a heuristic value just to a state (as in perfect information games), but to all states players consider possible.

Continual Resolving and Deepstack



[5] Picture from [Moravcik et al., 2017].

Generalization of Continual Resolving

Adaptation of continual resolving technique to other (security) domains is not straightforward:

- the actions are generally not observable (the defender does not know which host the attacker infected)
- the size of information sets (in number of possible states) increases exponentially with number of turns in the game
  - the size of the information sets is changing for the heuristic/neural network
  - the size of the information sets becomes impractical for large horizon
- the number of turns can be very large (e.g., Advanced Persistent Threats (APTs))

Modeling Games with Long/Infinite Horizon

We can use the formalism of Partially Observable Stochastic Games (POSGs).

There are several possible objectives in POSGs:

- **discounted sum** – future rewards are discounted with factor $\gamma < 1$
- **undiscounted sum** – e.g., attacker aims to compromise certain host while minimizing costs
- **reachability / safety criteria** – e.g., defender wants to ensure that certain hosts will not be compromised
- many others

Partially Observable Stochastic Games (POSGs)

POSGs are generally difficult to solve

- technically, one aims to solve an infinitely large extensive-form game with an infinite game tree and utilities defined over histories
- players need to consider their beliefs about the true state of the world, and beliefs the opponent has about the world, and the belief the opponent has about player's belief over the state of the world, and so on – **nested beliefs**
- one cannot avoid reasoning about the beliefs in this way without losing (approximate) optimality guarantees

However, using a fixed horizon is often too artificial (Why should the Flip-it Game stop after K iterations?).

Subclasses of Partially Observable Stochastic Games

From the practical/application perspective, we do not need to solve general POSG.

The goal can be to find robust (defensive) strategy against an attacker – we can assume **worst-case subclasses of POSGs**:

- One-Sided POSGs [Chatterjee and Doyen, 2014, Horak et al., 2017]
  - the attacker has perfect information
- POSGs with public actions/observations [Ghosh, et al., 2004]
  - POSG with public actions generalize poker to infinite/indefinite horizon (continual resolving approaches should still apply)
  - POSG with public observations are more general (e.g., both players can learn observations that change their beliefs)

Solving One-Sided Partially Observable Stochastic Games

One-Sided POSGs can be solved by adapting single-player algorithms designed for Partially Observable Markov Decision Processes (POMDPs).

We adapted Heuristic Search Value Iteration (HSVI) algorithm for solving games by [Horak et al., 2017]:

- showing that value function of One-Sided POSGs is a convex Lipschitz function in the belief of the defender,
- defining dynamic-programming operator that corresponds to solving a game at every stage,
- showing that the algorithm converges to value of the game.

The scalability is currently limited, but many of the possible improvements have not been tried/adapted yet.

Conclusions

... to conclude ...

Why Solving Dynamic Games

Dynamic Games provide **much more** realistic model for many real-world problems.

There already exists a variety of algorithms for solving zero-sum or non-zero-sum games.

Many of the existing algorithms can be tuned-up to improve scalability:

- DO can benefit from domain-specific best responses
- CFR can benefit from domain-specific transitions between the information sets

Most of the SOTA algorithms were developed thanks to poker application – **competition in dynamic security games**?

Challenges

It is not an easy task.

Start with the baseline algorithm:

- Provides a clear benchmark for further improvement.
- What if the baseline already scales well and all that is needed is a minor modification?

Typical bottlenecks:

- history dependent strategies (perfect recall) – Does the game (outcomes, strategies) really depends on the entire history?
- domain-dependent simplifications (which states/strategies can be safely removed?)

Try to generalize the methods.

Maybe a better, fundamentally different algorithm (or a representation) can be designed.

Thank you

bosansky@fel.cvut.cz

References I

[Bosansky et al., 2014] Bosansky, B., Kiekintveld, C., Lisy, V., and Pechoucek, M. (2014).

An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information.

Journal of Artificial Intelligence Research, 2014.

[Bowling et al., 2015] M. Bowling, N. Burch, M. Johanson, O. Tammelin.

Heads-up limit holdem poker is solved.

Science 347 (6218) (2015) 145–149.

[Brown and Sandholm, 2016] Brown, N. and Sandholm, T. (2016).

Strategy-Based Warm Starting for Regret Minimization in Games.

In *Proceedings of AAAI Conference on Artificial Intelligence*.

[Brown and Sandholm, 2017] Brown, N. and Sandholm, T. (2017).

Safe and Nested Subgame Solving for Imperfect-Information Games

In *Proceedings of 31st Conference on Neural Information Processing Systems (NIPS 2017)*.

References II

[Burch et al., 2014] Burch, N., Johanson, M., and Bowling, M. (2014).
Solving Imperfect Information Games Using Decomposition.
In *Proceedings of AAAI Conference on Artificial Intelligence*.

[Cerny et al., 2018] Cerny, J., Bosansky, B., and Kiekintveld, C. (2018).
Incremental Strategy Generation for Stackelberg Equilibria in Extensive-Form Games.
In *Proceedings of the ACM Conference on Economics and Computation*.

[Cermak et al., 2018] J. Čermák, B. Bošanský, K. Horák, V. Lisý, M. Pěchouček. (2018)
Approximating maxmin strategies in imperfect recall games using a-loss recall property.
International Journal of Approximate Reasoning 93:290–326.

[Ciancarini and Favini, 2010] Ciancarini, P. and Favini, G. P. (2010).
Monte Carlo tree search in Kriegspiel.
Artificial Intelligence, 174:670–684.

References III

[Chatterjee and Doyen, 2014] Chatterjee, K., and Doyen, L. (2014)
Partial-observation stochastic games: How to win when belief fails.
ACM Transactions on Computational Logic (TOCL) 15(2):16.

[Ghosh, et al., 2004] Ghosh, M. K.; McDonald, D.; and Sinha, S. (2004)
Zero-Sum Stochastic Games with Partial Information.
Journal of Optimization Theory and Applications 121(1):99–118.

[Hoda et al., 2010] S. Hoda, A. Gilpin, J. Peña, T. Sandholm, (2010)
Smoothing Techniques for Computing Nash Equilibria of Sequential Games.
Mathematics of Operations Research 35 (2) (2010) 494–512.

[Horak et al., 2017] Horak, K., Bosansky, B., and Pechoucek, M. (2017).
Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games
In *Proceedings of AAAI Conference on Artificial Intelligence.*

References IV

[Johanson et al., 2011] Johanson, M., Bowling, M., Waugh, K., and Zinkevich, M. (2011).

Accelerating best response calculation in large extensive games.

In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 258–265.

[Koller and Megiddo, 1992] D. Koller, N. Megiddo. (1992)

The Complexity of Two-person Zero-sum Games in Extensive Form,

Games and Economic Behavior 4:528–552.

[Kroer et al., 2018] Kroer, C., Waugh, K., Klnc-Karzan, F., Sandholm, T. (2018).

Faster algorithms for extensive-form game solving via improved smoothing functions.

Mathematical Programming, 1–33.

References V

[Lanctot, 2013] Lanctot, M. (2013).
Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision Making in Large Extensive-Form Games.
PhD thesis, University of Alberta.

[Moravcik et al., 2017] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, Deepstack: Expert-level Artificial Intelligence in Heads-up No-limit Poker, Science.

[Shoham and Leyton-Brown, 2009] Shoham, Y. and Leyton-Brown, K. (2009).
Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations.
Cambridge University Press.

[Tammelin, O. 2014] O. Tammelin, (2014)
CFR+,
CoRR, abs/1407.5042.

References VI

[Wichardt, 2008] Wichardt, P. C. (2008).

Existence of nash equilibria in finite extensive form games with imperfect recall: A counterexample.

Games and Economic Behavior, 63(1):366–369.

[Zinkevich et al., 2008] M. Zinkevich, M. Johanson, M. H. Bowling, C. Piccione. (2007)

Regret minimization in games with incomplete information.

In *Advances in Neural Information Processing Systems*, pp. 1729–1736.