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

Abstract

Stackelberg equilibrium is a solution concept in two-player games where the leader has commitment rights over the follower. In recent years, it has become a cornerstone of many security applications, including airport patrolling and wildlife poaching prevention. Even though many of these settings are sequential in nature, existing techniques pre-compute the entire solution ahead of time. In this paper, we present a theoretically sound and empirically effective way to apply search, which leverages extra online computation to improve a solution, to the computation of Stackelberg equilibria in general-sum games. Instead of the leader attempting to solve the full game upfront, an approximate "blueprint" solution is first computed offline and is then improved online for the particular subgames encountered in actual play. We prove that our search technique is guaranteed to perform no worse than the pre-computed blueprint strategy, and empirically demonstrate that it enables approximately solving significantly larger games compared to purely offline methods. We also show that our search operation may be cast as a smaller Stackelberg problem, making our method complementary to existing algorithms based on strategy generation.

1 Introduction

Strong Stackelberg equilibria (SSE) have found many uses in security domains, such as wildlife poaching protection (Fang et al. 2017) and airport patrols (Pita et al. 2008). Many of these settings, particularly those involving patrolling, are sequential by nature and are best represented as extensive-form games (EFGs). Finding a SSE in general EFGs is provably intractable (Letchford and Conitzer 2010). Existing methods convert the problem into a normal-form game and apply column or constraint generation techniques to handle the exponential blowup in the size of the normal-form game (Jain, Kiekintveld, and Tambe 2011). More recent methods cast the problem as a mixed integer linear program (MILP) (Bosansky and Cermak 2015). Current state-of-the-art methods build upon this by heuristically generating strategies, and thus avoid considering all possible strategies (Černỳ, Bošanskỳ, and Kiekintveld 2018).

All existing approaches for computing SSE are entirely offline. That is, they compute a solution for the entire game ahead of time and always play according to that offline solution. In contrast, **search** additionally leverages online computation to improve the strategy for the specific situations that come up during play. Search has been a key component for AI in single-agent settings (Lin 1965; Hart, Nilsson, and Raphael 1968), perfect-information games (Tesauro 1995; Campbell, Hoane Jr, and Hsu 2002; Silver et al. 2016, 2018), and zero-sum imperfect-information games (Moravčík et al. 2017; Brown and Sandholm 2017b, 2019). In order to apply search to two-player zero-sum imperfect-information games in a way that would not do worse than simply playing an offline strategy, **safe search** techniques were developed (Burch, Johanson, and Bowling 2014; Moravcik et al. 2016; Brown and Sandholm 2017a). Safe search begins with a **blueprint** strategy that is computed offline. The search algorithm then adds extra constraints to ensure that its solution is no worse than the blueprint (that is, that it approximates an equilibrium at least as closely as the blueprint). However, safe search algorithms have so far only been developed for two-player zero-sum games.

In this paper, we extend safe search to SSE computation in *general-sum* games. We begin with a blueprint strategy for the leader, which is typically some solution (computed offline) of a simpler abstraction of the original game. The leader follows the blueprint strategy for the initial stages of the game, but upon reaching particular subgames of the game tree, computes a refinement of the blueprint strategy *online*, which is then adopted for the rest of the game.

We show that with search, one can approximate SSEs in games much larger than purely offline methods. We also show that our search operation is itself solving a smaller SSE, thus making our method complementary to other methods based on strategy generation. We evaluate our method on a two-stage matrix game, the classic game of Goofspiel, and a larger, general-sum variant of Leduc hold'em. We demonstrate that in large games our search algorithm outperforms offline methods while requiring significantly less computation, and that this improvement increases with the size of the game. Our implementation is publicly available online: https://github.com/lingchunkai/SafeSearchSSE.

2 Background and Related Work

As is standard in game theory, we assume that the strategies of all players, including the algorithms used to compute

*Part of this work was conducted at Facebook AI Research.

those strategies, are common knowledge. However, the outcomes of stochastic variables are not known ahead of time.

EFGs model sequential interactions between players, and are typically represented as game trees in which each node specifies a state of the game where one player acts (except terminal nodes where no player acts). In two-player EFGs, there are two players, $\mathcal{P} = \{1, 2\}$. H is the set of all possible nodes h in the game tree, which are represented as sequences of actions (possibly chance). $A(h)$ is the set of actions available in node h and $\mathcal{P}(h) \in \mathcal{P} \cup \{c\}$ is the player acting at that node, where c is the chance player. If a sequence of actions leads from h to h', then we write $h \sqsubset h'$. We denote $Z \subseteq H$ to be the set of all terminal nodes in the game tree. For each terminal node z, we associate a payoff for each player, $u_i : Z \to \mathbb{R}$. For each node h, the function $\mathcal{C} : H \to [0, 1]$ is the probability of reaching h, assuming both players play to do so.

Nodes belonging to player $i \in \mathcal{P}$, i.e., $\{h \in H, \mathcal{P}(h) = i\}$ are partitioned into information sets $\mathcal{I}_i$. All nodes h belonging to the same information set $I_i \in \mathcal{I}_i$ are indistinguishable and players must behave the same way for all nodes in I_i. Furthermore, all nodes in the same information are required to have the same actions, if $h, h' \in I_i$, then $A(h) = A(h')$. Thus, we overload $A(I_i)$ to define the set of actions in I_i. We assume that the game exhibits *perfect recall*, i.e., players do not 'forget' past observations or own actions; for each player i, the information set I_i is preceded by a unique series of actions and information sets of i.

Sequence Form Representation. Strategies in games with perfect recall may be compactly represented in *sequence-form* (Von Stengel 1996). A *sequence* σ_i is an (ordered) list of actions taken by a single player i in order to reach node h. The *empty sequence* $\emptyset$ is the sequence without any actions. The set of all possible sequences achievable by player i is given by Σ_i. We write $\sigma_i a = \sigma_i'$ if a sequence $\sigma_i' \in \Sigma_i$ may be obtained by appending an action a to σ_i. With perfect recall, all nodes h in information sets $I_i \in \mathcal{I}_i$ may be reached by a unique sequence σ_i, which we denote by $\mathbf{Seq}_i(I_i)$ or $\mathbf{Seq}_i(h)$. Conversely, $\mathbf{Inf}_i(\sigma_i')$ denotes the information set containing the last action taken in σ_i'. Using the sequence form, mixed strategies are given by *realization plans*, $r_i : \Sigma_i \to \mathbb{R}$, which are distributions over sequences. Realization plans for sequences σ_i give the probability that this sequence of moves will be played, assuming all other players played such as to reach $\mathbf{Inf}_i(\sigma_i)$. Mixed strategies obey the sequence form constraints, $\forall i,\ r_i(\emptyset) = 1$, $\forall I_i \in \mathcal{I}_i, r_i(\sigma_i) = \sum_{a \in A_i(I_i)} r_i(\sigma_i a)$ and $\sigma_i = \mathbf{Seq}_i(I_i)$.

Sequence forms may be visualized using *treeplexes* (Hoda et al. 2010), one per player. Informally, a treeplex is a tree rooted at $\emptyset$ with subsequent nodes alternating between information sets and sequences, and are operationally useful for providing recursive implementations for common operations in EFGs such as finding best responses. Since understanding treeplexes is helpful in understanding our method, we provide a brief introduction in the Appendix.

Stackelberg Equilibria in EFGs. Strong Stackelberg Equilibria (SSE) describe games in which there is asymmetry in the commitment powers of players. Here, players 1 and 2 play the role of *leader* and *follower*, respectively. The leader is able to commit to a (potentially mixed) strategy and the follower best-responds to this strategy, while breaking ties by favoring the leader. By carefully commiting to a mixed strategy, the leader implicitly issues threats, and followers are made to best-respond in a manner favorable to the leader. SSE are guaranteed to exist and the value of the game for each player is unique. In one-shot games, a polynomial-time algorithm for finding a SSE is given by the multiple-LP approach (Conitzer and Sandholm 2006).

However, solving for SSE in EFGs in general-sum games with either chance or imperfect information is known to be NP-hard in the size of the game tree (Letchford and Conitzer 2010) due to the combinatorial number of pure strategies. Bosansky and Cermak (2015) avoid transformation to normal form and formulate a compact mixed-integer linear program (MILP) which uses a binary sequence-form follower best response variable to modestly-sized problems. More recently, Černý, Bošanský, and Kiekintveld (2018) propose heuristically guided incremental strategy generation.

Safe Search. For this paper, we adopt the role of the leader and seek to maximize his expected payoff under the SSE. We assume that the game tree may be broken into several disjoint *subgames*. For this paper, a subgame is defined as a set of states $H_{\text{sub}} \subseteq H$ such that (a) if $h \sqsubset h'$ and $h \in H_{\text{sub}}$ then $h' \in H_{\text{sub}}$, and (b) if $h \in I_i$ and $h \in H_{\text{sub}}$, then for all $h' \in I_i$, $h' \in H_{\text{sub}}$. Condition (a) implies that one cannot leave a subgame after entering it, while (b) ensures that information sets are 'contained' within subgames—if any history in an information set belongs to a subgame, then every history in that information set belongs to that subgame. For the j-th subgame H^j_{sub}, $\mathcal{I}^j_i \subseteq \mathcal{I}_i$ is the set of information sets belonging to player i within subgame j. Furthermore, let $\mathcal{I}^j_{i,\text{head}} \subseteq \mathcal{I}^j_i$ be the 'head' information sets of player i in subgame j, i.e., $I_i \in \mathcal{I}^j_{i,\text{head}}$ if and only if $\mathbf{Inf}_i(\mathbf{Seq}_i(I_i))$ does not exist or does not belong to $\mathcal{I}^j_i$. With a slight abuse of notation, let $I^j_{i,head}(z)$ be the (unique, if existent) information set in $\mathcal{I}^j_{i,\text{head}}$ preceding leaf z.

At the beginning, we are given a *blueprint* strategy for the leader, typically the solution of a smaller abstracted game. The leader follows the blueprint strategy in actual play until reaching some subgame. Upon reaching the subgame, the leader computes a refined strategy and follows it thereafter. The pseudocode is given in Algorithm 1. The goal of the paper is to develop effective algorithms for the refinement step (*). Algorithm 1 implicitly defines a leader strategy distinct from the blueprint. Crucially, this implies that the follower responds to this implicit strategy and not the blueprint. Search is said to be *safe* when the leader applies Algorithm 1 such that its expected payoff is no less than the blueprint, supposing the follower best responds to the algorithm.

3 Unsafe Search

To motivate our algorithm, we first explore how unsafe behavior may arise. **Naïve search** assumes that prior to entering a subgame, the follower plays the best-response to the

```
Input: EFG specification, leader blueprint
while game is not over do
    if currently in some subgame j then
        if first time in this subgame then
            (*) Refine leader strategy for subgame j
        end
        Play action according to refined strategy
    else
        Play action according to blueprint
    end
end
```

Algorithm 1: Generic search template.

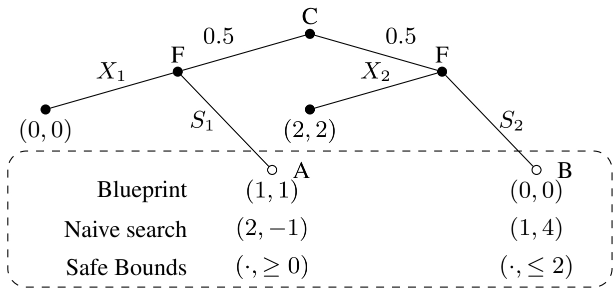

Figure 1: Unsafe naïve search and its game tree. Boxed regions denote subgames. Expected values for each player under (i) the blueprint strategy and its best response and (ii) under naïve search is shown in the box, as are bounds guaranteeing no change of follower strategies after refinement.

blueprint. For each subgame, the leader computes a normalized distribution of initial (subgame) states and solves a new game with initial states in obeying this distribution.

Consider the 2-player EFG in Figure 1, which begins with chance choosing each branch with equal probability. The follower then decides to e(**X**)it, or (**S**)tay, where the latter brings the game into a subgame, denoted by the dotted box. Upon reaching A, the follower recieves an expected value (EV) of 1 when best responding to the blueprint. Upon reaching B, the follower recieves an EV of 0 when best responding to the blueprint. Thus, under the blueprint strategy, the follower chooses to stay(exit) on the left(right) branches, and the expected payoff per player is $(1.5, 1.5)$.

Example 1. Suppose the leader performs naïve search in Figure 1, which improves the leader's EV in A from 1 to 2 but reduces the follower's EV in A from 1 to -1. The follower is aware that the leader will perform this search and thus chooses X_1 over S_1 even before entering A, since exiting gives a payoff of 0. Conversely, suppose this search improves the leader's EV in B from 0 to 1 and also improves the follower's EV from 0 to 4. Then the higher post-search payoff in B causes the follower to switch from X_2 to S_2. These changes cause the leader's EV to drop from 1.5 to 0.5. Thus, sticking to the blueprint is preferable to naïve search, which means naïve search is *unsafe*.[1]

[1]This counterexample is because of the general-sum nature of this game, and does not occur in zero-sum games.

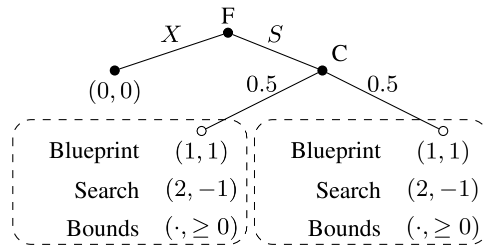

Figure 2: Failure when searching in multiple subgames.

Insight: Naïve search may induce changes in the follower's strategy *before* the subgame, which adjusts the probability of entering each state *within* the subgame. If one could enforce that in the refined subgame, payoffs to the follower in A remain no less than 0, then the follower would continue to stay, but possibly with leader payoffs greater than the blueprint. Similarly, we may avoid entering B by enforcing that follower payoff in B not exceed 2.

Example 2. Consider the game in Figure 2. Here, the follower chooses to exit or stay before the chance node is reached. If the follower chooses stay, then the chance node determines which of two identical subgames is entered. Under the blueprint, the follower receives an EV of 1 for choosing stay and an EV of 0 for choosing exit.

Suppose search is performed *only* in the left subgame, which decreases the follower's EV in that subgame from 1 to -1. Then, the expected payoff for staying is $(1.5, 0)$. The follower continues to favor staying (breaking ties in favor of the leader) and the leader's EV increases from 1.0 to 1.5.

Now suppose search is performed on whichever subgame is encountered during play. Then the follower knows that his EV for staying will be -1 regardless of which subgame is reached, and thus will exit. Exiting decreases the leader's payoff to 0 compared to the blueprint value of 1, and thus the search is *unsafe*.[2]

Insight: Performing search using Algorithm 1 is equivalent to performing search for *all* subgames. Even if conducting search only in a *single* subgame does not cause a shift in the follower's strategy, the combined effect of applying search to multiple subgames may. Again, one could remedy this by carefully selecting constraints. If we bound the follower post-search EVs for each of the 2 subgames to be ≥ 0, then we can guarantee that X would never be chosen. Note that this is not the only scheme which ensures safety, e.g., a lower bound of 1 and -1 for the left and right subgame is safe too.

4 Safe Search for SSE

The crux of our method is to modify naïve search such that the follower's best response remains the same even when search is applied. This, in turn, can be achieved by enforcing bounds on the follower's EV in any subgame strategies computed via search. Concretely, our search method comprises 3 steps, (i) preprocess the follower's best response

[2]This issue occurs in zero-sum games as well (Brown and Sandholm 2017a).

to the blueprint and its values, (ii) identify a set of non-trivial safety bounds on follower payoffs[3], and (iii) solving for the SSE in the subgame reached constrained to respect the bounds computed in (ii).

Preprocessing of Blueprint. Denote the leader's sequence form blueprint strategy as r_1^{bp}. We will assume that the game is small enough such that the follower's (pure, tiebreaks leader-favored) best response to the blueprint may be computed— denote it by r_2^{bp}. We call the set of information sets which, based on r_2^{bp} have non-zero probability of being reached the *trunk*, $T \subseteq \mathcal{I}_2 : r_2^{\text{bp}}(\mathbf{Seq}_2(T)) = 1$. Next, we traverse the follower's treeplex bottom up and compute the payoffs at each information set and sequence (accounting for chance factors $\mathcal{C}(z)$ for each leaf). We term these as best-response values (BRVs) under the blueprint. These are recursively computed for both $\sigma_2 \in \Sigma_2$ and $I_2 \in \mathcal{I}_2$ recursively, (i) $BRV(I_2) = \max_{\sigma_2 \in A(I_2)} BRV(\sigma_2)$, and (ii) $BRV(\sigma_2) = \sum_{I' \in \mathcal{I}_2 : \mathbf{Seq}_2(I') = \sigma_2} BRV(I') + \sum_{\sigma_1 \in \Sigma_1} r_1(\sigma_1) g_2(\sigma_1, \sigma_2)$, where $g_i(\sigma_i, \sigma_{-i})$ is the expected utility of player i over all nodes reached when executing the sequence pair (σ_i, σ_{-i}), $g_i(\sigma_i, \sigma_{-i}) = \sum_{h \in Z : \sigma_k = \mathbf{Seq}_k(h)} u_i(h) \cdot \mathcal{C}(h)$. This processing step involves just a single traversal of the game tree.

Generating Safety Bounds. Loosely speaking, we traverse the follower's treeplex top down while propagating follower payoffs bounds which guarantee that the follower's best response remains r_2^{bp}. This is recursively done until we reach an information set I belonging to some subgame j. The EV of I is then required to satisfy its associated bound for future steps of the algorithm. We illustrate the bounds generation process using the worked example in Figure 3. Values of information sets and sequences are in blue and annotated in order of traversal alongside their bounds, whose computation is as follows.

- The empty sequence $\emptyset$ requires a value greater than $-\infty$.
- For each information set (in this case, B) which follows $\emptyset$, we require (vacuously) for their values to be $\geq -\infty$.
- We want the sequence C to be chosen. Hence, the value of C has to be ≥ 3, which, with the lower bound of $-\infty$ gives a final bound of ≥ 3.
- Sum of values for parallel information sets D and H must be greater than C. Under the blueprint, their sum is 5. This gives a 'slack' of 2, split evenly between D and H, yielding bounds of $2 - 1 = 1$ and $3 - 1 = 2$ respectively.
- Sequence E requires a value no smaller than F, G, and the bound for by the D, which contains it. Other actions have follower payoffs smaller than 1. We set a *lower* bound of 1 for E and an *upper* bound of 1 for F and G.
- Sequence I should be chosen over J. Furthermore, the value of sequence I should be ≥ 2—this was the bound propagated into H. We choose the tighter of the J's blueprint value and the propagated bound of 2, yielding a bound of ≥ 2.5 for I and a bound of ≤ 2.5 for J.
- Sequences K and L should not be reached if the follower's best response to the blueprint is followed—we cannot make this portion too appealing. Hence, we apply upper bounds of 1.5 for sequences K and L.

[3]One could trivially achieve safety by sticking to the blueprint.

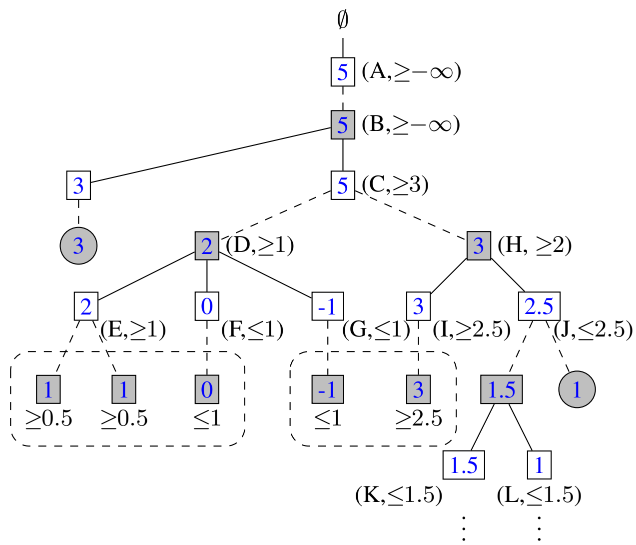

Figure 3: Example of bounds computation. Filled boxes represent information sets, circled nodes are terminal payoff entries, hollow boxes are sequences which may be followed by parallel information sets, which are in turn preceded by dashed lines. The dashed rectangle indicates subgames, of which we only show the head information sets of. BRVs of sequences and information sets are within the boxes and the (labels, computed bounds) are placed next to them.

A formal description for bounds generation is deferred to the Appendix. The procedure is recursive and identical to the worked example. It takes as input the game, blueprint, best response r_i^{bp}, and follower BRVs and returns upper and lower bounds $\mathcal{B}(I)$ for all head information sets of subgame j, $\mathcal{I}_{2,\text{head}}^j$. Since the blueprint strategy and its best response satisfies these bounds, feasibility is guaranteed. By construction, lower and upper bounds are obtained for information sets within and outside the trunk respectively. Note also that bounds computation requires only a single traversal of the follower's treeplex, which is smaller than the game tree.

The bounds generated are not unique. (i) Suppose we are splitting lower bounds at an information set I between child sequences (e.g., the way bounds for sequences E, F, G under information set D were computed). Let I have a lower bound of $\mathcal{B}(I)$ and the best and second best actions σ^* and σ' under the blueprint is v^* and v' respectively. Our implementation sets lower and upper bounds for σ^*, σ' to be $\max\{(v^* + v')/2, \mathcal{B}(I)\}$. However, any bound of the form $\max\{\alpha \cdot v^* + (1-\alpha) \cdot v', \mathcal{B}(I)\}, \alpha \in [0,1]$ achieves safety. (ii) Splitting lower bounds at sequences σ between parallel information sets under σ (e.g., when splitting the slack at C between D and H, or in Example 2.). Our implementation splits slack evenly though any non-negative split suffices. We explore these issues in our experiments.

MILP formulation for constrained SSE. Once safety bounds are generated, we can include them in a MILP similar to that of Bosansky and Cermak (2015). The solution of this MILP is the strategy of the leader, normalized such that $r_{\mathbf{Seq}_1(I_1)}$ for all $I_1 \in \mathcal{I}^j_{1,\text{head}}$ is equal to 1. Let Z^j be the set of terminal states which lie within subgame j, $Z^j = Z \cap H^j_{\text{sub}}$. Let $\mathcal{C}^j(z)$ be the new chance probability when all actions taken prior to the subgame are converted to be by chance, according to the blueprint. That is, $\mathcal{C}^j(z) = \mathcal{C}(z) \cdot r_1^{\text{bp}}\, \mathbf{Seq}_1(I_{1,\text{head}}(z)) \cdot r_2^{\text{bp}}\, \mathbf{Seq}_2(I_{2,\text{head}}(z))$. Similarly, we set $g_2^j(\sigma_1, \sigma_2) = \sum_{h \in Z^j : \sigma_k = \mathbf{Seq}_k(h)} u_2(h) \cdot \mathcal{C}^j(h)$. Let $\mathcal{M}(j)$ be the total probability mass entering subgame j in the original game when the blueprint strategy and best response, $\mathcal{M}(j) = \sum_{z \in Z^j} \mathcal{C}(z) \cdot r_1^{\text{bp}}(\mathbf{Seq}_1(z)) \cdot r_2^{\text{bp}}(\mathbf{Seq}_2(z))$.

$$\max_{p,r,v,s} \sum_{z \in Z^j} p(z) u_1(z) \mathcal{C}^j(z) \tag{1}$$

$$v_{\mathbf{Inf}_2(\sigma_2)} = s_{\sigma_2} + \sum_{I' \in \mathcal{I}_2 : \mathbf{Seq}_2(I') = \sigma_2} v_{I'} + \sum_{\sigma_1 \in \Sigma_1} r_1(\sigma_1)\, g_2^j(\sigma_1, \sigma_2) \qquad \forall \sigma_2 \in \Sigma_2^j \tag{2}$$

$$r_i(\sigma_i) = 1 \quad \forall i \in \{1,2\}, I_i \in \mathcal{I}^j_{i,\text{head}} : \mathbf{Seq}_i(I_i) = \sigma_i \tag{3}$$

$$r_i(\sigma_i) = \sum_{a \in A_i(I_i)} r_i(\sigma_i a) \qquad \forall i \in \{1,2\} \quad \forall I_i \in \mathcal{I}_i^j, \sigma_i = \mathbf{Seq}_i(I_i) \tag{4}$$

$$0 \le s_{\sigma_2} \le (1 - r_2(\sigma_2)) \cdot M \qquad \forall \sigma_2 \in \Sigma_2^j \tag{5}$$

$$0 \le p(z) \le r_2(\mathbf{Seq}_2(z)) \qquad \forall z \in Z^j \tag{6}$$

$$0 \le p(z) \le r_1(\mathbf{Seq}_1(z)) \qquad \forall z \in Z^j \tag{7}$$

$$\sum_{z \in Z^j} p(z) \mathcal{C}^j(z) = \mathcal{M}(j) \tag{8}$$

$$v_{I_2} \ge \mathcal{B}(I_2) \qquad \forall I_2 \in \mathcal{I}^j_{2,\text{head}} \cap T \tag{9}$$

$$v_{I_2} \le \mathcal{B}(I_2) \qquad \forall I_2 \in \mathcal{I}^j_{2,\text{head}} \cap \overline{T} \tag{10}$$

$$r_2(\sigma_2) \in \{0,1\} \qquad \forall \sigma_2 \in \Sigma_2^j \tag{11}$$

$$0 \le r_1(\sigma_1) \le 1 \qquad \forall \sigma_1 \in \Sigma_1^j \tag{12}$$

Conceptually, $p(z)$ is such that the probability of reaching z is $p(z)\mathcal{C}(z)$. r_1 and r_2 are the leader and follower sequence form strategies, v is the value of information set when r is adopted and s is the slack for each sequence.

Objective (1) is the expected payoff *in the full game* that the leader gets from subgame j, (3) and (4) are sequence form constraints, (5), (6), (7), (11) and (12) ensure the follower is best responding, and (8) ensures that the probability mass entering j is identical to the blueprint. Constraints (9) and (10) are bounds previously generated and ensure the follower does not deviate from r_2^{bp} after refinement. We discuss more details of the MILP in the appendix.

Safe Search as SSE solutions. One is not restricted to using a MILP to enforce these safety bounds. Here we show that the constrained SSE to be solved may be interpreted as the solution to another SSE problem. This implies that we can employ other SSE solvers, such as those involving strategy generation (Černỳ, Bošanskỳ, and Kiekintveld 2018). We briefly describe how transformation is performed on the j-th subgame, under the mild assumption that follower head information sets $\mathcal{I}^j_{2,\text{head}}$ are the initial states in H^j_{sub}. More detail is provided in the Appendix. Figure 4 shows an example construction based on the game in Figure 1.

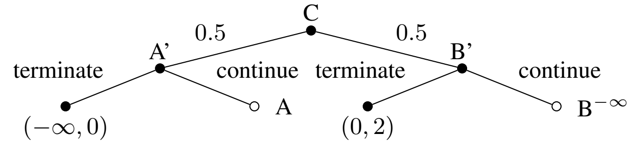

Figure 4: The transformed tree for solving the constrained SSE with the safety bounds of Figure 1. A' and B' are auxiliary states introduced for the follower. $B^{-\infty}$ is identical to B, except that leader payoffs are $-\infty$.

For every state $h \in \mathcal{I}^j_{2,\text{head}}$, we compute the probability ω_h of reaching h under the blueprint, assuming the follower plays to reach it. The transformed game begins with chance leading to a normalized distribution of ω over these states. Now, recall that we need to enforce bounds on follower payoffs for head information sets $I_2 \in \mathcal{I}^j_{2,\text{head}}$. To enforce a lower bound $BRV(I_2) \ge \mathcal{B}(I_2)$, we use a technique similar to *subgame resolving* (Burch, Johanson, and Bowling 2014). Before each state $h \in I_2$, insert an auxiliary state h' belonging to a new information set I'_2, where the follower may opt to terminate the game with a payoff of $(-\infty, \mathcal{B}(I_2)/(\omega_h |I_2|))$ or continue to h, whose subsequent states are unchanged.[4] If the leader's strategy has $BRV(I_2) < \mathcal{B}(I_2)$, the follower would do better by terminating the game, leaving the leader with $-\infty$ payoff.

Enforcing upper bounds $BRV(I_2) \le \mathcal{B}(I_2)$ may be done analogously. First, we reduce the payoffs to the leader for all leaves underneath I_2 to $-\infty$. Second, the follower has an additional action at I_2 to terminate the game with a payoff of $(0, \mathcal{B}(I_2)/(\omega_h |I_2|))$. If the the follower's response to the leader's strategy gives $BRV(I_2) > \mathcal{B}(I_2)$, then the follower would choose some action other than to terminate the game, which nets the leader $-\infty$. If the bounds are satisfied, then the leader gets a payoff of 0, which is expected given that an upper bound implies that I_2 is not part of the trunk.

5 Experiments

In this section we show experimental results for our search algorithm (based on the MILP in Section 4) in synthetic 2-stage games, Goofspiel and Leduc hold'em poker (modified to be general-sum). Experiments were conducted on a Intel i7-7700K @ 4.20GHz with 4 cores and 64GB of RAM. We use the commercial solver Gurobi (Gurobi Optimization 2021) to solve all instances of MILPs.

[4]The factor $\omega_h |I_2|$ arises since $\mathcal{B}$ was computed in treeplexes, which already takes into account chance.

We show that even if Stackelberg equilibrium computation for the entire game (using the MILP of Bosansky and Cermak (2015)) is warm started using the blueprint strategy r_1^{bp} and follower's best response r_2^{bp}, then in large games it is still intractable to compute a strategy. In fact, in some cases it is intractable to even generate the model, let alone solve it. In contrast, our safe search algorithm can be done at a far lower computational cost and with far less memory. Since our games are larger than what Gurobi is able to solve to completion in reasonable time, we instead constrain the time allowed to solve each (sub)game and report the incumbent solution. We consider only the time taken by Gurobi in solving the MILP, which dominates preprocessing and bounds generation, both of which only require a constant number of passes over the game tree. In all cases, we warm-start Gurobi with the blueprint strategy.

To properly evaluate the benefits of search, we perform search on *every* subgame and combine the resulting subgame strategies to obtain the implicit full-game strategy prescribed by Algorithm 1. The follower's best response to this strategy is computed and used to evaluate the leader's payoff. Note that *this is only done to measure how closely the algorithm approximates a SSE*—in practice, search is applied only to the subgame reached in actual play and is performed just once. Hence, the worst-case time for a *single* playthrough is no worse than the longest time required for search over a *single* subgame (and not the sum over all subgames).

We compare our method against the MILP proposed by Bosansky and Cermak (2015) rather the more recent incremental strategy generation method proposed by Černỳ, Bošanskỳ, and Kiekintveld (2018). The former is flexible and applies to all EFGs with perfect recall, while the latter involves the Stackelberg Extensive Form Correlated Equilibrium (SEFCE) as a subroutine for strategy generation. Computing an SEFCE is itself computationally difficult except in games with no chance, in which case finding an SEFCE can be written as a linear program.

Two-Stage Games. The two-stage game closely resembles a 2-step Markov game. In the first stage, both players play a general-sum matrix game G_{main} of size $n \times n$, after which, actions are made public. In the second stage, one out of M secondary games $\{G_{\text{sec}}^{j}\}$, each general-sum and of size $m \times m$ is chosen and played. Each player obtains payoffs equal to the sum of their payoffs for each stage. Given that the leader played action a_1, the probability of transitioning to game j is given by the mixture, $\mathbb{P}(G_{\text{sec}}^{(j)}|a_1) = \kappa \cdot X_{j,a_1} + (1-\kappa) \cdot q_j$, where X_{j,a_1} is a $M \times n$ transition matrix non-negative entries and columns summing to 1 and q_j lies on the M dimensional probability simplex. Here, κ governs the level of influence the leader's strategy has on the next stage. [5]

The columns of X are chosen by independently drawing weights uniformly from $[0,1]$ and re-normalizing, while q is uniform. We generate 10 games each for different settings of M, m, n and κ. A subgame was defined for each action pair played in the first stage, together with the secondary game transitioned into. The blueprint was chosen to be the SSE of the first stage *alone*, with actions chosen uniformly at random for the second stage. The SSE for the first stage was solved using the multiple LP method and runs in negligible time (<5 seconds). For full-game solving, we allowed Gurobi to run for a maximum of 1000s. For search, we allowed 100 seconds—in practice, this never exceeds more than 20 seconds for *any* subgame.

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

Table 1: Average leader payoffs for two-stage games.

We report the average quality of solutions in Table 1. The full-game solver reports the optimal solution if converges. This occurs in the smaller game settings where ($M = m \leq 10$). In these cases search performs near-optimally. In larger games ($M = m \geq 100$), full-game search fails to converge and barely outperforms the blueprint strategy. In fact, in the largest setting only 2 out of 10 cases resulted in *any* improvement from the blueprint, and even so, still performed worse than our method. Our method yields substantial improvements from the blueprint regardless of κ.

Goofspiel. Goofspiel (Ross 1971) is a game where 2 players simultaneously bid over a sequence of n prizes, valued at $0, \cdots, n-1$. Each player owns cards worth $1, \cdots, n$, which are used in closed bids for prizes auctioned over a span of n rounds. Bids are public after each round. Cards bid are discarded regardless of the auction outcome. The player with the higher bid wins the prize. In a tie, neither player wins and the prize is discarded. Hence, Goofspiel is not zero-sum, players can benefit by coordinating to avoid ties.

In our setting, the n prizes are ordered uniformly in an order unknown to players. Subgames are selected to be all states which have the same bids and prizes after first m rounds are resolved. As m grows, there are fewer but larger subgames. When $m = n$, the only subgame is the entire game. The blueprint was chosen to be the NE under a zero (constant)-sum version of Goofspiel, where players split the prize evenly in ties. The NE of a zero-sum game may be computed efficiently using the sequence form representation (Von Stengel 1996). Under the blueprint, the leader obtains

[5] One may be tempted to first solve the M Stackelberg games independently, and then apply backward induction, solving the first stage with payoffs adjusted for the second. This intuition is incorrect—the leader can issue non-credible threats in the second stage, inducing the follower to behave favorably in the first.

n	$(\lvert\Sigma\rvert, \lvert\mathcal{I}\rvert)$	m	Num. of sub-games	Max. time per sub-game (s)	Leader utility
4	$(2.1, 1.7) \cdot 10^4$	2	1728	5	3.02
		3	64	5	3.07
		$4^{\ddagger}$	1	5	4.06
				$1.0 \cdot 10^2$	4.15
				$5.5 \cdot 10^{2\dagger}$	4.23
5	$(2.7, 2.2) \cdot 10^6$	3	8000	$1.0 \cdot 10^2$	5.19
		4	125	$1.0 \cdot 10^2$	5.29
		$5^{\ddagger}$	1	$1.0 \cdot 10^3$	5.03
				$1.0 \cdot 10^4$	5.03
				$1.8 \cdot 10^{4\dagger}$	5.65

Table 2: Results for Goofspiel. †This is the earliest time that the incumbent solution achieves the given utility. ‡This is equivalent to full-game search.

a utility of 3.02 and 5.03 for $n = 4$ and $n = 5$ respectively.

Table 2 summarizes the solution quality and running time as we vary n, m. When $n = m$, Gurobi struggles to solve the program to optimality and we report the best incumbent solution found within a shorter time frame. As a sanity check, observe the leader's utility is never worse than the blueprint. When $n = 4$, the incumbent solution for solving the full game has improved significantly from the blueprint in fewer than 5 seconds. This indicates that the game is sufficiently small that performing search is not a good idea. However, when $n = 5$, solving the full game ($m = 5$) required 180 times longer compared to search ($m \in \{3, 4\}$) in order to obtain *any* improvement over the blueprint, while search only needed 100 seconds in order to improve upon the blueprint in a subgame. Furthermore, full-game solving required more than 50 GB of memory while search required less than 5 GB.

Leduc Hold'em. Leduc hold'em (Southey et al. 2012) is a simplified form of Texas hold'em. Players are dealt a single card in the beginning. In our variant there are n cards with 2 suits, 2 betting rounds, an initial bet of 1 per player, and a maximum of 5 bets per round. The bet sizes for the first and second round are 2 and 4. In the second round, a public card is revealed. If a player's card matches the number of the public card, then he/she wins in a showdown, else the higher card wins (a tie is also possible).

Our variant of Leduc includes *rake*, which is a commission fee to the house. We assume for simplicity a fixed rake $\rho = 0.1$. This means that the winner receives a payoff of $(1-\rho)x$ instead of x. The loser still receives a payoff of $-x$. When $\rho > 0$, the game is not zero-sum. Player 1 assumes the role of leader. Subgames are defined to be all states with the same public information from the second round onward. The blueprint strategy was obtained using the unraked ($\rho = 0$, zero-sum) variant and is solved efficiently using a linear program. We limited the full-game method to a maximum of 5000 seconds and 200 seconds *per subgame* for our method. We reiterate that since we perform search only on subgames encountered in actual play, 200 seconds is an *upper bound*

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

Table 3: Leader payoffs for Leduc hold'em with n cards.

α	0.1	0.25	$0.5^{\dagger\dagger}$	0.75	0.9
Goofspiel	5.32	5.34	5.29	5.31	5.30
Leduc	-0.184	-0.185	-0.186	-0.188	-0.189
β	$1^{\dagger\dagger}$	2	4	8	16
Goofspiel	5.29	5.35	5.55	5.56	5.50
Leduc	-0.186	-0.182	-0.178	-0.212	-0.212

Table 4: Leader payoffs for varying α and β. We consider Goofspiel with $n = 5, m = 4$ and Leduc Hold'em with $n = 4$. Time constraints are the same as previous experiments. ††These are the default values for α and β.

on the time taken for a single playthrough when employing search (some SSE are easier than others to solve).

The results are summarized in Table 3. For large games, the full-game method struggles with improving on the blueprint. In fact, when $n = 8$ the number of terminal states is so large that the Gurobi model could not be created even after 3 hours. Even when $n = 6$, model construction took an hour—it had near $7 \cdot 10^5$ constraints and $4 \cdot 10^5$ variables, of which $2.3 \cdot 10^4$ are binary. Even when the model was successfully built, no progress beyond the blueprint was made.

Varying Bound Generation Parameters. We now explore how varying α affects solution quality. Furthermore, we experiment with multiplying the slack (see information sets D and H in Section 4) by a constant $\beta \geq 1$. This results in weaker but potentially unsafe bounds. Results on Goofspiel and Leduc are summarized in Figure 4. We observe that lower values of α yield slightly better performance in Leduc, but did not see any clear trend for Goofspiel. As β increases, we observe significant improvements initially. However, when β is too large, performance suffers and even becomes unsafe in the case of Leduc. These results suggest that search may be more effective with principled selections of α and β, which we leave for future work.

6 Conclusion

In this paper, we have extended safe search to the realm of SSE in EFGs. We show that safety may be achieved by adding a few straightforward bounds on the value of follower information sets. We showed it is possible to cast the bounded search problem as another SSE, which makes our approach complementary to other offline methods. Our experimental results on Leduc hold'em demonstrate the ability of our method to scale to large games beyond those which MILPs may solve. Future work includes relaxing constraints on subgames and extension to other equilibrium concepts.

References

Bosansky, B.; and Cermak, J. 2015. Sequence-form algorithm for computing stackelberg equilibria in extensive-form games. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Brown, N.; and Sandholm, T. 2017a. Safe and nested subgame solving for imperfect-information games. In *Advances in neural information processing systems*, 689–699.

Brown, N.; and Sandholm, T. 2017b. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* eaao1733.

Brown, N.; and Sandholm, T. 2019. Superhuman AI for multiplayer poker. *Science* 365(6456): 885–890.

Burch, N.; Johanson, M.; and Bowling, M. 2014. Solving imperfect information games using decomposition. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Campbell, M.; Hoane Jr, A. J.; and Hsu, F.-h. 2002. Deep blue. *Artificial intelligence* 134(1-2): 57–83.

Černỳ, J.; Bošanskỳ, B.; and Kiekintveld, C. 2018. Incremental Strategy Generation for Stackelberg Equilibria in Extensive-Form Games. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, 151–168. ACM.

Conitzer, V.; and Sandholm, T. 2006. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, 82–90. ACM.

Fang, F.; Nguyen, T. H.; Pickles, R.; Lam, W. Y.; Clements, G. R.; An, B.; Singh, A.; Schwedock, B. C.; Tambe, M.; and Lemieux, A. 2017. PAWS-A Deployed Game-Theoretic Application to Combat Poaching. *AI Magazine* 38(1): 23.

Gurobi Optimization, L. 2021. Gurobi Optimizer Reference Manual. URL http://www.gurobi.com.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2): 100–107.

Hoda, S.; Gilpin, A.; Pena, J.; and Sandholm, T. 2010. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research* 35(2): 494–512.

Jain, M.; Kiekintveld, C.; and Tambe, M. 2011. Quality-bounded solutions for finite bayesian stackelberg games: Scaling up. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 997–1004. International Foundation for Autonomous Agents and Multiagent Systems.

Letchford, J.; and Conitzer, V. 2010. Computing optimal strategies to commit to in extensive-form games. In *Proceedings of the 11th ACM conference on Electronic commerce*, 83–92. ACM.

Lin, S. 1965. Computer solutions of the traveling salesman problem. *Bell System Technical Journal* 44(10): 2245–2269.

Moravčík, M.; Schmid, M.; Burch, N.; Lisỳ, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356(6337): 508–513.

Moravcik, M.; Schmid, M.; Ha, K.; Hladik, M.; and Gaukrodger, S. J. 2016. Refining subgames in large imperfect information games. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Pita, J.; Jain, M.; Marecki, J.; Ordóñez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, 125–132. International Foundation for Autonomous Agents and Multiagent Systems.

Ross, S. M. 1971. Goofspiel — the game of pure strategy. *Journal of Applied Probability* 8(3): 621–625. doi:10.2307/3212187.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529(7587): 484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362(6419): 1140–1144.

Southey, F.; Bowling, M. P.; Larson, B.; Piccione, C.; Burch, N.; Billings, D.; and Rayner, C. 2012. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411* .

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM* 38(3): 58–68.

Von Stengel, B. 1996. Efficient computation of behavior strategies. *Games and Economic Behavior* 14(2): 220–246.