Iterated Regret Minimization in Game Graphs

Emmanuel Filiot, Tristan Le Gall, Jean-François Raskin

Université Libre de Bruxelles (ULB)

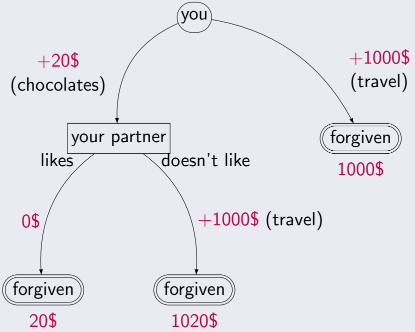
CIRM/ LMeASI, 10 décembre 2010

Some (Almost) Real Life Example

Input data s

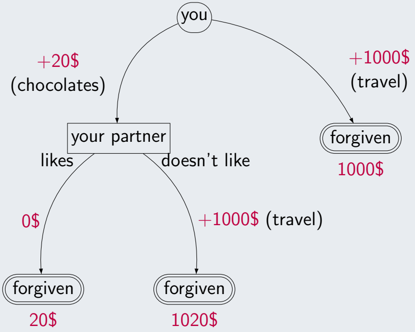
- you spent too much time working on your MFCS paper
- your partner is angry and you want to be forgiven
- your partner likes travelling
- you don't know if she/he likes chocolates (even Belgian ones)
- Cost(chocolates) << Cost(travelling)

Game Tree



Some (Almost) Real Life Example

Game Tree

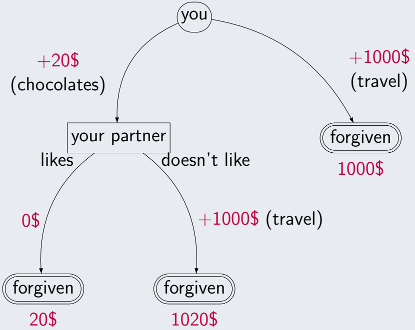


Minmax Backward Induction

Offer a travel and pay 1000$

Some (Almost) Real Life Example

Game Tree



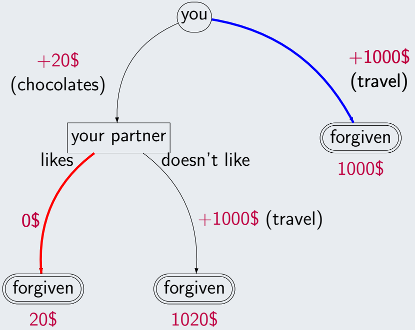
Regret Minimization

$$\text{regret}_1(\lambda_1, \lambda_2) \;=\; \text{cost}(\lambda_1, \lambda_2) - \text{best-response}(\lambda_2)$$

you (λ_1) / partner (λ_2)	travel	chocolates
likes chocolates		
doesn't		

Some (Almost) Real Life Example

Game Tree

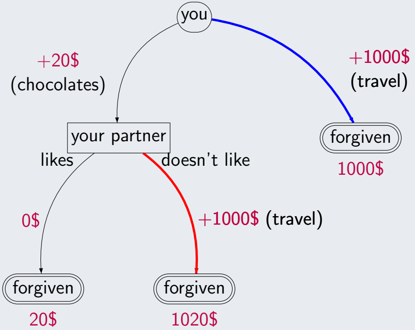


Regret Minimization

$$\text{regret}_1(\lambda_1, \lambda_2) \;=\; \text{cost}(\lambda_1, \lambda_2) - \text{best-response}(\lambda_2)$$

you (λ_1) / partner (λ_2)	travel	chocolates
likes chocolates	1000 - 20 = 980	
doesn't		

Game Tree



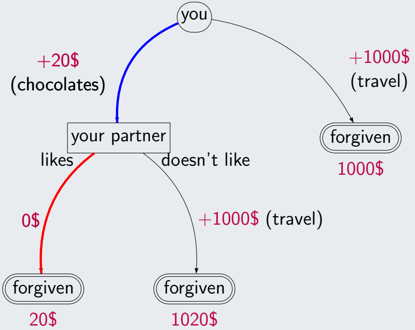
Regret Minimization

$$\text{regret}_1(\lambda_1, \lambda_2) \;=\; \text{cost}(\lambda_1, \lambda_2) - \text{best-response}(\lambda_2)$$

you (λ_1) / partner (λ_2)	travel	chocolates
likes chocolates	100 - 20 = 980	
doesn't	1000 - 1000 = 0	

Some (Almost) Real Life Example

Game Tree



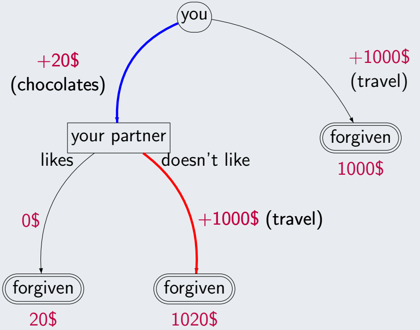
Regret Minimization

$$\text{regret}_1(\lambda_1, \lambda_2) \;=\; \text{cost}(\lambda_1, \lambda_2) - \text{best-response}(\lambda_2)$$

partner (λ_2) \ you (λ_1)	travel	chocolates
likes chocolates	1000 - 20 $=$ 980	20 - 20 $=$ 0
doesn't	1000 - 1000 $=$ 0	

Some (Almost) Real Life Example

Game Tree



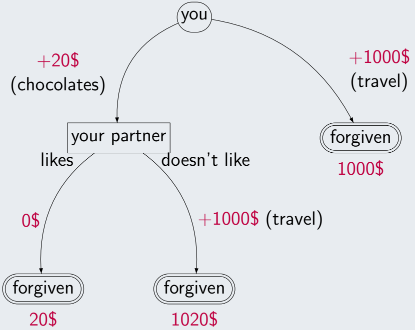
Regret Minimization

$$\text{regret}_1(\lambda_1, \lambda_2) \;=\; \text{cost}(\lambda_1, \lambda_2) - \text{best-response}(\lambda_2)$$

you (λ_1) / partner (λ_2)	travel	chocolates
likes chocolates	1000 - 20 = 980	20 - 20 = 0
doesn't	1000 - 1000 = 0	1020 - 1000=20

Some (Almost) Real Life Example

Game Tree



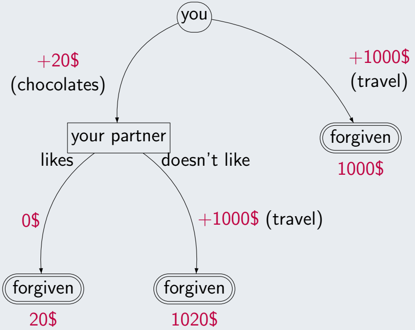
Regret Minimization

$$\text{regret}_1(\lambda_1, \lambda_2) \;=\; \text{cost}(\lambda_1, \lambda_2) - \text{best-response}(\lambda_2)$$

partner (λ_2) \ you (λ_1)	travel	chocolates
likes chocolates	980	0
doesn't	0	20

Some (Almost) Real Life Example

Game Tree

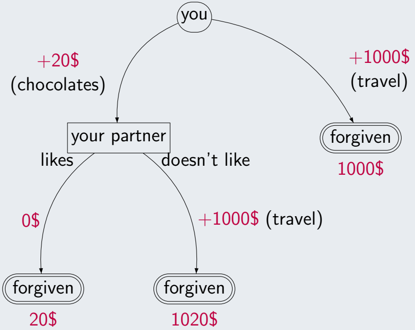


Regret Minimization

$$\text{regret}_1(\lambda_1) \;=\; \max_{\lambda_2}\;(\text{cost}(\lambda_1,\lambda_2) - \text{best-response}(\lambda_2))$$

you (λ_1) / partner (λ_2)	travel	chocolates
likes chocolates	980	0
doesn't	0	20

Game Tree



Regret Minimization

$$\text{regret}_1 \;=\; \min_{\lambda_1} \; \max_{\lambda_2} \; (\text{cost}(\lambda_1, \lambda_2) - \text{best-response}(\lambda_2))$$

you (λ_1) / partner (λ_2)	travel	chocolates
likes chocolates	980	0
doesn't	0	20

Related Work

Regret Minimization

- first introduced in the 50's [Savage,51] [Niehans,48]
- in general, a cost function c_i models what Player i pays
- decision under uncertainty
- choose a *good* strategy no matter the adversary does

Regret Minimization

- first introduced in the 50's [Savage,51] [Niehans,48]
- in general, a cost function c_i models what Player *i* pays
- decision under uncertainty
- choose a *good* strategy no matter the adversary does

Iterated Regret Minimization

- regret minimization selects a set of strategies for each player
- can be iterated
- new solution concept proposed by [Halpern and Pass, IJCAI'09] in **strategic games** where the costs are given by a matrix
- iterated regret more reasonable than Nash equilibria for various classes of games (Centipede Game, Travellers dilemma,...)

Comparison with Nash Equilibria

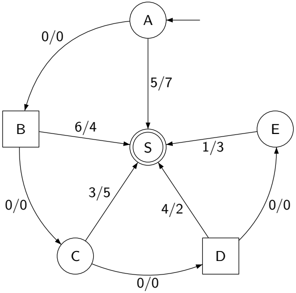
Definition

A couple of strategies $\langle \lambda_1, \lambda_2 \rangle$ *forms a Nash equilibrium if:*

- $c_1(\lambda_1, \lambda_2) = \min_{\lambda_1^*} c_1(\lambda_1^*, \lambda_2)$
- $c_2(\lambda_1, \lambda_2) = \min_{\lambda_2^*} c_1(\lambda_1, \lambda_2^*)$

Remember: $c_1(\lambda_1, \lambda_2)$ represents what Player 1 pays

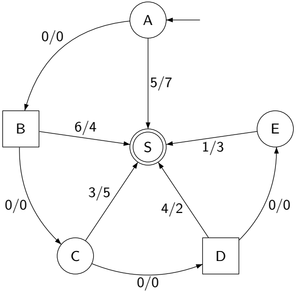
The Centipede Game



Principle

- At each turn, the active player can choose to stop the game, or to continue
- If she stops the game, a player pays less than if the other player stops the game immediatly after
- But if both players continue, they both pay less (cooperation is rewarded)

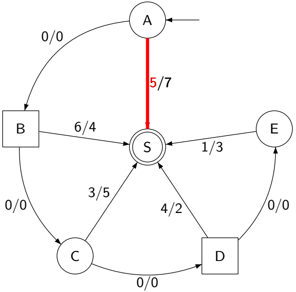
The Centipede Game



Regret Minimization vs Nash Equilibrium

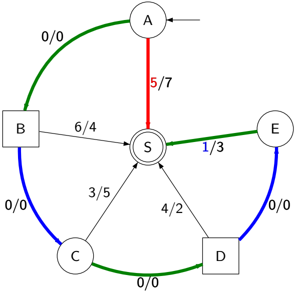
- Nash equilibrium suggests to stop the game immediately

The Centipede Game



Regret Minimization vs Nash Equilibrium

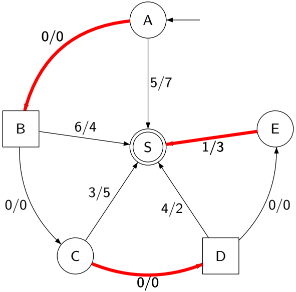
- Nash equilibrium suggests to stop the game immediately
- $\mathrm{reg}_1(A \to S) = 5$



Regret Minimization vs Nash Equilibrium

- Nash equilibrium suggests to stop the game immediately
- $\mathrm{reg}_1(A \rightarrow S) = 5 - 1 = 4$

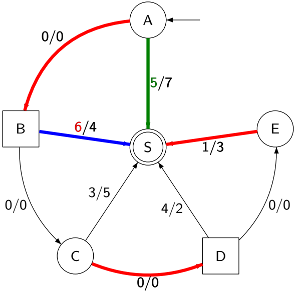
The Centipede Game



Regret Minimization vs Nash Equilibrium

- Nash equilibrium suggests to stop the game immediately
- $\mathrm{reg}_1(A \rightarrow S) = 5 - 1 = 4$
- $\mathrm{reg}_1(A \rightarrow B; C \rightarrow D; E \rightarrow S) =$

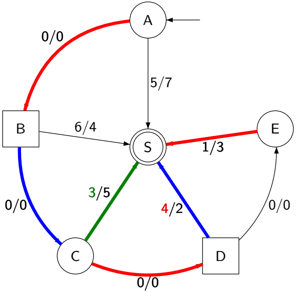
The Centipede Game



Regret Minimization vs Nash Equilibrium

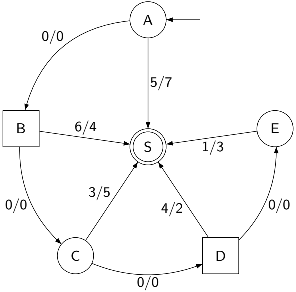
- Nash equilibrium suggests to stop the game immediately
- $\text{reg}_1(A \to S) = 5 - 1 = 4$
- $\text{reg}_1(A \to B; C \to D; E \to S) \;=\; 6 - 5$ or

The Centipede Game



Regret Minimization vs Nash Equilibrium

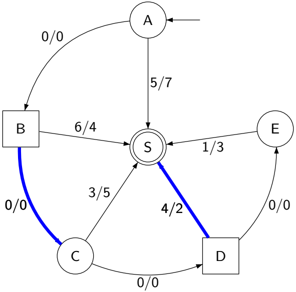
- Nash equilibrium suggests to stop the game immediately
- $\mathrm{reg}_1(A \rightarrow S) = 5 - 1 = 4$
- $\mathrm{reg}_1(A \rightarrow B; C \rightarrow D; E \rightarrow S) \ = \ 6 - 5 \text{ or } 4 - 3 \ = \ 1$



Regret Minimization vs Nash Equilibrium

- Nash equilibrium suggests to stop the game immediately
- $\text{reg}_1(A \to S) = 5 - 1 = 4$
- $\text{reg}_1(A \to B; C \to D; E \to S) \; = \; 6 - 5 \text{ or } 4 - 3 \; = \; 1$
- $\text{reg}_2(B \to S) = 4 - 2 = 2$

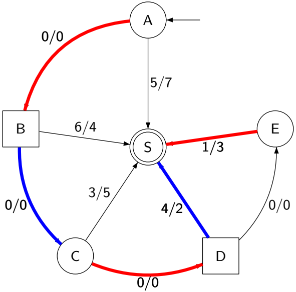
The Centipede Game



Regret Minimization vs Nash Equilibrium

- Nash equilibrium suggests to stop the game immediately
- $\text{reg}_1(A \to S) = 5 - 1 = 4$
- $\text{reg}_1(A \to B; C \to D; E \to S) \; = \; 6 - 5 \text{ or } 4 - 3 \; = \; 1$
- $\text{reg}_2(B \to S) = 4 - 2 = 2$
- $\text{reg}_2(B \to C; D \to S) = 5 - 4 = 1$

The Centipede Game



Regret Minimization vs Nash Equilibrium

- Nash equilibrium suggests to stop the game immediately
- $\mathrm{reg}_1(A \to S) = 5 - 1 = 4$
- $\mathrm{reg}_1(A \to B; C \to D; E \to S) \; = \; 6 - 5$ or $4 - 3 \; = \; 1$
- $\mathrm{reg}_2(B \to S) = 4 - 2 = 2$
- $\mathrm{reg}_2(B \to C; D \to S) = 5 - 4 = 1$
- $\mathrm{reg}_1^\infty(A \to B; C \to D; E \to S) = 0$ and $\mathrm{reg}_2^\infty(B \to C; D \to S) = 0$.

- Nash equilibria: the players implicitly need to know the other player strategy
- Regret minimization: each player wants to use a *good* strategy no matter the other player does
- Regret minimization is a *robust* solution concept. In average (#nodes$\leq$10000, max weight$\leq$30):
    - Regret: payoffs are 40% worst when the other player changes her strategy while you stick to your strategy
    - Subgame Perfect Nash Equilibria: 200% to 500% worst

Our Work

- 2-players non-zero sum game given implicitely by a finite graph
- Partition of the vertices of the graph
- reachability objectives
- edges (or target nodes) are weighted: cost to the first visit to a target node
- tree arenas and finite graph arenas
- algorithms to compute the (iterated) regret and to select strategies
- extends to n players

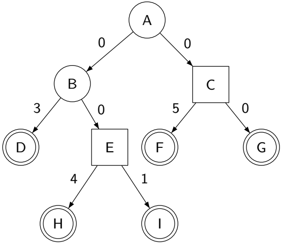
Our aim is to compute:

$$\begin{aligned} \text{regret}_1 &= \min_{\lambda_1} \max_{\lambda_2} [\text{cost}_1(\lambda_1, \lambda_2) - \text{best-response}(\lambda_2)] \\ &= \min_{\lambda_1} \max_{\lambda_2} \left[\text{cost}_1(\lambda_1, \lambda_2) - \min_{\lambda_1^*} \text{cost}_1(\lambda_1^*, \lambda_2)\right] \end{aligned}$$

and the iteration of the operator that deletes strictly dominated strategies.

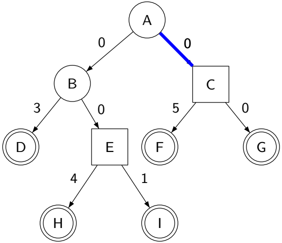
	trees	target-weighted graphs	edge-weighted graphs
#strategies	exp	∞	∞
regret	$O(n)$	PTime	ExpTime
iterated regret	$O(n^2)$	?	Pseudo-ExpTime (with weigths > 0)

Regret Minimization in Trees



To maximize Player 1's regret, Player 2 should cooperate in the non-reachable subtrees

Example

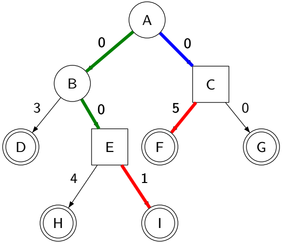


To maximize Player 1's regret, Player 2 should cooperate in the non-reachable subtrees

Regrets for Player 1

- $\text{reg}_1(A \to C) \; =$

Example

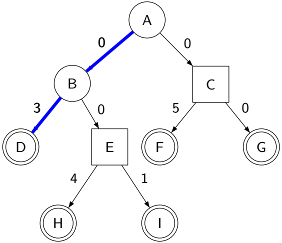


To maximize Player 1's regret, Player 2 should cooperate in the non-reachable subtrees

Regrets for Player 1

- $\text{reg}_1(A \to C) = 5 - 1 = 4$

Example

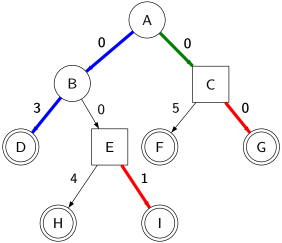


To maximize Player 1's regret, Player 2 should cooperate in the non-reachable subtrees

Regrets for Player 1

- $\text{reg}_1(A \to C) \;=\; 5 - 1 \;=\; 4$
- $\text{reg}_1(A \to B; B \to D) \;=$

Example



To maximize Player 1's regret, Player 2 should cooperate in the non-reachable subtrees

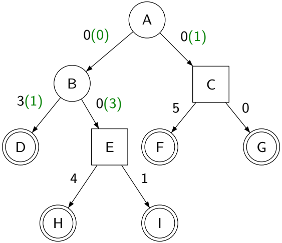
Regrets for Player 1

- $\text{reg}_1(A \to C) \ = \ 5 - 1 \ = \ 4$
- $\text{reg}_1(A \to B; B \to D) \ = \ 3 - 0 \ = \ 3$

Remarks

- regret is not preserved by subtrees, no bottom-up algorithm
- we must know the best alternative seen so far
- we transform the game to obtain a min-max game

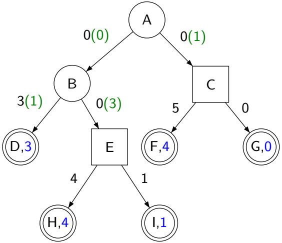
Algorithm



Steps of Computation

1. For each root-to-leaf path, compute the regret of this path:

   (cost of the path) − (best alternative along this path)

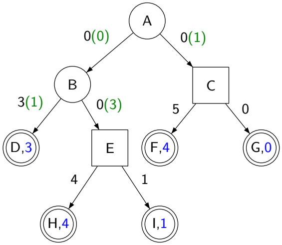


Steps of Computation

1. For each root-to-leaf path, compute the regret of this path:

$$(\text{cost of the path}) - (\text{best alternative along this path})$$

Algorithm



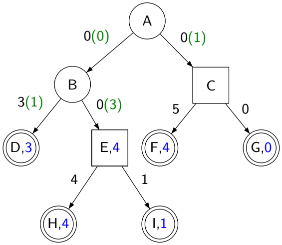
Steps of Computation

1. For each root-to-leaf path, compute the regret of this path:

$$(\text{cost of the path}) \;-\; (\text{best alternative along this path})$$

2. Solve a min-max game in the tree T

$$\begin{array}{lcll} \mathrm{reg}_T(s(T_1, T_2)) & = & \min(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P1's position} \\ \mathrm{reg}_T(s(T_1, T_2)) & = & \max(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P2's position} \\ \mathrm{reg}_T(s) & = & \mathrm{reg}_T(\mathrm{path}(s_0, s)) & \text{if } s \text{ is leaf} \end{array}$$



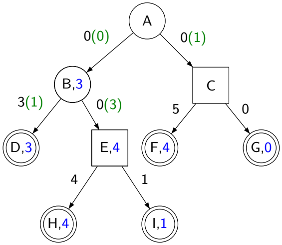
Steps of Computation

1. For each root-to-leaf path, compute the regret of this path:

   (cost of the path) $-$ (best alternative along this path)

2. Solve a min-max game in the tree T

$$
\begin{array}{lcll}
\mathrm{reg}_T(s(T_1, T_2)) & = & \min(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P1's position} \\
\mathrm{reg}_T(s(T_1, T_2)) & = & \max(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P2's position} \\
\mathrm{reg}_T(s) & = & \mathrm{reg}_T(\mathrm{path}(s_0, s)) & \text{if } s \text{ is leaf}
\end{array}
$$



Steps of Computation

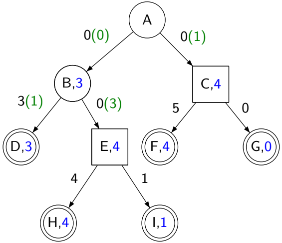
1. For each root-to-leaf path, compute the regret of this path:

$$(\text{cost of the path}) \;-\; (\text{best alternative along this path})$$

2. Solve a min-max game in the tree T

$$\begin{array}{lcll} \mathrm{reg}_T(s(T_1, T_2)) & = & \min(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P1's position} \\ \mathrm{reg}_T(s(T_1, T_2)) & = & \max(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P2's position} \\ \mathrm{reg}_T(s) & = & \mathrm{reg}_T(\mathrm{path}(s_0, s)) & \text{if } s \text{ is leaf} \end{array}$$

Algorithm



Steps of Computation

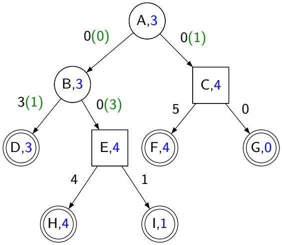
1. For each root-to-leaf path, compute the regret of this path:

$$\text{(cost of the path)} \;-\; \text{(best alternative along this path)}$$

2. Solve a min-max game in the tree T

$$\begin{array}{lcll}
\mathrm{reg}_T(s(T_1, T_2)) & = & \min(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P1's position} \\
\mathrm{reg}_T(s(T_1, T_2)) & = & \max(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P2's position} \\
\mathrm{reg}_T(s) & = & \mathrm{reg}_T(\mathrm{path}(s_0, s)) & \text{if } s \text{ is leaf}
\end{array}$$

Algorithm



Steps of Computation

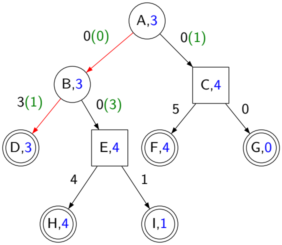
1. For each root-to-leaf path, compute the regret of this path:

$$\text{(cost of the path)} - \text{(best alternative along this path)}$$

2. Solve a min-max game in the tree T

$$\begin{array}{lcll}
\mathrm{reg}_T(s(T_1, T_2)) & = & \min(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P1's position} \\
\mathrm{reg}_T(s(T_1, T_2)) & = & \max(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) & \text{if } s \text{ is a P2's position} \\
\mathrm{reg}_T(s) & = & \mathrm{reg}_T(\mathrm{path}(s_0, s)) & \text{if } s \text{ is leaf}
\end{array}$$

Algorithm



Steps of Computation

1. For each root-to-leaf path, compute the regret of this path:

   (cost of the path) $-$ (best alternative along this path)

2. Solve a min-max game in the tree T

$$
\begin{aligned}
\mathrm{reg}_T(s(T_1, T_2)) &= \min(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) && \text{if } s \text{ is a P1's position}\\
\mathrm{reg}_T(s(T_1, T_2)) &= \max(\mathrm{reg}_T(T_1), \mathrm{reg}_T(T_2)) && \text{if } s \text{ is a P2's position}\\
\mathrm{reg}_T(s) &= \mathrm{reg}_T(\mathrm{path}(s_0, s)) && \text{if } s \text{ is leaf}
\end{aligned}
$$

Algorithm

1. compact representation of strategies that minimize the regret: delete non-optimal edges
2. keep only the reachable states
3. do the same for Player 2
4. compute the regret in the new tree
5. **goto** 1 until convergence

Iterated Regret Minimization

Algorithm

1. compact representation of strategies that minimize the regret: delete non-optimal edges
2. keep only the reachable states
3. do the same for Player 2
4. compute the regret in the new tree
5. **goto** 1 until convergence

Overall complexity: quadratic time

What about graphs?

Target-Weighted Graphs

Description of the arena

- Finite graph arena
- Costs are located on the target nodes (alternative definition: on the edges leading to a target node)

Reduction to a min-max game

- Cut the losing part of the graph
- Enrich the graph with best alternative information
- Remark: there are as many possible best alternatives as the number of target nodes
- $\rightarrow$ PTime Complexity

Remember

- Infinite number of strategies
- Each player has a set of target states, payoffs are on the edges
- Utility of an outcome: sum of edge payoffs until it first reaches a target state
- If all payoffs are strictly positive integers, pseudo-polynomial time algorithm (unfolding $\rightarrow$ tree arena)

Challenges

Edge-Weighted Graphs

Remember

- Infinite number of strategies
- Each player has a set of target states, payoffs are on the edges
- Utility of an outcome: sum of edge payoffs until it first reaches a target state
- If all payoffs are strictly positive integers, pseudo-polynomial time algorithm (unfolding $\rightarrow$ tree arena)

Challenges

- Loops !

Edge-Weighted Graphs

Remember

- Infinite number of strategies
- Each player has a set of target states, payoffs are on the edges
- Utility of an outcome: sum of edge payoffs until it first reaches a target state
- If all payoffs are strictly positive integers, pseudo-polynomial time algorithm (unfolding $\rightarrow$ tree arena)

Challenges

- Loops !
- Player i's choices are no longer important (for her) when her target is reached

Edge-Weighted Graphs

Remember

- Infinite number of strategies
- Each player has a set of target states, payoffs are on the edges
- Utility of an outcome: sum of edge payoffs until it first reaches a target state
- If all payoffs are strictly positive integers, pseudo-polynomial time algorithm (unfolding $\rightarrow$ tree arena)

Challenges

- Loops !
- Player i's choices are no longer important (for her) when her target is reached
- Avoid subset construction

Sketch of the Reduction to a Tree Arena

Loops no more

- Loop-free strategies are sufficient to minimize regrets ...
- ... but we must consider all strategies when computing iterated regret
- Assuming payoffs $> 0 \longrightarrow$ eliminate loop issues

Sketch of the Reduction to a Tree Arena

Transformation

- Unfold the graph and detect loops
- Nodes of the unfolding $\langle n, u_1, u_2, b_1, b_2 \rangle$ where
  - n is a node of the original graph
  - u_i represents the utility for Player i (sum of payoffs) up to this node
  - b_i is a boolean remembering whether Player i's target has been reached

Sketch of the Reduction to a Tree Arena

Transformation

- Unfold the graph and detect loops
- Nodes of the unfolding $\langle n, u_1, u_2, b_1, b_2 \rangle$ where
  - n is a node of the original graph
  - u_i represents the utility for Player i (sum of payoffs) up to this node
  - b_i is a boolean remembering whether Player i's target has been reached
- Transition function: $\langle n, u_1, u_2, b_1, b_2 \rangle \longrightarrow \langle n', u'_1, u'_2, b'_1, b'_2 \rangle$ if:
  - there is an edge $n \to n'$
  - $u'_i =$ if b_i then u_i else $u_i + \mathrm{c}_i(n \to n')$
  - $b'_i = b_i$ or n' is one of Player i's target states

Sketch of the Reduction to a Tree Arena

Transformation

- Unfold the graph and detect loops
- Nodes of the unfolding $\langle n, u_1, u_2, b_1, b_2\rangle$ where
  - n is a node of the original graph
  - u_i represents the utility for Player i (sum of payoffs) up to this node
  - b_i is a boolean remembering whether Player i's target has been reached
- Transition function: $\langle n, u_1, u_2, b_1, b_2\rangle \longrightarrow \langle n', u'_1, u'_2, b'_1, b'_2\rangle$ if:
  - there is an edge $n \rightarrow n'$
  - $u'_i =$ if b_i then u_i else $u_i + \mathrm{c}_i(n \rightarrow n')$
  - $b'_i = b_i$ or n' is one of Player i's target states
- Loop detection: if $u_i > M$ where M is the maximal utility on a loop-free paths of the graph
- Stop unfolding when $b_1 \&\& b_2$ or when a loop is detected
- Utility of a leaf: $\mathrm{c}_i(\langle n, u_1, u_2, b_1, b_2\rangle) =$ if b_i then u_i else $+\infty$

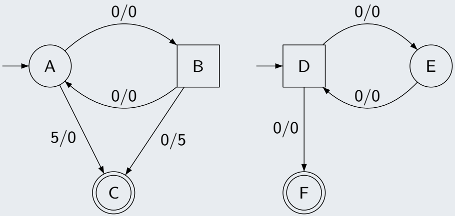
Sketch of the Reduction to a Tree Arena

Transformation

- Unfold the graph and detect loops
- Nodes of the unfolding $\langle n, u_1, u_2, b_1, b_2 \rangle$ where
  - n is a node of the original graph
  - u_i represents the utility for Player i (sum of payoffs) up to this node
  - b_i is a boolean remembering whether Player i's target has been reached
- Transition function: $\langle n, u_1, u_2, b_1, b_2 \rangle \longrightarrow \langle n', u_1', u_2', b_1', b_2' \rangle$ if:
  - there is an edge $n \to n'$
  - $u_i' =$ if b_i then u_i else $u_i + \mathrm{c}_i(n \to n')$
  - $b_i' = b_i$ or n' is one of Player i's target states
- Loop detection: if $u_i > M$ where M is the maximal utility on a loop-free paths of the graph
- Stop unfolding when $b_1 \&\& b_2$ or when a loop is detected
- Utility of a leaf: $\mathrm{c}_i(\langle n, u_1, u_2, b_1, b_2 \rangle) =$ if b_i then u_i else $+\infty$
- Overall complexity: pseudo-polynomial time and space

Future work

- Extend the class of graphs: iterated regret minimization in general game graphs
  - problem: 0-cost loops
  - challenge: add fairness condition to the graph



- Extend the quantitative measure: quantitative languages (Chatterjee, Doyen, Henzinger, 2008), ...

Thank you