Subgame solving without common knowledge

Anonymous Author(s)
Affiliation
Address
email

Abstract

In imperfect-information games, subgame solving is significantly more challenging than in perfect-information games, but in the last few years, such techniques have been developed. They were the key ingredient to the milestone of superhuman play in no-limit Texas hold'em poker. Current subgame-solving techniques analyze the entire *common-knowledge closure* of the player's current information set, that is, the smallest set of nodes within which it is common knowledge that the current node lies. While this is acceptable in games like poker where the common-knowledge closure is relatively small, many practical games have more complex information structure, which renders the common-knowledge closure impractically large to enumerate or even reasonably approximate. We introduce an approach that overcomes this obstacle, by instead working with only low-order knowledge. Our approach allows an agent, upon arriving at an infoset, to basically prune any node that is no longer reachable, thereby massively reducing the game tree size relative to the common-knowledge subgame. We prove that, as is, our approach can increase exploitability compared to the blueprint strategy. However, we develop three avenues by which safety can be guaranteed. First, safety is guaranteed if the results of subgame solves are incorporated back into the blueprint. Second, we provide a method where safety is achieved by limiting the infosets at which subgame solving is performed. Third, we prove that our approach, when applied at every infoset reached during play, achieves a weaker notion of equilibrium, which we coin *affine equilibrium*, and which may be of independent interest. We show that affine equilibria cannot be exploited by any Nash strategy of the opponent, so an opponent who wishes to exploit must open herself to counter-exploitation. Even without the safety-guaranteeing additions, experiments on medium-sized games show that our approach always reduced exploitability even when applied at every infoset, and a depth-limited version of it led to—to our knowledge—the first strong AI for the massive challenge problem *dark chess*.

1 Introduction

Subgame solving is the standard technique for playing perfect-information games that has been used by strong agents in a wide variety of games, including chess [7, 23] and go [21]. Methods for subgame solving in perfect-information games exploit the fact that a solution to a subgame can be computed independently of the rest of the game. However, this condition fails in the imperfect-information setting, where the optimal strategy in a subgame can depend on strategies outside that subgame.

Recently, subgame solving techniques have been extended to imperfect-information games [9, 14]. Some of those techniques are provably *safe* in the sense that, under reasonable conditions, incorporating them into an agent cannot make the agent more exploitable [6, 20, 2, 19, 5, 24, 1, 16]. These techniques formed the core ingredient toward recent superhuman breakthroughs in AIs for no-limit Texas hold'em poker [3, 4]. However, all of the prior techniques have a shared weakness that

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

limits their applicability: as a first step, they enumerate the entire *common-knowledge closure* of the player's current infoset, which is the smallest set of states within which it is common knowledge that the current node lies. In two-player community-card poker (in which each player is dealt private hole cards, and all actions are public, e.g., Texas hold'em), for example, the common-knowledge closure contains one node for each assignment of hole cards to both players. This set has a manageable size in such poker games, but in other games, it is unmanageably large.

We introduce a completely different technique to avoid having to enumerate the entire common-knowledge closure. We enumerate only the set of nodes corresponding to kth-order knowledge for finite k—in the present work, we focus mostly on the case $k = 1$, for it already gives us interesting results. This allows an agent to only conduct subgame solving on still-reachable states, which in general is a much smaller set than the whole common-knowledge subgame.

We prove that, as is, the resulting algorithm, 1-KLSS, does not guarantee safety, but we develop three avenues by which safety can be guaranteed. First, safety is guaranteed if the results of subgame solves are incorporated back into the blueprint strategy. Second, we provide a method by which safety is achieved by limiting the infosets at which subgame solving is performed. Third, we prove that our approach, when applied at every infoset reached during play, achieves a weaker notion of equilibrium, which we coin *affine equilibrium* and which may be of independent interest. We show that affine equilibria cannot be exploited by any Nash strategy of the opponent: an opponent who wishes to exploit an affine equilibrium must open herself to counter-exploitation. Even without these three safety-guaranteeing additions, experiments on medium-sized games show that 1-KLSS always reduced exploitability even when applied at every infoset.

We use depth-limited 1-KLSS to create, to our knowledge, the first agent capable of playing *dark chess*, a massive benchmark game, at a high level. We test it against opponents of various levels, including a baseline agent, an amateur-level human, and the world's highest-rated player. Our agent defeated the former two handily, and, despite losing to the top human, exhibited strong performance in the opening and midgame, often gaining a significant advantage before losing it in the endgame.

2 Notation and definitions

An *extensive-form perfect-recall zero-sum game with explicit observations* (hereafter *game*) Γ between two players $\oplus$ and $\ominus$ consists of: **1)** a tree H of *nodes* with labeled edges, rooted at a *root node* $\emptyset \in H$. The set of leaves, or *terminal nodes*, of H will be denoted Z. The labels on the edges are called *actions*. The child node reached by playing action a at node h will be denoted ha. **2)** a *utility function* $u : Z \to \mathbb{R}$. **3)** a map $P : (H \setminus Z) \to \{\text{NATURE}, \oplus, \ominus\}$ denoting which player's turn it is. **4)** for each player $i \in \{\oplus, \ominus\}$, and each internal node $h \in H \setminus Z$, an *observation* $\mathcal{O}_i(h)$ that player i learns upon reaching h. The observation must uniquely determine whether player i has the move; i.e., if $\mathcal{O}_i(h) = \mathcal{O}_i(h')$, then either $P(h), P(h') = i$, or $P(h), P(h') \neq i$. **5)** for each node h with $P(h) = \text{NATURE}$, a distribution $p(\cdot|h)$ over the actions at h.

A player i's *observation sequence* (hereafter *sequence*) mid-playthrough is the sequence of observations made and actions played by i so far. The set of sequences of player i will be denoted Σ_i. The observation sequence at node h (immediately after i observes $\mathcal{O}_i(h)$) will be denoted $s_i(h)$.

We say that two states $h = \emptyset a_1 \dots a_t$ and $h' = \emptyset b_1 \dots b_t$ are *indistinguishable to player* i, denoted $h \sim_i h'$, if $s_i(h) = s_i(h')$. An equivalence class of nodes $h \in H$ under $\sim_i$ is an *information set*, or *infoset* for player i. If two nodes at which player i moves belong to the same infoset I, the same set of actions must be available at h and h'. If a is a legal action at I, we will use Ia to denote the sequence reached by playing action a at I.

If u, u' are nodes or sequences, $u \preceq u'$ means u is an ancestor or prefix (respectively) of u' (or $u' = u$). If S is a set of nodes, $h \succeq S$ means $h \succeq h'$ for some $h' \in S$, and $\overline{S} = \{z : z \succeq S\}$.

A *sequence-form mixed strategy* (hereafter *strategy*) of player i is a vector $x \in \mathbb{R}^{\Sigma_i}$, in which $x(s)$ denotes the probability that player i plays all the actions in the sequence s. If h is a node or infoset, then we will use the overloaded notation $x(h) := x(s_i(h))$. The set of valid strategies for each player forms a convex polytope [15], which we will denote X and Y for $\oplus$ and $\ominus$ respectively. A strategy profile $(x, y) \in X \times Y$ is a pair of strategies. The payoff for $\oplus$ in a strategy profile (x, y) will be denoted $u(x, y) := \sum_{z \in Z} u(z)p(z)x(z)y(z)$, where $p(z)$ is the probability that nature plays all the strategies on the path from $\emptyset$ to z. (The payoff for $\ominus$ is $-u(x, y)$ since the game is zero-sum.) The

payoff matrix is the matrix $A \in \mathbb{R}^{\Sigma_\oplus \times \Sigma_\ominus}$ whose bilinear form is the utility function, that is, for which $\langle x, Ay \rangle = u(x, y)$. Most common game-solving algorithms, such as linear programming [15], counterfactual regret minimization and its modern variants [29, 8], and first-order methods such as EGT [13, 17] work directly with the payoff matrix representation of the game.

The *counterfactual best-response value* (hereafter *best-response value*) $u^*(x|Ia)$ to a $\oplus$-strategy $x \in X$ upon playing action a at I is the normalized best value for $\ominus$ against x after playing a at I: $u^*(x|Ia) = \frac{1}{\sum_{h \in I} p(h)x(h)} \min_{y \in Y : y(Ia)=1} \sum_{z : s_\ominus(z) \succeq Ia} u(z)p(z)x(z)y(z)$. The best-response value at an infoset I is defined as $u^*(x|I) = \max_a u^*(x|Ia)$. The *best-response value* $u^*(x)$ (without specifying an infoset) is the best-response value at the root, i.e., $\min_{y \in Y} u(x, y)$. Analogous definitions hold for $\ominus$-strategy y and $\oplus$-infoset I. A player is playing an *ε-best response* in a strategy profile (x, y) if $u(x, y)$ is within ε of the best-response value of her opponent's strategy. We say that (x, y) is an *ε-Nash equilibrium* (ε-NE) if both players are playing ε-best responses. *Best responses* and *Nash equilibria* are, respectively, 0-best responses and 0-Nash equilibria. An *NE strategy* is one that is part of an NE. The set of NE strategies is also a convex polytope [15].

We say that two nodes h and h' are *transpositions* if an observer who begins observing the game at h or h' and sees both players' actions and observations at every timestep cannot distinguish between the two nodes. Formally, h, h' are transpositions if, for all action sequences $a_1 \dots a_t$: **1)** $ha_1 \dots a_t$ is valid (i.e., for all j, a_j is a legal move in $ha_1 \dots a_{j-1}$) if and only if $h'a_1 \dots a_t$ is valid, and in this case, we have $\mathcal{O}_i(ha_1 \dots a_j) = \mathcal{O}_i(h'a_1 \dots a_j)$ for all players i and times $0 \leq j \leq t$, and **2)** $ha_1 \dots a_t$ is terminal if and only if $h'a_1 \dots a_t$ is terminal, and in this case, we have $u(ha_1 \dots a_t) = u(h'a_1 \dots a_t)$. For example, ignoring draw rules, two chess positions are transpositions if they have equal piece locations, castling rights, and *en passant* rights.

3 Common-knowledge subgame solving

In this section we discuss prior work on subgame solving. First, $\oplus$ computes a blueprint strategy x for the full game. During a playthrough, $\oplus$ reaches an infoset I, and would like to perform subgame solving to refine her strategy for the remainder of the game. All prior subgame solving methods that we are aware of require, as a first step, constructing [6, 20, 2, 19, 5, 24, 1, 16], or at least approximating via samples [25], the *common-knowledge closure* of I.

Definition 1. The *infoset hypergraph* G of a game Γ is the hypergraph whose vertices are the nodes of Γ, and whose hyperedges are information sets.

Definition 2. Let S be a set of nodes in Γ. The *order-k knowledge set* S^k is the set of nodes that are at most distance $k - 1$ away from S in G. The *common-knowledge closure* S^∞ is the connected component of G containing S.

Intuitively, if we know that the true node is in S, then we know that the opponent knows that the true node is in S^2, we know that the opponent knows that we know that the true node is in S^3, etc., and it is common knowledge that the true node is in S^∞. After constructing I^∞ (where I, as above, is the infoset $\oplus$ has reached), standard techniques then construct the subgame $\overline{I^\infty}$ (or an abstraction of it), and solve it to obtain the refined strategy. In this section we describe three variants: *resolving* [6], *maxmargin* [20], and *reach subgame solving* [2].

Let H_{top} be the set of root nodes of I^∞, that is, the set of nodes $h \in I^\infty$ for which the parent of h is not in I^∞. In *subgame resolving*, the following gadget game is constructed. First, nature chooses a node $h \in H_{\text{top}}$ with probability proportional to $p(h)x(h)$. Then, $\ominus$ observes her infoset $I_\ominus(h)$, and is given the choice to either *exit* or *play*. If she exits, the game ends at a terminal node z with $u(z) = u^*(x|I_\ominus(h))$. This payoff is called the *alternate payoff* at $I_\ominus(h)$. Otherwise, the game continues from node h. In *maxmargin* solving, the objective is changed to instead find a strategy x' that maximizes the minimum *margin* $M(I) := u^*(x'|I) - u^*(x|I)$ associated with any $\ominus$-infoset I intersecting H_{top}. (Resolving only ensures that all margins are positive). This can be accomplished by modifying the gadget game. In *reach subgame solving*, the alternative payoffs $u^*(x|I)$ are decreased by the *gift* at I, which is a lower bound on the magnitude of error that $\ominus$ has made by playing to reach I in the first place. Reach subgame solving can be applied on top of either resolving or maxmargin.

The full game Γ is then replaced by the gadget game, and the gadget game is resolved to produce a strategy x' that $\oplus$ will use to play to play after I. To use nested subgame solving, the process repeats when another new infoset is reached.

4 Knowledge-limited subgame solving

In this section we introduce the main contribution of our paper, *knowledge-limited subgame solving*. The core idea is to reduce the computational requirements of safe subgame solving methods by discarding nodes that are "far away" (in the infoset hypergraph G) from the current infoset.

Fix an odd positive integer k. In *order-k knowledge-limited subgame solving* (*k-KLSS*), we *fix* $\oplus$'s strategy outside $\overline{I^k}$, and then perform subgame solving as usual. Pseudocode for all algorithms can be found in the appendix. This carries many advantages: **1)** Since $\oplus$'s strategy is fixed outside $\overline{I^k}$, $\ominus$'s best response outside $\overline{I^{k+1}}$ is also fixed. Thus, all nodes outside $\overline{I^{k+1}}$ can be pruned and discarded. **2)** At nodes $h \in \overline{I^{k+1}} \setminus \overline{I^k}$, $\oplus$'s strategy is again fixed. Thus, the payoff at these nodes is only a function of $\ominus$'s strategy in the subgame and the blueprint strategy. These payoffs can be computed from the blueprint and added to the row of the payoff matrix corresponding to $\oplus$'s empty sequence. These nodes can then also be discarded, leaving only $\overline{I^k}$. **3)** Transpositions can be accounted for if $k = 1$ and we allow a slight amount of incorrectness. Suppose that $h, h' \in I$ are transpositions. Then $\oplus$ cannot distinguish h from h' ever again. Further, $\ominus$'s information structure after h in $\overline{I^k}$ is identical to her information structure in h' in $\overline{I^k}$. Thus, in the payoff matrix of the subgame, h and h' induce two disjoint sections of the payoff matrix A_h and $A_{h'}$ that are identical except for the top row (thanks to Item 2 above). We can thus remove one (say, at random) without losing too much. If one section of the matrix contains entries that are all not larger than the corresponding entries of the other part, then we can remove the latter part without any loss since it is weakly dominated.

The transposition merging may cause incorrect behavior (over-optimism) in games such as poker, but we believe that its effect in a game like dark chess, where information is transient at best and the evaluation of a position depends more on the actual position than on the players' information, is minor. Other abstraction techniques can also be used to reduce the size of the subgame, if necessary. We will denote the resulting gadget game $\Gamma[I^k]$.

In games like dark chess, even individual infosets can have size 10^7, which means even I^2 can have size 10^{14} or larger. This is wholly unmanageable in real time. Further, very long shortest paths can exist in the infoset hypergraph G. As such, it may be difficult to even determine whether a given node is in I^∞, much less expand all its nodes, even approximately. Thus, being able to reduce to I^k for finite k is a large step in making subgame solving techniques practical.

The benefit of KLSS can be seen concretely in the following parameterized family of games which we coin *N-matching pennies*. We will use it as a running example in the rest of the paper. Nature first chooses an integer $n \in \{1, \dots, N\}$ uniformly at random. $\oplus$ observes $\lfloor n/2 \rfloor$ and $\ominus$ observes $\lfloor (n+1)/2 \rfloor$. Then, $\oplus$ and $\ominus$ simultaneously choose heads or tails. If they both choose heads, $\oplus$ scores n. If they both choose tails, $\oplus$ scores $N - n$. If they choose opposite sides, $\oplus$ scores 0. For any infoset I just after nature makes her move, there is no common knowledge whatsoever, so $\overline{I^\infty}$ is the whole game except for the root nature node. However, I^k consists of only $\Theta(k)$ nodes.

On the other hand, in community-card poker, $I^3 = I^\infty$ for every I (and I^2 is already very close to I^∞, excluding only "blockers"). Further, I^∞ itself is quite small: indeed, in heads-up Texas Hold'Em, I^∞ always has size at most $\binom{52}{2} \cdot \binom{50}{2} \approx 1.6 \times 10^6$ and even fewer after public cards have been dealt. Furthermore, lossless abstraction can be used to make the game even smaller [11, 26]. This is manageable in real time, and is the key that has enabled recent breakthroughs in AIs for no-limit Texas hold'em [19, 3, 4]. In such settings, we do not expect our techniques to give much improvement over the current state of the art.

The rest of this section addresses the *safety* of KLSS. The techniques in Section 3 are *safe* in the sense that applying them at every infoset reached during play in a nested fashion cannot increase exploitability compared to the blueprint strategy [6, 20, 2]. KLSS is not safe in that sense:

Proposition 3. *There exists a game and blueprint for which applying 1-KLSS at every infoset reached during play increases exploitability by a factor linear in the size of the game.*

Proof. Consider the following game. Nature chooses an integer $n \in \{1, \dots, N\}$, and tells $\oplus$ but not $\ominus$. Then the two players play matching pennies, with $\ominus$ winning if the pennies match. Consider the blueprint strategy for $\oplus$ that plays heads with probability exactly $1/2 + 2/N$, regardless of n. This strategy is a $\Theta(1/N)$-equilibrium strategy for $\oplus$. However, if maxmargin 1-KLSS is applied independently at every infoset reached, $\oplus$ will deviate to playing tails for all n, because she is treating

her strategy at all $m \neq n$ as fixed, and the resultant strategy is more balanced. This strategy is exploitable by $\ominus$ always playing tails. $\square$

Despite the above negative example, we now give multiple methods by which we can obtain safety guarantees when using KLSS.

Safety by updating the blueprint. Our first method of obtaining safety is to immediately and permanently update the blueprint strategy after every subgame solution is computed. Proofs of the results in this section can be found in the appendix.

Theorem 4. *Suppose that whenever k-KLSS is performed at infoset I (e.g., it can be performed at every infoset reached during play in a nested manner), and that subgame strategy is immediately and permanently incorporated into the blueprint, thereby overriding the blueprint strategy in $\overline{I^k}$. The resulting strategy has exploitability at most that of the blueprint.*

One way to satisfy this safety condition is to store the computed solutions to all subgames that the agent has ever solved. In games where only a reasonably small number of paths get played in practice (this can depend on the strength and style of the players), this is feasible. In other games this might be prohibitively storage intensive.

Another way to guarantee safety is to satisfy this theorem only if I^{k+1} can be reached again. In games where a large number of paths are played, this is typically very unlikely in late-game situations. For example, in dark chess, it is incredibly unlikely that a situation 20 moves deep would be reached ever again. In such use cases, one can just do the blueprint updates in the early parts of the game.

In the rest of this section we prove forms of safety guarantees for 1-KLSS that do not require the blueprint to be updated at all.

Safety by allocating deviations from the blueprint. We now show that another way to achieve safety of 1-KLSS is to carefully allocate how much it is allowed to deviate from the blueprint. Let G' be the graph whose nodes are infosets for $\oplus$, and in which two infosets I and I' share an edge if they contain nodes that are in the same $\ominus$-infoset. In other words, G' is the infoset hypergraph G, but with every $\oplus$-infoset collapsed into a single node.

Theorem 5. *Let x be an ε-NE blueprint strategy for $\oplus$. Let $\mathcal{I}$ be an independent set in G' that is closed under ancestor (that is, if $I \succeq I'$ and $I \in \mathcal{I}$, then $I' \in \mathcal{I}$). Suppose that 1-KLSS is performed only at infosets in $\mathcal{I}$, to create a strategy x'. Then x' is also an ε-NE strategy.*

There are at least two reasonable methods to apply Theorem 5 in practice. The first is to generate $\mathcal{I}$ *incrementally*: maintain a set of infosets $\mathcal{I}$ at which subgame solving has been performed in the past. Upon reaching an infoset I, if I can be added to $\mathcal{I}$ without breaking the necessary independence property, do so and run 1-KLSS at I. Otherwise, play according to the blueprint. For example, consider using this method on N-maching pennies. Given a blueprint x, suppose that we will use the blueprint to play a fixed number of games T against an opponent, and $T \ll \sqrt{N}$. Then the birthday paradox means that it is highly unlikely (probability approaching 0 for increasing N) for nature to draw adjacent infosets in any two of the N playthroughs. As a result, in this setting, we are usually able to perform subgame solving on every playthrough.

The second method is to pick some probability distribution π over independent sets of G', which induces a map $p : V(G') \to \mathbb{R}$ where $p(I) = \Pr_{\mathcal{I}\sim\pi}[I \in \mathcal{I}]$. Then, upon reaching infoset I, with probability $1 - p(I)$, play the blueprint until the end of the game; otherwise, run 1-KLSS at I (possibly resulting in more nested subgame solves) and play that strategy instead. It is always safe to set $p(I) \leq 1/\chi(G'[I^\infty])$ where χ denotes the chromatic number and $G'[I^\infty]$ is the subgraph of G' induced by the infosets in the common-knowledge closure I^∞. For example, if the game is perfect information, then $G'[I^\infty]$ is the trivial graph with only one node I, so, as expected, it is safe to set $p(I) = 1$, that is, perform subgame solving everywhere.

Affine equilibrium, which guarantees safety against all equilibrium strategies. We now introduce the notion of *affine equilibrium*. We will show that such equilibrium strategies are safe against all NE strategies, which implies that they are only exploitable by playing non-NE strategies, that is, by opening oneself up to counter-exploitation. We then show that 1-KLSS finds such equilibria.

Definition 6. A vector x is an *affine combination* of vectors $x_1, \ldots, x_k$ if $x = \sum_{i=1}^k \alpha_i x_i$ with $\sum_i \alpha_i = 1$, where the coefficients α_i can have arbitrary magnitude and sign.

Definition 7. An *affine equilibrium strategy* is an affine combination of NE strategies.

In particular, if the NE is unique, then so is the affine equilibrium. Before stating our safety guarantees, we first state another fact about affine equilibria that illuminates their utility.

Proposition 8. *Every affine equilibrium is a best response to every NE strategy of the opponent.*

In other words, every affine equilibrium is an NE of the restricted game Γ' in which $\ominus$ can only play her NE strategies in Γ. That is, affine equilibria are not exploitable by NE strategies of the opponent, not even by safe exploitation techniques [10]. So, the only way for the opponent to exploit an affine equilibrium is to open herself up to counter-exploitation. Affine equilibria may be of independent interest as a reasonable relaxation of NE in settings where finding an exact or approximate NE strategy may be too much to ask for.

Theorem 9. *Let x be a blueprint strategy for $\oplus$, and suppose that x happens to be an NE strategy. Suppose that we run 1-KLSS using the blueprint x, at every infoset reached during play. Then the resulting strategy is an affine equilibrium strategy.*

The theorem could perhaps be generalized to approximate equilibria, but the loss of a large factor (linear in the size of the game, in the worst case) in the approximation would be unavoidable: the counterexample in the proof of Proposition 3 has a $\Theta(1/N)$-NE becoming a $\Theta(1)$-NE, in a game where the Nash equilibria are already affine-closed (that is, all affine combinations of Nash equilibria are Nash equilibria). Furthermore, it is nontrivial to even define ε-affine equilibrium.

Theorem 9 and Proposition 3 together suggest that 1-KLSS may make mistakes when x suffers from *systematic* errors (e.g., playing a certain action a too frequently *overall* rather than in a particular infoset). 1-KLSS may overcorrect for such errors, as the counterexample clearly shows. Intuitively, if the blueprint plays action a too often (e.g., folds in poker), 1-KLSS may try to correct for that game-wide error fully in each infoset, thereby causing the strategy to overall be very far from equilibrium (e.g., folding way too infrequently in poker). However, we will demonstrate that this overcorrection never happens in our experiments, even if the blueprint contains very systematic errors.

Strangely, the proofs of both Theorem 9 and Theorem 5 do not work for k-KLSS when $k > 1$, because it is no longer the case that the strategies computed by subgame solving are necessarily played—in particular, for $k > 1$, k-KLSS on an infoset I computes strategies for infosets I' that are no longer reachable, and such strategies may never be played. For $k = \infty$—that is, for the case of common knowledge—it is well known that the theorems hold via different proofs [6, 20, 2]. We leave the investigation of the case $1 < k < \infty$ for future research.

5 Dark chess: An agent from only a value function rather than a blueprint

In this section, we detail our dark chess agent that utilizes 1-KLSS. Although we wrote our agent in a game-specific fashion, many techniques in this section also apply to other games.

Definition 10. A *trunk* of a game Γ is a modified version of Γ in which some internal nodes h of Γ have been replaced by terminal nodes and given utilities. We will call such nodes *internal leaves*. When working with a trunk, internal leaves h can be *expanded* by adding all of their children into the tree, giving these children utilities, and removing the utility assigned to h.

In dark chess, constructing a blueprint is already a difficult problem due to the sheer size of the game, and expanding the whole game tree is clearly impractical. Instead, we resort to a *depth-limited* version of 1-KLSS. In depth-limited subgame solving, only a trunk of the game tree is expanded explicitly, and approximations are made to the leaves of the trunk.

Conventionally in depth-limited subgame solving of imperfect-information games, at each trunk leaf, both players are allowed to choose among *continuation strategies* for the remainder of the game [5, 1, 16, 25]. In the absence of a mechanism for creating a reasonable blueprint, much less multiple blueprints to be used as continuation strategies, we resort to only using an approximate value function $\tilde{u} : H \to \mathbb{R}$. We will not formally define what a good value function is, except that it should roughly approximate "the value" of a node $h \in H$, to the extent that such a quantity exists (for a more rigorous treatment of value functions in subgame solving, see Kovařík et al., 2021 [16]). In this setting, this is not too bothersome: the dominant term in any reasonable node-value function in dark chess will be material count, which is common knowledge anyway.

For our dark chess agent, we use the value function given by running the chess engine *Stockfish 13* [23], which is currently the strongest available chess engine, at depth 1, and then clamping the

reward to a range $[-\tau, \tau]$ (where τ is a tuneable hyperparameter; we set $\tau = 6$ pawns) via the mapping $x \mapsto \tanh(x/\tau)$. Using Stockfish's evaluation function saves us the trouble and resources required to learn chess from scratch, and clamping it to a finite range ensures that our agent understands that, after a certain point, a higher evaluation does not indicate a substantially higher probability of victory. Subgame solving in imperfect-information games with only approximate leaf values (and no continuation strategies) has not been explored to our knowledge (since it is not theoretically sound), but it seems reasonable to assume that it would work well with sufficient depth, since increasing depth effectively amounts to adding more and more continuation strategies.

To perform nested subgame solving, every time it is our turn, we perform 1-KLSS at our current information set. The generated subgame then replaces the original game, and the process repeats. This approach has the notable problem of information loss over time: since all the solves are depth-limited, eventually, we will reach a point where we fall off the end of the initially-created game tree. At this point, those nodes will disappear from consideration. From a game-theoretic perspective, this equates to always assuming that the opponent knew the exact state of the game d timesteps ago, where d is the search depth. As a remedy, one may consider sampling some number of infosets $I' \succeq I^2 \setminus I$ to continue expanding. We do not investigate this possibility here, as we believe that it would not yield a significant performance benefit in dark chess (and may even hurt in practice: since no blueprint is available at I', a new blueprint would have to be computed. This effectively amounts to 3-KLSS, which may lack theoretical guarantees compared to 1-KLSS).

Adapting techniques from perfect-information game solving. *Iterative deepening* is a natural approach to incrementally generate the game tree when solving a game in the perfect-information setting [22], and is used by most strong chess engines. We suggest a natural extension of iterative deepening to imperfect-information games. At all times, maintain a trunk that initially contains only the root node. Solve the trunk game exactly (e.g., with an LP solver). If time permits, expand all internal leaves that are in the support of *either* player's strategy, and repeat. This technique is sound in the sense that if it does not expand any node, then an equilibrium of the full game has been found. It carries some resemblance to recent techniques for generating *certificates* [27, 28], but unlike in that paper, we do not assume nontrivial upper bounds on internal node utilities, so we cannot expand only the nodes reached by both players.

If a reasonable *move ordering* exists over moves that approximates how "interesting" or "strong" a move is in a given position, it can be used to focus the search. Instead of expanding *all* leaves in the support of either player's strategy, we use the move ordering to judiciously pick which nodes to expand. If an internal node h in the support of at least one player's strategy has multiple unexpanded children ha, we start by only expanding those children that are in the support of *both* players' current strategies. Of the children that are not, we expand only the child that is the most "interesting", delaying the expansion of the other children to a later iteration. For our dark chess agent, the "interestingness" of a child is defined by its estimated value $\tilde{u}(ha)$, except that checks, captures, and promotions are always defined to be more interesting than other moves. This change allows us to focus our attention on parts of the game tree that are easy for the value function $\tilde{u}$ to misunderstand—namely, positions in which there are forcing moves—thereby allowing a much deeper search.

Dealing with lost particles. Upon reaching a new infoset I in a playthrough, because we are performing non-uniform iterative deepening, it is likely that some nodes in I do not appear in the subgame search tree. It is even possible that *no* node in I appears in the subgame search tree. For this reason, in addition to nested subgame solving, we maintain the exact set I (up to transpositions, as per Section 4). The set I rarely exceeds size 10^7, making it reasonable to maintain and update in real time. Let I' be the set of game nodes currently being considered by the player. We set a lower limit L on the number of "particles" (subgame root states) being considered. If $|I'| \leq L$ and $I' \subsetneq I$, then we sample at most $L - |I'|$ nodes uniformly at random without replacement from $I \setminus I'$, and add them as roots of the subgame tree. At such nodes h, our agent assumes that the opponent knows the exact node. The alternate payoff at h is defined to be $\min(\tilde{u}(h), \hat{u})$ where $\hat{u}$ is the estimate of our current value in the game, as deduced from the previous subgame solve. This alternate payoff setting prevents the agent from over-valuing states with $\tilde{u}(h)$ values that are unattainable due to lack of information. Since this results in a highly lopsided tree (the newly-sampled root states have not been expanded at all, whereas other states may have been searched deeply), on the dth iteration of the iterative deepening loop, we only allow the expansion of nodes at depth at most d unless those nodes are in the support of both players' strategies. This allows the newly-sampled roots to "catch up" to the rest of the game tree in depth.

We set $L = 200$, which we find gives a reasonable balance between achievable depth in subgame solving and representative coverage of root nodes. To prevent the set I from growing too large to manage, we explicitly incentivize the agent to discover information: for each action a available to the agent at the root infoset of the subgame, let $\mathcal{H}(a)$ denote the binary entropy of the next observation after playing action a, assuming that the true root is uniformly randomly drawn from I'. Then we give an explicit penalty of $2^{-\mathcal{H}(a)}|I|/M$ if the agent plays action a, where M is a tunable hyperparameter. In our experiments, we set $M = 10^7$. The only purpose of this explicit penalty is to prevent the agent from running out of memory or time trying to compute I; typically $|I|$ is small enough that it is a non-factor and the agent is able to seek information without much explicit incentive.

Performing particle filtering over I^∞ was suggested as an alternative in parallel work [25]. We believe that particle filtering would not work as well as our method in dark chess. If we maintained I^∞ instead of I, the L particles would have to cover the entire common-knowledge closure I^∞, not just I, which means a coarser and thus inferior approximation of I^∞. In a domain like dark chess where managing one's own uncertainty of the position is a critical part of playing good moves (since good moves in chess are highly position dependent), this will degrade performance, especially when I^∞ is large compared to I (which will typically be the case in dark chess).

Choice of subgame solving variant. The choice of subgame solving variant is a nontrivial one in our setting. Due to the various approximations and heuristics used, it is often impossible to make all margins positive in a subgame. Thus, we make a hybrid decision: we first attempt *reach-maxmargin* subgame solving [2], which is a generalization of maxmargin subgame solving that incorporates the fact that we can give back the gifts the opponent has given us and still be safe (Section 3)[1]. Using reach reasoning (i.e., mistakes reasoning) gives us a larger safe strategy space to optimize over and thus larger margins. If all margins in that optimization are positive, we stop. Otherwise, we use reach-resolving instead. This makes our agent *pessimistic on offense* (if margins are positive, it assumes that the opponent is able to exactly minimize the margin), and *optimistic on defense* (in the extreme case when all margins are negative, the distribution of root nodes is assumed to be uniform random). This guarantees that all margins are made positive whenever possible, and thus, that at least modulo all the approximations, the theoretical guarantees of Theorem 9 are maintained. We find that this gives the best practical performance in experiments.

6 Experiments

Experiments in medium-sized games. We conducted experiments on various small and medium-sized games to test the practical performance of 1-KLSS. To do this, we created a blueprint strategy for $\oplus$ that is intentionally weak by forcing $\oplus$ to play an ε-uniform strategy (i.e., at every infoset I, every action a must be played with probability at least ε/m where m is the number of actions at I). During subgame solving, the same restriction is applied at every infoset except the root, which means theoretically that it is possible for any strategy to arise from nested solving applied to every infoset in the game. The mistakes made by playing with this restriction are highly systematic (namely, playing bad actions with positive probability ε); thus, the argument at the end of Section 4 suggests that we may expect order-1 subgame solving to perform poorly in this setting.

We tested on a wide variety of games, including some implemented in the open-source library *OpenSpiel* [18]. All games were solved with Gurobi 9.0 [12], and subgames were solved using *maxmargin* solving. We found that 1-KLSS in practice always decreases the exploitability of the blueprint, suggesting that 1-KLSS decreases exploitability in practice, despite the lack of matching theoretical guarantees. Experimental results can be found in Table 1. We also conducted experiments at $\varepsilon = 0$ (so that the blueprint is an exact NE strategy, and all the subgame solving needs to do is not inadvertently ruin the equilibrium), and found that, in all games tested, the equilibrium strategy was indeed not ruined (that is, exploitability remained 0). Gurobi was reset before each subgame solution was computed, to avoid warm-starting the subgame solution at equilibrium.

The experimental results suggest that despite the behavior of 1-KLSS in our counterexample to Proposition 3, in practice 1-KLSS can be applied at every infoset without increasing exploitability despite lacking theoretical guarantees.

[1] Because we do not know a lower bound on the gifts the opponent has given us in dark chess, we use $\sum_{I'a' \prec I} (u^*(x|I'a') - u^*(x|I))$ as a gift estimate, where the values u^* are computed from the blueprint.

Table 1: Experimental results in medium-sized games. Reward ranges in all games were normalized to lie in $[-1, 1]$. *Ratio* is the blueprint exploitability divided by the post-subgame-solving exploitability. ε was set to 0.25 in all experiments, but the results are qualitatively similar with smaller values of ε such as 0.1. A description of all games can be found in the appendix.

game	exploitability of blueprint	exploitability after 1-KLSS	ratio
Kuhn poker	.0124	.0015	8.3
2x2 Abrupt Dark Hex	.0683	.0625	1.093
4-card imperfect-info Goofspiel, random card order	.171	.077	2.2
4-card imperfect-info Goofspiel, fixed increasing card order	.17	.0	∞
3-rank limit Leduc poker	.0207	.0191	1.087
Liar's Dice, 5-sided die	.181	.125	1.45
100-Matching pennies	.0013	.0	∞

Experiments in dark chess. We used the techniques of Section 5 to create an agent capable of playing dark chess. We tested on dark chess instead of other imperfect-information chess variants, such as *Kriegspiel* or *recon chess*, because dark chess has recently been implemented by a major chess website, chess.com (under the name *Fog of War Chess*), and has thus exploded in recent popularity, producing strong human expert players. Our agent runs on a single machine with 6 CPU cores.

We tested our agent by playing three different opponents: **1)** A 100-game match against a baseline agent, which is, in short, the same algorithm as our agent, except that it only performs imperfect-information search to depth 1, and after that uses *Stockfish*'s perfect-information evaluation with iterative deepening. The baseline agent is described in more detail the appendix. Our agent defeated it by a score of 59.5–40.5, which is statistically significant at the 95% level. **2)** One of the authors of this paper is rated approximately 1700 on chess.com in Fog of War, and has played upwards of 20 games against the agent, winning only two and losing the remainder. **3)** Ten games against FIDE Master Luis Chan ("luizzy"), who is currently the world's strongest player on the Fog of War blitz rating list[2] on chess.com, with a rating of 2416. Our agent lost the match 9–1. Despite the loss, our agent demonstrated strong play in the opening and midgame phases of the game, often gaining a large advantage before throwing it away in the endgame by playing too pessimistically.

The performances against the two humans put the rating of our agent at approximately 2000, which is a strong level of play. The agent also exhibited nontrivial plays such as bluffing by attacking with unprotected pieces, and making moves that exploit the opponent's lack of knowledge—something that agents like the baseline agent could never do. Game play samples can be found in the appendix.

7 Conclusions and future research

We developed a novel approach to subgame solving, k-KLSS, in imperfect-information games that avoids dealing with the common-knowledge closure. Our methods vastly increase the applicability of subgame solving techniques; they can now be used in settings where the common-knowledge closure is too large to enumerate or approximate. We proved that as is, this does not guarantee safety of the strategy, but we developed three avenues by which safety guarantees can be achieved. First, safety is guaranteed if the results of subgame solves are incorporated back into the blueprint strategy. Second, the usual guarantee of safety against *any* strategy can be achieved by limiting the infosets at which subgame solving is performed. Third, we proved that 1-KLSS, when applied at every infoset reached during play, achieves a weaker notion of equilibrium, which we coin *affine equilibrium* and which may be of independent interest. We showed that affine equilibria cannot be exploited by any Nash strategy of the opponent, so an opponent who wishes to exploit an affine equilibrium must open herself to counter-exploitation. Even without the safety-guaranteeing additions, experiments on medium-sized games showed that 1-KLSS always reduced exploitability even when applied at every infoset, and depth-limited 1-KLSS led to, to our knowledge, the first strong AI for dark chess.

This opens many future research directions: **1)** Analyze k-KLSS for $1 < k < \infty$ in theory and practice. **2)** Incorporate function approximation via neural networks to generate blueprints, particles, or both. **3)** Improve techniques for large games such as dark chess, especially managing possibly-game-critical uncertainty about the opponent's position and achieving deeper, more accurate search.

[2] That rating list is by far the most active, so it is reasonable to assume those ratings are most representative.

References

[1] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[2] Noam Brown and Tuomas Sandholm. Safe and nested subgame solving for imperfect-information games. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[3] Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[4] Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[5] Noam Brown, Tuomas Sandholm, and Brandon Amos. Depth-limited solving for imperfect-information games. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[6] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[7] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

[8] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Faster game solving via predictive Blackwell approachability: Connecting regret matching and mirror descent. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[9] Sam Ganzfried and Tuomas Sandholm. Endgame solving in large imperfect-information games. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2015. Early version in AAAI-13 workshop on Computer Poker and Imperfect Information.

[10] Sam Ganzfried and Tuomas Sandholm. Safe opponent exploitation. *ACM Transaction on Economics and Computation (TEAC)*, 3(2):8:1–28, 2015. Best of EC-12 special issue.

[11] Andrew Gilpin and Tuomas Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM*, 54(5), 2007.

[12] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2020.

[13] Samid Hoda, Andrew Gilpin, Javier Peña, and Tuomas Sandholm. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2), 2010.

[14] Eric Jackson. A time and space efficient algorithm for approximately solving large imperfect information games. In *AAAI Workshop on Computer Poker and Imperfect Information*, 2014.

[15] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Fast algorithms for finding randomized strategies in game trees. In *ACM Symposium on Theory of Computing (STOC)*, 1994.

[16] Vojtěch Kovařík, Dominik Seitz, and Viliam Lisỳ. Value functions for depth-limited solving in imperfect-information games. In *AAAI Reinforcement Learning in Games Workshop*, 2021.

[17] Christian Kroer, Gabriele Farina, and Tuomas Sandholm. Solving large sequential games with the excessive gap technique. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[18] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.

[19] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, May 2017.

[20] Matej Moravcik, Martin Schmid, Karel Ha, Milan Hladik, and Stephen Gaukrodger. Refining subgames in large imperfect information games. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[21] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[22] David J Slate and Lawrence R Atkin. Chess 4.5—the Northwestern University chess program. In *Chess skill in Man and Machine*, pages 82–118. Springer, 1983.

[23] Stockfish. https://stockfishchess.org/.

[24] Michal Šustr, Vojtěch Kovařík, and Viliam Lisý. Monte Carlo continual resolving for online strategy computation in imperfect information games. In *Autonomous Agents and Multi-Agent Systems*, pages 224–232, 2019.

[25] Michal Šustr, Vojtěch Kovařík, and Viliam Lisý. Particle value functions in imperfect information games. In *AAMAS Adaptive and Learning Agents Workshop*, 2021.

[26] Kevin Waugh. A fast and optimal hand isomorphism algorithm. In *AAAI Workshop on Computer Poker and Incomplete Information*, 2013.

[27] Brian Hu Zhang and Tuomas Sandholm. Small Nash equilibrium certificates in very large games. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[28] Brian Hu Zhang and Tuomas Sandholm. Finding and certifying (near-)optimal strategies in black-box extensive-form games. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[29] Martin Zinkevich, Michael Bowling, Michael Johanson, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2007.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Each section of the paper describes the limitations of that section.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In appendix.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] In appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] In appendix.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [Yes] In the appendix.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [Yes] In the appendix.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] There was no compensation.