Safe Search for Stackelberg Equilibria in Extensive-Form Games

Chun Kai Ling[1], Noam Brown[2]

[1] Carnegie Mellon University

[2] Facebook AI Research

chunkail@cs.cmu.edu, noambrown@fb.com

Extensive Form Games (EFG)

Game tree use to model settings with sequential interactions

Imperfect information modeled by *information sets*

- Perfect recall: never forget your own past actions or observations

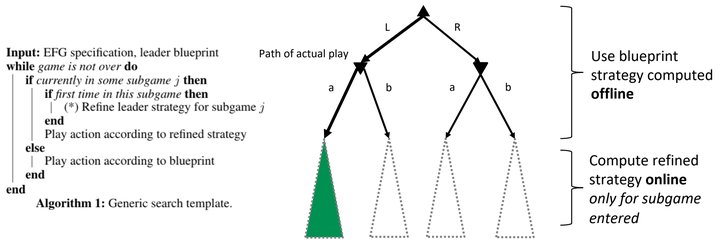
Very general formulation with many applications

- e.g., Poker (and bots Libratus and DeepStack)

Safe Search

A crucial component of successful bots these days is **search**. In perfect information games like chess, one applies search to a limited depth and continues search in actual play. Hence search is only initiated from states encountered in actual play. The natural analogue in games of imperfect information is for a **blueprint** (typically from a simple abstraction of the original) to be computed offline and followed in actual play. Upon entering a subgame, a **refinement** is computed **online** only for the subgame entered. However, unlike perfect information games,

- Different distribution over states, depending on the previous actions taken by both players.
- Search can be **unsafe**: Performing search carelessly can result in a strategy which is *worse than the blueprint.*



Strong Stackelberg Equilibria (SSE)

Used 2-player *general-sum* games with a distinguished *leader* (P1) and *follower* (P2)

Leader enjoys commitment privileges

- Commits to a (mixed) strategy before the game starts

Follower best-responds to leader's strategy

- Strong Stackelberg Equilibrium (SSE): In the event of a tie, follower chooses strategy which yields highest payoff to the leader.

Applications in Security Games

- Strong Stackelberg Equilibrium (SSE): In the event of a tie, follower chooses strategy which yields highest payoff to the leader.

Solving SSEs in EFGs is NP-hard in general

- Special cases: games without chance, perfect information, normal form games
- Existing methods rely on heuristics/strategy generation. No online method exists yet.

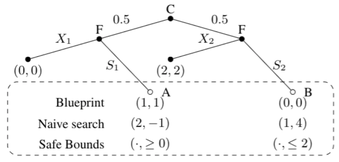
Our contributions

Safe search is primarily applied to *zero-sum games*. Our work applies safe search to **finding SSE in extensive form games,** where the leader is the player performing search. We also show that the refinement step in our algorithm can be reformulated as finding *the SSE of a modified game,* meaning our method is complemented by fast offline solvers.

Causes of Unsafe Search

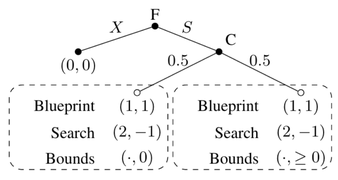
Important: Follower best responds to the entire subgame search algorithm, and *not* the blueprint. That is, follower knows the "source code" of the leader and best responds to it.

(I) Follower changes pre-subgame actions



- Blueprint: follower plays (S1, X2), leader payoff =1.5
- Under Naïve Search, if follower continues playing (S1, X2), leader payoff = 2. However, follower best-responds and switches to (X1, S2), leader payoff = 0.5
- Blueprint performs better than refinement---unsafe
- Problem: follower is incentivized to deviate from blueprint best response because of search
- Can prevent this by enforcing upper and lower bounds on *follower* payoffs.

(II) Multiple subgames



- Suppose leader applies subgame search *only* to the left subgame. Under the blueprint,, follower plays S, leader payoff = follower payoff 1. Under refinement, leader payoff = 1.5, follower payoff = 0. Game appears to be safe (tiebreaks favor leader).
- However, follower knows that if search is applied to the left subgame, it would also have been applied to the right subgame if it was reached.
- Hence, search is applied to *all* subgames, follower payoff = -1, leading to the follower choosing X over C---unsafe.
- Well known issue in zero-sum settings.
- Problem can be averted by again enforcing bounds.

Our algorithm

Phase 1: Bounds computation

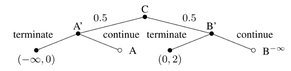
- Computes bounds that guarantee that follower not deviate from blueprint prior to subgames
- Done recursively top-down and may contain upper and lower bounds. Lower bounds are for information sets which are traversed, while upper bounds otherwise.
- Requires a single pass of a player's treeplex (no bigger than game tree)

Phase 2: Computing a refinement

- Solve the SSE of subgame which respects aforementioned bounds.
- Direct method: Plug linear constraints into mathematical programs.
- Gadget method: Reformulate the constrained SSE problem as a solution to another SSE.

Bounds generation using a gadget

Important: Follower responds to the entire subgame search algorithm, and *not* the blueprint. That is, follower knows the "source code" of the leader and best responds to it.



To enforce lower bounds:
Insert auxiliary state(s) A' with (−∞ , <bound>) leader payoffs

If follower payoff is less than <bound>, follower terminates and gives leader −∞

Enforcing upper bounds:
Insert auxiliary state(s) B' with (0, <bound>) leader payoffs

If follower payoff is greater than <bound>, follower continues and gives −∞ leader payoff

Experimental Setup

Compare with MILP-based full-game solver run for max of 1000s

- If the solver has not run to completion, we take the best incumbent solution

For evaluation, run subgame search (refinement step) using the direct method

- Done for *all* subgames for maximum of 100s each.
- Combine solutions together to obtain approximate full-game solution
- Done only for purposes of evaluation---in practice, will only run refinement in subgames that are encountered in actual play.

All MILPs were solved using Gurobi.

Initialization of strategies (both online and offline) to be blueprint

- In larger games, full-game solver does not even find a *feasible* solution in 1000s without the blueprint

2-Stage Game

Randomly generated Markov game (almost) with 2 stages

Blueprint: SSE of first stage alone, randomly take actions in second stage.

Our method performs better for the leader for the larger games.

Sanity check: our method performs better than blueprint in all cases.

n	M	m	κ	Blueprint	Ours	Full-game
2	2	2	0	1.2945	1.4472	**1.4778**
			0.1	1.2945	1.4477	**1.4779**
			0.9	1.2951	1.4519	**1.4790**
2	10	10	0	1.1684	1.6179	**1.6186**
			0.1	1.1689	1.6180	**1.6186**
			0.9	1.1723	1.6183	**1.6190**
2	100	100	0	1.1730	**1.6696**	1.3125
			0.1	1.1729	**1.6696**	1.2652
			0.9	1.1722	**1.6696**	1.4055
5	100	100	0	1.3756	**1.8722**	1.4074
			0.1	1.3756	**1.8723**	1.4073
			0.9	1.3752	**1.8723**	1.4534

Rake Poker

Simplified game of poker with n cards, 2 suits, and 2 betting rounds.

Rake: dealer takes 10% of pot every game.

Blueprint: Nash of zero-sum version of the game.

n	$\lvert\Sigma\rvert$	$\lvert\mathcal{I}\rvert$	Blueprint	Ours	Full-game
3	5377	2016	-0.1738	-0.1686	**-0.1335**
4	9985	3744	-0.1905	**-0.1862**	-0.1882
5	16001	6000	-0.2028	**-0.2003**	-0.2028
6	23425	8784	-0.1832	**-0.1780**	-0.1832
8	42497	15936	-0.1670	**-0.1609**	N/A

[1] Part of this work was conducted when the author was an intern at Facebook AI Research