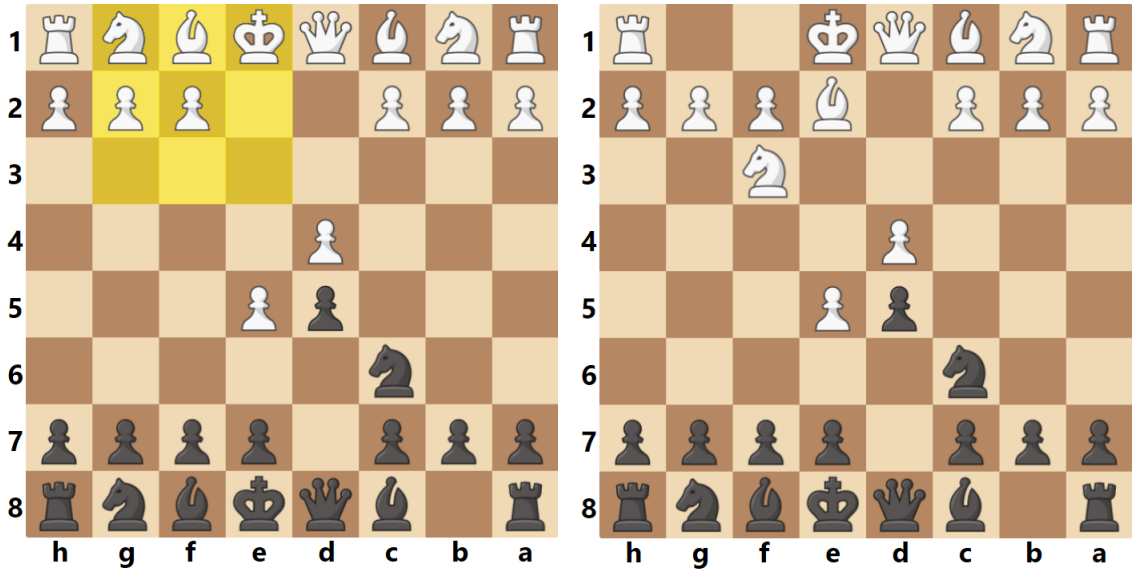
Abstract

Reconnaissance Blind Chess (RBC) is a variant of Chess in which players can only sense a 3×3 grid within the 8×8 Chess board after each move of the opponent. This restriction makes RBC a game of imperfect information, with many new challenges to address. In this paper, we present `Fianchetto`, our agent that won the NeurIPS 2021 RBC competition by a large margin. `Fianchetto` builds on the publicly-available code base of `StrangeFish` (the 2019 winner), and includes four major changes: (1) a faster, "batched" board evaluation function, (2) Bayesian belief updating, (3) incentives for strategic RBC moves, and (4) a mechanism to regulate decision making based on the *size* of the information set. We present a series of experiments to validate these changes, which we supplement with an analysis of actual games from the 2021 competition. We discuss how elements of our solution may generalise to other games of imperfect information.

1. Reconnaissance Blind Chess

Advances in computational Chess have been beacons for the progress of the field of AI, beginning with the early efforts of Turing (1953). Efforts over nearly half a century culminated in a demonstration of super-human play in the 1990s (Campbell, Hoane Jr., & Hsu, 2002); today's neural reinforcement learning techniques can *learn* to play even better by training for just a few hours (Silver, Hubert, Schrittwieser, Antonoglou, Lai, Guez, Lanctot, Sifre, Kumaran, Graepel, Lillicrap, Simonyan, & Hassabis, 2018). In spite of its forbidding complexity, Chess is still a game of *perfect* information. Many real-world tasks face the significant challenge of performing decision making with imperfect (or hidden) information. Games such as Scrabble (Sheppard, 2002) and Poker (Moravčík, Schmid, Burch, Lisỳ, Morrill, Bard, Davis, Waugh, Johanson, & Bowling, 2017; Brown & Sandholm, 2018a) have been test beds for research on imperfect information games.



(a) Sense region (b) Sense result

Figure 1: Change b to show only the sensed pieces, before the move.

Reconnaissance Blind Chess (RBC) (Gardner, Lowman, Richardson, Llorens, Markowitz, Drenkow, Newman, Clark, Perrotta, Perrotta, Highley, Shcherbina, Bernadoni, Jordan, & Asenov, 2020) is a recently-proposed imperfect-information variant of Chess. In RBC, each player controls traditional Chess pieces on a regular Chess board, but cannot directly observe the locations of their opponent's pieces. Rather, at the beginning of each turn, the player selects a 3×3 region of the 8×8 board, and is shown the pieces occupying the 9 corresponding cells, along with their positions (see Figure 1 for an illustration). After this *reconnaissance move*, the player performs a *regular move*: moving one of their pieces (as in Chess). An interesting addition to the set of regular moves is a *pass* move (not present in Chess), which leaves the positions of all the pieces unchanged. The opponent is not informed which region was scanned or which move was played; they must gather this information indirectly through their own reconnaissance moves. The only information a player obtains from a regular move is whether they captured some opponent piece, or that the move was illegal (such as a pawn moving forward into an occupied cell). A full specification of the rules of RBC is provided in Appendix A.

To accurately model the decision making task in RBC, the most appropriate formal abstraction would be the two-player zero-sum imperfect-information extensive form game (CITE), with a 2-stage turn per player to incorporate their reconnaissance and regular moves. The technical challenge in this setting is uncertainty not only over the true board state, but also over the opponent's history (equivalently their belief). Given the impracticality of computing with this full-blown model, one might settle for an approximation in which the *opponent* is assumed to have full observability of the task, and moreover, they follow a fixed, known strategy. Even with this drastic (and potentially debilitating) simplification, the decision making task remains a Partially Observable Markov Decision Problem (POMDP). POMDPs are hard to solve when they have long horizons [], in fact undecidable over an infinite horizon []. Games between well-matched RBC players can last 30–50 moves, with a total of 15 minutes clock time per player. It is quite evident that there is no hope of playing optimally. On the other hand, any agent for RBC must be *engineered* to address a complex decision making problem, with significant constraints on time and memory, against a range of opponents. In this regard, RBC poses a conjunction of several unique challenges.

1.1 Challenges of RBC

We highlight the challenges of RBC (1) when compared with other imperfect information games, and (2) when viewed as a generalisation of Chess.

1.1.1 Comparison with other imperfect information games

Significant advances have been made on several imperfect information games in the last two decades, including Scrabble (Sheppard, 2002; Richards & Amir, 2007), Poker (Moravčík et al., 2017; Brown & Sandholm, 2018a), Bridge (Ginsberg, 1999, 2001), Dou Dizhu (Whitehouse, Powley, & Cowling, 2011), and Battleship (Silva & Vinhas, 2007; Clementis, 2013). All these games have a substantial amount of *public* information. For example, in Scrabble, although the tiles held by the opponent are hidden, those placed on the board (typically much larger in number) are visible to both players, as also are the actions of both players. When there is a substantial amount of public information, it directly guides decision

making, and also reduces uncertainty over the hidden information. In RBC, each player's stream of information is almost entirely private. When a player performs a reconnaissance move, the opponent is neither given the positions of pieces in the sense region, nor conveyed which region was sensed. The only information given to both players in the course of the game is the positions of captures (again not indicating which opponent piece has captured). The virtual absence of public information in RBC results in very large information sets. In general a player does not know for certain which pieces are still in play, or even whether the move they intend to play is a legal one.

There are a handful of other games in which there is very little public information for decision making. Kriegspiel (Wetherell, Buckholtz, & Booth, 1972; Favini, 2010; Ciancarini & Favini, 2007, 2010; Russell & Wolfe, 2005) is another variant of Chess in which the opponent's pieces are invisible; before playing any move, a player consults a referee, who answers whether the move is legal. In Dark Chess (REFERENCE), a player only knows the positions of their own pieces and the positions they can reach through legal moves. Phantom Go (Cazenave, 2005; Wang, Zhu, Li, Hsueh, & Wu, 2017) is an imperfect information variant of Go built along the same lines as Kriegspiel for Chess. In Kriegspiel, Dark Chess, and Phantom Go, there is no explicit move for sensing, like the reconnaissance move in RBC. The judicious use of the reconnaissance move, so as to complement the strategy for regular moves, is a new challenge to tackle in RBC.

The recent string of successes on Poker (Brown & Sandholm, 2018b; Moravčík et al., 2017)) are a notable breakthrough in the context of imperfect information games. On the one hand, the Poker game tree contains a substantial amount of public information. Moreover, the effective game size can be compressed by many orders of magnitude through *abstraction*, which is a way of aggregating states that are equivalent for decision making (Burch, Johanson, & Bowling, 2014; Brown & Sandholm, 2018b; Šustr, Kovařík, & Lisý, 2019). A decomposition into sub-games of reduced complexity facilitates the computation of equilibrium strategies (Zinkevich, Johanson, Bowling, & Piccione, 2007; Brown & Sandholm, 2018b). Moreover, it appears feasible in Poker to generalise across *belief states* for learning behaviour (Brown, Bakhtin, Lerer, & Gong, 2020), thereby removing the non-Markovian influence of hidden information. Chess does not have any obvious form of "local regularity" in the state space: boards that differ in the position of only a single piece, which occupies adjacent cells in these boards, would typically have very different evaluations. Moreover, in RBC, the preponderance of private information yields belief states over a very large and diverse set of states. Markowitz, Gardner, and Llorens (2018) estimate that the average number of opponent states within an agent's information set in RBC is in the order of 10^{68}; the same quantity is about 10^3 for 2-player Texas Hold 'Em Poker and 10^{14} for 6-player Texas Hold 'Em Poker. It remains a challenge yet to extract predictive features from belief states in RBC.

1.1.2 Size and Complexity

A widely adopted method by Shannon (1950) compares the game size based on the number of possible states that a player can encounter in a game. As shown in Table 10, the number of states in RBC is in the same order of No-Limit Poker and Go which is way larger than Chess or Limit Heads-Up Texas Hold'em Poker. The average number of possible states in

the information set also serves as a good metric for comparing the complexity of games as it represents the difficulty in evaluation of a perceived state. Table 11 (Markowitz et al., 2018) shows the approximate values for several games. The average number of states in the information set for RBC (considering the different possible states based on opponent knowledge) exceeds that of Six-player-No-Limit Poker (Guo, Wang, Qi, Qian, & Zhang, 2020).

Game	Size
Texas Hold'em Poker	10^{13}
Chess	10^{43}
RBC	10^{139}
No-Limit Poker	10^{162}
Go (19×19)	10^{170}

Table 1: Approximate size of games

Game	Game States
Heads-Up No-Limit	1083
Chess	1
RBC	1.3×10^{68}
No-Limit Poker	6.4×10^{14}
Go (19×19)	1

Table 2: Approximate number of possible opponents states in an information set

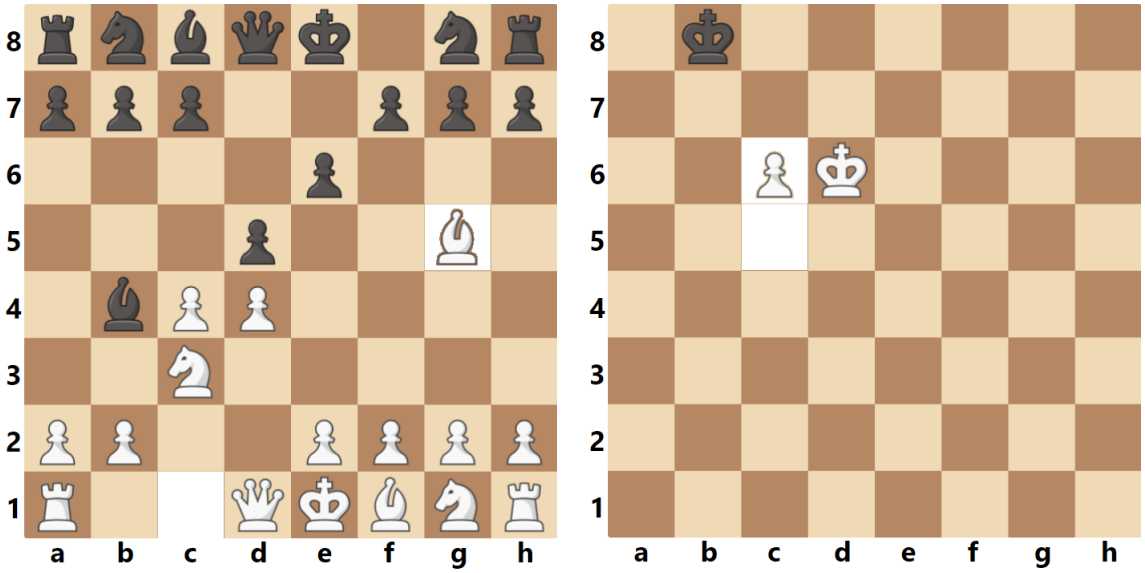
RBC games between good players often go beyond 25 moves per player; the size of the information set (the set of boards consistent with the observation history) usually varies in the range 10^3–10^5. Unlike Poker, there appears to be no easy way to compress the state representation through abstraction. Markowitz et al. (2018) estimate the number of plausible information sets in RBC to be about 10^{139}, which is roughly 10^{93} times the estimated number of states in Chess.

1.1.3 RBC as a Generalisation of Chess

It is but natural to seek solutions for RBC that exploit the vast amount of knowledge available on Chess as a game (gained over many centuries) as well as Chess-playing programs (gained over many decades). Indeed RBC can be viewed as a proper generalisation of Chess along two dimensions. If sensing is restricted to an $m \times m$ grid, RBC is run with $m = 3$, whereas Chess uses $m = 8$. The other change is the inclusion of an additional "pass" move in RBC. Both changes introduce pitfalls in the transfer of knowledge.

Board evaluation is perhaps the most critical primitive in determining the success of a game-playing program. While one might reasonably expect a "good" position in Chess to also be good for RBC (and vice versa), it is relatively common to encounter contradictions. Figure 2 provides two illustrations of the divergence between Chess and RBC evaluations,

one arising from incomplete observability, and another due to the pass move. Hence, although evaluation functions such as Stockfish (Romstad, Costalba, & Kiiski, 2021) and AlphaZero (Silver et al., 2018) perform extremely well for Chess, they can often mislead decision making in RBC. Moreover, since an RBC information set typically contains a few thousands of boards, an agent must contend with the additionally challenge of aggregating their individual evaluations.



(a) Bad in Chess, good in RBC.

(b) Good in Chess, bad in RBC.

Figure 2: In (a), the move of white's bishop from c1 to g5, although being a terrible move in Chess, might be a good move in RBC, as it might potentially win the queen in the next move—quite possible if the opponent misses to sense g5. In (b), moving the white pawn from c5 to c6 is a winning move in Chess. This move, however, leads to a *defensible* position for black in RBC as black can move its king from b8 to c8 in the next turn and repeatedly play the *pass* move thereafter.

With a reliable evaluation function not readily available, one could explore the possibility of *learning* one specifically for RBC. However, this prospect is also beset by technical challenges that are not faced in Chess. In principle, the Markovian "state" (an information set) at any juncture comprises not only an agent's (known) sequence of moves and outcomes, but also the opponent's (unknown) sequence of moves and outcomes. XYZ estimate that the number of states encountered in a typical RBC game is in the order of 10^{139}. The challenge is not so much the size of the state space—much larger than Chess at 10^{43}, but still much smaller than Go at 10^{170}—but the agent's lack of access to important state information.

It is well-known

- learning, mcts, etc. are complicated by hidden state. mcts kriegspiel. negative results.

Achieving optimality for imperfect information games becomes far more difficult as the entire game tree (including all the unreachable states) also needs to be considered for approximating an optimal action. Hence, Monte Carlo tree search (MCTS) approaches (Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, & Colton, 2012), which were an essential part for algorithm of AlphaZero (Silver et al.,

2018), cannot be directly applied to imperfect information games. Variants like Information set MCTS (ISMCTS) (Cowling, Powley, & Whitehouse, 2012) is developed to work for imperfect information games but it does not provide any optimality guarantees.

Several AI approaches have been developed for chess in recent years, for example —Google DeepMinds's AlphaZero algorithm (Silver, Schrittwieser, Simonyan, Antonoglou, Huang, Guez, Hubert, Baker, Lai, Bolton, et al., 2017) and the Stockfish engine (Romstad et al., 2021). But since they do not account for uncertainty, they can't be applied directly to RBC without modification.

START FROM HERE. ADD 1-2 more points

–practically hard to experiment against. changes against one can be bad against others (non-transitive). not players on-line.

write about the state number and the assumption on not modeling opponent sensing here.

RBC is also novel in comparison to other imperfect-information variants of Chess. For instance, in Kriegspiel, state information is gathered through the regular moves themselves, with no need for decoupled sensing. On the other hand, reconnaissance moves in RBC must be selected judiciously, as they are significant to information gathering and hence to the player's overall performance.

In Banqi (REFERENCE), which is sometimes played on one half of the usual 8×8 Chess board,

Are there other games which have perfect and imperfect information game counterpart

- simulation soccer
- Go vs Phantom Go

1.2 Contribution

An RBC competition for automated agents has been conducted every year since 2019: as a part of the NeurIPS competition track in 2019 and 2021, and as an on-line competition in 2020. In this article, we describe our agent `Fianchetto`, which won the NeurIPS 2021 competition by an aggregate win-loss record of 931–89 against 17 other agents, with at least a 2:1 win-loss ratio against each of them. `Fianchetto` is built on the publicly-available code base of `StrangeFish`, which won the 2019 competition.

The success of `Fianchetto` owes to innovations in all key areas of game play: board-evaluation, uncertainty-modeling, strategic play, and information-management. We describe each of these components of `Fianchetto`. Where relevant, we also examine the applicability of our ideas more broadly within the arena of imperfect information games, in the spirit that *"... one valuable way to advance the field is to study complete agents in specific, complex domains, with the ultimate goal of drawing general lessons from the specific implementations."* (STONE 2007). In order to directly facilitate more research on RBC, we also release the source code of `Fianchetto` as an accompaniment to this article[1].

After first describing the `StrangeFish` baseline in Section 2, we present the main elements of `Fianchetto` in Section 3. Results from the NeurIPS 2021 RBC, which we review in Section 6, lend strong support to the overall strength of `Fianchetto`. However, we must

1. 1234

note that its dominance over a contemporary crop of players is no proof of `Fianchetto` playing (even close to) optimally. On the other hand, we believe the agent can be improved in many ways; we outline some directions in Section 7.

more about higc etc. here, the field is lacking.

2. Baselines

In this section, we describe `StrangeFish` which forms the basis for `Fianchetto`. We also highlight aspects of some other agents that have participated in RBC competitions.

2.1 `StrangeFish`

The victory of `StrangeFish` in the 2019 competition was followed by a public release of its source code (Perrotta & Perrotta, 2019). The flow of control in `StrangeFish` is depicted schematically in Figure 3. Our own approach to develop `Fianchetto` was to analyse the play of `StrangeFish` and update modules assessed to have the most room for improvement.

Primitives. `StrangeFish`'s sense and move strategies both depend on functions to associate (1) a weight with each board in the information set, and (2) a numeric score with an entire information set. Under `StrangeFish`, both functions are derived from the Stockfish Chess engine (Romstad et al., 2021). For a board s (reached by the opponent's move), the weight ascribed is $\text{weight}(s) = \text{squash}(\text{StockFishScore}(s))$, where the squashing function is a sigmoid that maps the score provided by Stockfish to $(0, 1)$. Weights on states in an information set induce a probability distribution, which is useful to aggregate per-state computations into expectations. To obtain a score for an information set I, a separate score is calculated for each board $s \in I$ using a combination of $\text{StockfishScore}(s)$ and some hand-coded rules. In turn, the score for I is a convex combination of the best, worst, and expected score among boards $s \in I$.

Sense strategy. The input to decision making on each move is the information set I computed immediately after the agent's previous turn. I comprises all the boards that can possibly be the true underlying state at the moment, given the sequence of outcomes of the agent's (regular and reconnaissance) moves, and allowing for the opponent to have played arbitrary moves. The first step is to expand I to $I^O \supseteq I$ by simulating all possible opponent moves. Since I^O can be large, and there is a time limit on the game, `StrangeFish` samples a subset of boards $I^R \subseteq I^O$ uniformly at random from I^O, with $|I^R|$ decided by the remaining time (specified later in the section).

Now, for each of the 36 non-boundary squares i that can be sensed, there will be some finite number of possible outcomes p depending on the boards present in I^R. `StrangeFish` simulates sense move i and collates all the boards from I^R that would yield the same outcome p into a set $I^S_{i,p}$. To decide which sense move i to actually perform (in the real world), `StrangeFish` considers two aspects: the ability of i to discover important activity on the board (detecting which would imply a large change of score), and the effect of i on reducing uncertainty in the information set. To this end, a quantity α_i is defined to be the magnitude of the difference between the score of I^R and the average expected score of all qualifying $I^S_{i,p}$. A quantity β_i, which denotes the expected reduction in the size information set by

sensing i, is also calculated based on the sizes and weights of qualifying sets $I^S_{i,p}$. The sense move eventually chosen is $i^\star = \text{argmax}_i(\alpha_i + C \cdot \beta_i)$, where C is a constant.

Move strategy. The reconnaissance move helps prune the information set to $I' \subseteq I^O$. I' only contains boards s' that are consistent with the observation z from sensing. As in the sense strategy, a random set I'^R is sampled. `StrangeFish` assigns weights to the boards in I'^R by squashing the Stockfish score. Each contemplated move results in a next state s'' for each board $s' \in I'^R$. The set of all such s'' for any given move (akin to an information set) is scored using the primitive described earlier. The move to be played is selected at random from moves whose score is in a top quantile. Randomisation counters the uncertainty caused by large information sets, with the added benefit of avoiding deterministic move patterns that opponents could exploit.

Time management. On each turn, `StrangeFish` allots a 1/30 fraction of the remaining time (initially 15 minutes) to the current move. Of this allotted chunk, 80% is given to the sensing strategy. A fixed quantum of 0.001s is provided for move score evaluation. The size of the subset of boards sampled at each turn (that is, $|I^R|$) is a linear function of the time allotted for the move.

The flowchart in Figure 3 is annotated with labels from the set $\{\text{V1}, \text{V2}, \text{V3}, \text{V4}\}$, which refer to the versions of `Fianchetto` that are described in Section 3, and indicate the modules updated in each of these versions.

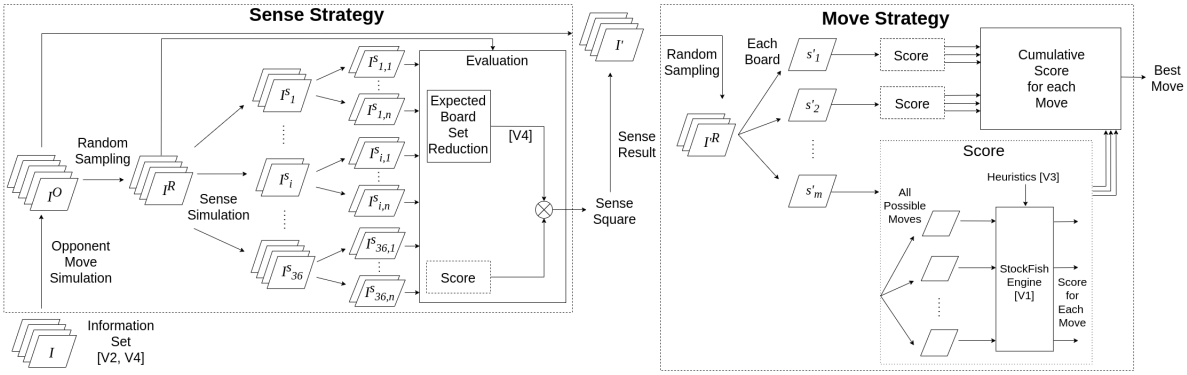


Figure 3: Control flow in `StrangeFish` (Perrotta & Perrotta, 2019). On each turn the agent begins with the information set I from the previous turn, which the sense strategy maps to a sense square (equivalently, the reconnaissance move). Sensing prunes the information set to $I' \subseteq I$. Based on I', the move strategy returns a regular move to be played. Internals of the sense and move strategies are explained in the Section 2. The labels V1, V2, V3, V4 in the figure refer to versions of `Fianchetto` (see Section 3), and are placed alongside modules in `StrangeFish` that are modified in these versions. Figure: draw Move Strategy Box below Sense Strategy. Enlarge, use full page.

2.2 Other agents in RBC competitions

Before proceeding to describe `Fianchetto`, we briefly introduce some other agents that have taken part in previous RBC competitions. Descriptions of these agents are taken from the relatively sparse published literature on RBC. It must be noted that many agents are under constant development: one cannot be sure that their competition entries match their descriptions, although some overlap may be expected.

Along with `StrangeFish`, Gardner et al. (2020) describe `Oracle`, `trout`, `attacker`, and `random`. These four agents, intended as baselines, have been developed by the organisers of the RBC competitions at the Johns Hopkins University Applied Physics Laboratory (Gardner et al., 2020). These agents may all be downloaded from the RBC server and executed locally. `Oracle` follows the same control flow as `StrangeFish`, but incorporates some minor changes. Its reconnaissance move aims to minimise the expected size of the resulting information set, while accommodating some special rules for sensing threats to its King. Unlike `StrangeFish`, it selects its regular move based on a simple (unweighted) majority. `trout` maintains a *single* board estimate based on the last observation from each square, choosing the move recommended by Stockfish for this estimated board. For positions with a past capture, or the possibility of a capture, `trout` senses over the capture square; otherwise it selects (at random) a 3×3 region that does not contain any of its own pieces. `attacker` selects an opening picked uniformly at random from a set of hand-coded ones, playing it as long as possible (and thereafter keeps playing the *pass* move). `random` selects reconnaissance as well as regular moves uniformly at random from those available.

Other than `StrangeFish`, one strong competitor that we find described in the literature is `penumbra` (Clark, 2021a), which won the 2020 on-line RBC competition. This agent maintains a belief state, and performs decision time planning using roll-outs. The key component of the approach is a feature-based abstraction of information states (called "synopses"). `penumbra` also uses a specialised model of each opponent, which is trained from the logs of each opponent's games on the RBC server.

The `LaSalle` agent achieved the second rank in the 2019 competition, and its successor `LA-Q` the second rank in the 2020 competition (Highley, Funk, & Okin, 2020; Blowitski & Highley, 2021). In contrast with the agents previously described, these agents maintain a separate probability distribution for each *piece*, reflecting the agent's belief about where that piece might be located on the board. `LaSalle` and `LA-Q` did not participate in the NeurIPS 2021 competition.

3. `Fianchetto`

In this section, we present the construction of `Fianchetto` through a sequence of changes to `StrangeFish`, which itself is now denoted V0 (version 0). The versions of `Fianchetto` presented are V1–V4. We evaluate each version in two ways: (1) a match against V0, conducted locally, and (2) matches against 5 agents played on the on-line RBC server from 22 November–4 December 2021 (subsequent to the NeurIPS competition). While the match against V0 is a controlled experiment involving a known opponent, matches played on the server convey the spirit of actual tournament play, against opponents that we do not control (and could possibly be changing between games). Consistent with the tournament format, all matches are of 60 games, split equally as black and white. All the experiments were

Version	V0	SF2	Or	tr	att	ran	Overall
V0	30-30	16-44	39-21	57-3	56-4	60-0	258-102
V1	24-36	13-47	36-24	58-2	50-10	59-1	240-120
V2	39-21	35-25	46-14	59-1	58-2	60-0	297-63
V3	48-12	32-28	47-13	59-1	58-2	59-1	303-57
V4	48-12	35-25	47-13	59-1	60-0	60-0	309-51

Table 3: Win-loss scores from a 60-game match between row agent and column agent. V0 is the same as `StrangeFish`; its row is populated using its last 60 games in a specified window in November-December 2021 on the RBC server. The column for V0 is obtained locally, whereas all other columns (SF2 = `StrangeFish2`, Or = `Oracle`, tr = `trout`, att = `attacker`, ran = `random`) are obtained from games played on the server.

performed by playing 4 simultaneous games of RBC on a 24 core cloud VM with 2 Nvidia Tesla T4 GPUs. Results are compiled in Table 3. Unfortunately, even the locally-run experiments are not perfectly replicable since `StrangeFish` uses multiprocessing and its logic depends on the remaining time, and hence on the system load. However, note that *logs* of all the games can be furnished upon request.

3.1 V1: Batched board evaluations

Recall from Section 2 that on every turn, `StrangeFish` evaluates every possible move on every board in the current information set by sending the resulting board to Stockfish. Observing that this step is the main computational bottleneck in each turn, we explore a more efficient alternative for board evaluation. Lc0 (Pascutto & Linscott, 2021), which is a pre-trained, open-source, neural network-based Chess engine, presents itself as a convenient replacement for Stockfish. Unlike Stockfish, which requires a separate call for each (board, move) pair, Lc0 takes a board position as input, and returns scores for every possible move on this board in a *single* call. Internally, Lc0 operates as a policy network, generating activations for each possible move through a forward pass with the board provided as input.

As shown in Table 4, the time for a single call to Lc0 well exceeds that for a call to Stockfish. However, a more favourable comparison emerges if we assume that there are 30 moves on average in a position, and while Stockfish needs to be called for each move, Lc0 needs to be called only once per board. The advantage swings decidedly in favour of Lc0 if we use a GPU installation, which allows for multiple *boards* to be evaluated in parallel. This "batching" feature is a natural one for Lc0 since all it needs is a pass through a pre-trained neural network. We are not aware of a similar possibility to parallelise calls to the more complicated Stockfish engine. Lc0 offers a variety of pre-trained networks—of different sizes, and hence different strengths—for the game of Chess.

The one used to obtain our results in Table 4 is the "T75" network (ID `w752050`) (Pascutto & Linscott, 2021), which has 15 convolutional blocks with 192 filters. This network can evaluate approximately 1024 Chess boards in a single batch, while taking up 1 GB of RAM on the GPU. To study the effect of the network size on game performance, we run a comparisons with a larger network (ID `w6d379e55`) and a smaller network (ID `wf6f9ab83`). Results collated in Table 5 display an interesting trend. In the first row of the table are the Bayesian Elo ratings of these three networks in a Chess tournament played among each

	Time per engine call (s)	Effective no. of boards evaluated / s Without batching	With batching
Stockfish	0.005	200	3200 (16 Threads)
Lc0 (1GB GPU)	0.321	93	95232 (batch size = 1024)

Table 4: Comparison of throughput of Stockfish and Lc0, performed on a desktop machine with Intel Core i5-4690 CPU@3.50GHz and Nvidia GeForce GTX 980 GPU.

other, also including three variants using Stockfish (with search depths 4, 6, and 8). Indeed we observe a consistent increase in ratings with the size of the network. In the second row are Bayesian Elo ratings on RBC, obtained from hundreds of games played by each agent on the RBC server on 14, 15, and 26 December 2021, respectively (against several opponents). In this case, notice that the medium-sized network achieves a much higher rating than the smaller and larger networks. The result is not surprising, since the larger network is likely overfitted to Chess. The medium-sized network is thus preferable both for its performance on RBC, and the quicker evaluations it provides.

	Small n/w	Medium n/w	Large n/w
Chess Rating	1416	1453	1572
RBC Rating	1248	1502	1350

Table 5: Comparison of ratings of different-sized Lc0 networks (explained in text). The large network is best for Chess, but the medium-sized one is best for RBC.

As apparent from Table 3, the switch from Stockfish (V0) to Lc0 (V1) actually results in a slight *worsening* of performance on RBC. However, as we see next, the speedup provided by batched evaluations enables a whole extra step of probabilistic expansion of the search tree, with the associated probabilities themselves coming from Lc0's policy network.

Put in the comparison with chess in the appendix.

3.2 V2: Persistent board belief

Against any *fixed* opponent, the game of RBC is equivalent to a POMDP, with states being the boards accessible to the agent, and the actions those of sensing and moving. If b is the belief state (a probability distribution over states in the information set) after the agent's move, and observation z is received before its next regular move, then Bayes rule yields the following update to obtain the new belief state $b' = \mathbb{P}\{\cdot|b, z\}$.

$$\mathbb{P}\{s'|b, z\} = \sum_a \mathbb{P}\{s'|b, a, z\} \sum_s \mathbb{P}\{a|s\}\mathbb{P}\{s|b\};$$
$$\mathbb{P}\{s'|b, a, z\} \propto \mathbb{P}\{z|s', a\} \sum_s \mathbb{P}\{s'|s, a\}\mathbb{P}\{s|b\}.$$

Here s' is the new state (after the opponent's move), a is the opponent's action (regular move), with s iterating over states in the agent's information set. Although the opponent's

strategy $\mathbb{P}\{a|s\}$ is unknown (admittedly it is simplistic even to assume such a strategy for an opponent, who must in reality act according to their own history/belief), we *approximate* it by the policy encoded in the Lc0 neural network. Note that it is feasible to perform this computation (which involves a sum over a) on every turn only because Lc0 directly returns a probability distribution over all actions anyway.

Our approach of persisting a belief vector over time is in stark contrast with `StrangeFish`. Rather, `StrangeFish`'s calculations on each turn assume that the probability the opponent plays action a from state s is proportional to squash(StockfishScore(s')), where s' is the state reached by playing a from s. Crucially, in this scheme, the aggregate probability of playing a does not depend on the prior belief placed on s. Moreover, since many reachable states from the current information set are often similar, they are all ascribed with nearly the same probability.

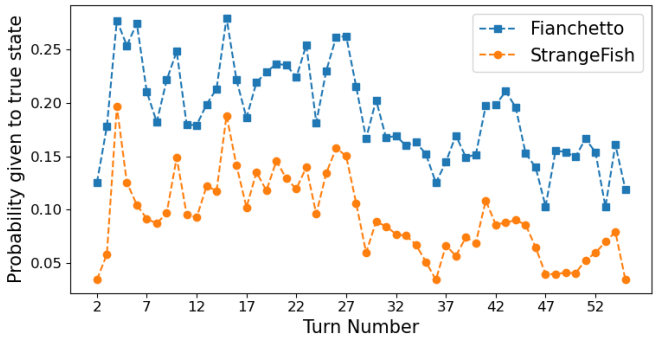


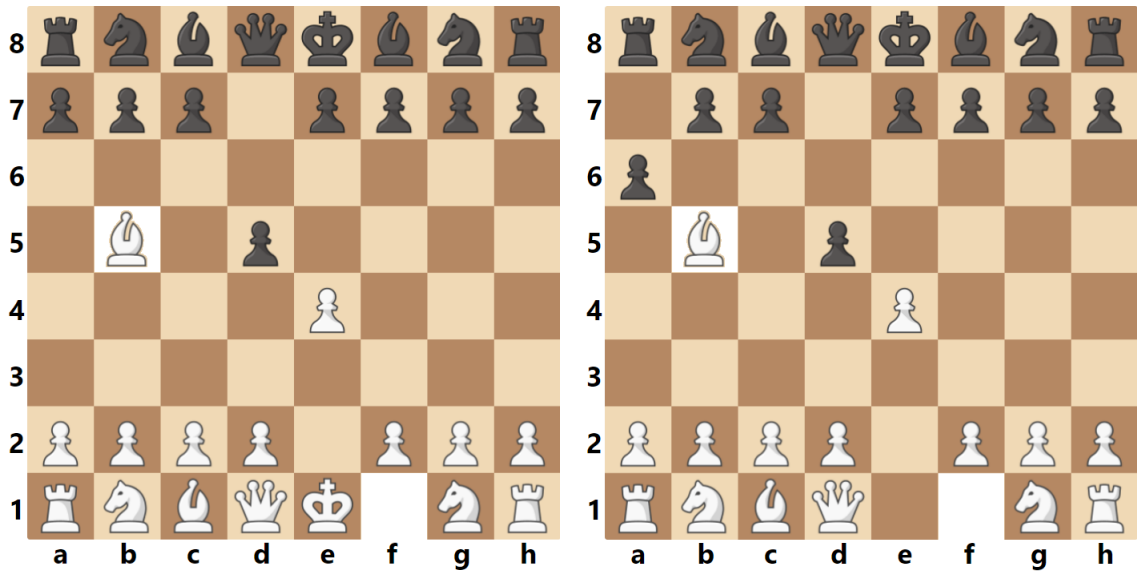
Figure 4: Probability associated with true state by `Fianchetto` (V2) and `StrangeFish`, averaged over hundreds of games played by V2 on the RBC server from 16–24 December 2021.

Of course, the effectiveness of our principled belief update depends on the accuracy of Lc0 as the opponent model. We do find empirical validation. Figure 4 compares the probabilities given to the true board state by `Fianchetto` and `StrangeFish` as the game progresses; observe that `Fianchetto` is consistently more accurate. Table 3 also confirms that the switch from V1 to V2 yields a significant boost to scores against all the opponents tested. We proceed to describe further adjustments to `Fianchetto` to fit our models even better to RBC.

3.3 V3: Strategic RBC moves

The lack of full observability opens up some surprising opportunities in RBC. Since our underlying evaluation function (Lc0) is tuned for Chess, we add a layer of incentives to promote certain classes of RBC moves. Given a board as input, the Lc0 network gives a real-valued activation for each possible move. We add a move-specific constant as an incentive to this activation, before obtaining probabilities by performing a softmax operation.

Piece-wise dynamic sneak reward. "Sneak attacks", such as the moves shown in Figure 5 are checks to the opponent king, which if they go undetected, can yield a potentially winning move. `StrangeFish` provides a constant reward for such moves. However, since the risk incurred by such moves is variable, we provide incentives that depend on the piece being moved and the threat it faces.



(a) Low risk (b) High risk

Figure 5: A "sneak attack" move (bishop from f1 to b5) that is low-risk on the left board, but risks loss of the bishop on the right board.

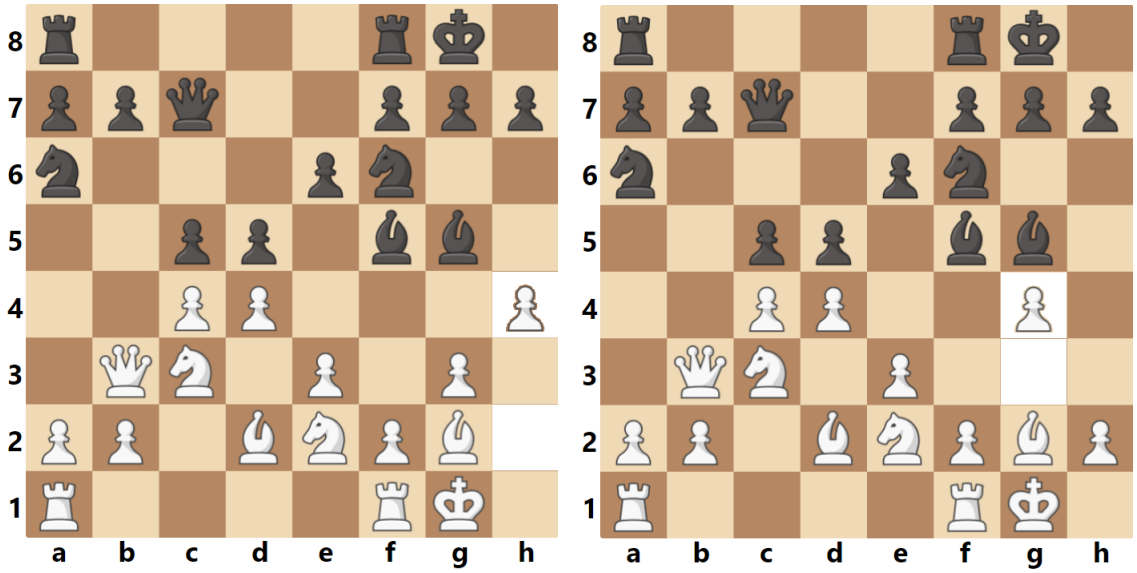
Pawn attacks. Similar to sneak attacks on the opponent king, we incentivise our pawns to attack opponent pieces, again tuning the reward based on the associated risk. Figure 6 provides an illustration. An opponent who is aware of this strategy on part of `Fianchetto` can lay a trap to weaken our pawn structure and profit from it. Although agents from the 2021 NeurIPS competition were not sophisticated enough to do so, we see that in principle, `Fianchetto` can be exploited.

Faraway defense. We employ yet another tactic that is especially useful in endgame, when the board is wide open and there is a high probability of check on our king. When our king is likely under check, we incentivise moves to block the check from afar by moving another piece (itself supported), rather than moving the king itself. The rationale is that the opponent is quite likely to sense the position of the king in the next move, and try to capture it since it has not moved. If our ploy works, we stand a good chance of capturing the attacking opponent piece. Figure 7 provides an illustration.

Exact details of the rewards used for these strategic moves are available in our code. We observe from Table 3 that these changes (in V3) yield a substantial improvement over V2 when playing `StrangeFish`, although there are no clear gains against agents played on the RBC server.

3.4 V4: Regulating information set size

Although `Fianchetto`'s success owes a great deal to the principled approach to maintain a belief vector (introduced in V2), we observe empirically that poor decisions are often taken when the information set size becomes very large (equivalently, there is a high degree of uncertainty on the true state). This phenomenon is observed especially towards the endgame,



(a) Low Risk (b) High Risk

Figure 6: The pawn move from h2 to h4 in (a) and that from g3 to g4 in (b) can both potentially capture an opponent bishop. Unlike the moved pawn in (a), the moved pawn in (b) is unprotected making it a high risk move which gets lesser incentive

when the information set grows very rapidly, and moreover, certain adversaries repeatedly play the *pass* move to trigger its exponential growth. To keep the information size regulated, we implement two measures. The specific numbers and thresholds mentioned were tuned to obtain an improved performance against locally available bots. There is scope for tuning these to a optimal combination to achieve even better performance.

Dynamic weightage on uniform probability. Firstly, to reduce the bias of the Chess policy obtained from Lc0, we set the belief on the information set I as a mixture of the distribution obtained from a Bayesian update, and the uniform distribution; the combining parameter $\lambda \in [0, 1]$ is a function of the information set size. For small information sets, almost no weightage is given to the uniform distribution since the belief is reliable (an information set of size 1 corresponds to Chess). A large information set suggests that the opponent is playing quite differently from our Lc0 model of them, and hence our Bayesian belief updates could be misinformed. For the *pass* move, the belief is set using only the uniform probability, since the policy from the Lc0 engine does not contain the *pass* move.

$$\text{Belief}(I) \propto \lambda \cdot \text{Bayesian Belief}(I) + (1 - \lambda) \cdot \text{Uniform Distribution}$$

$$\lambda = \begin{cases} 0.8 & size(I) < 5000 \\ 0.5 & 5000 < size(I) < 20000 \\ 0 & size(I) > 20000 \end{cases}$$

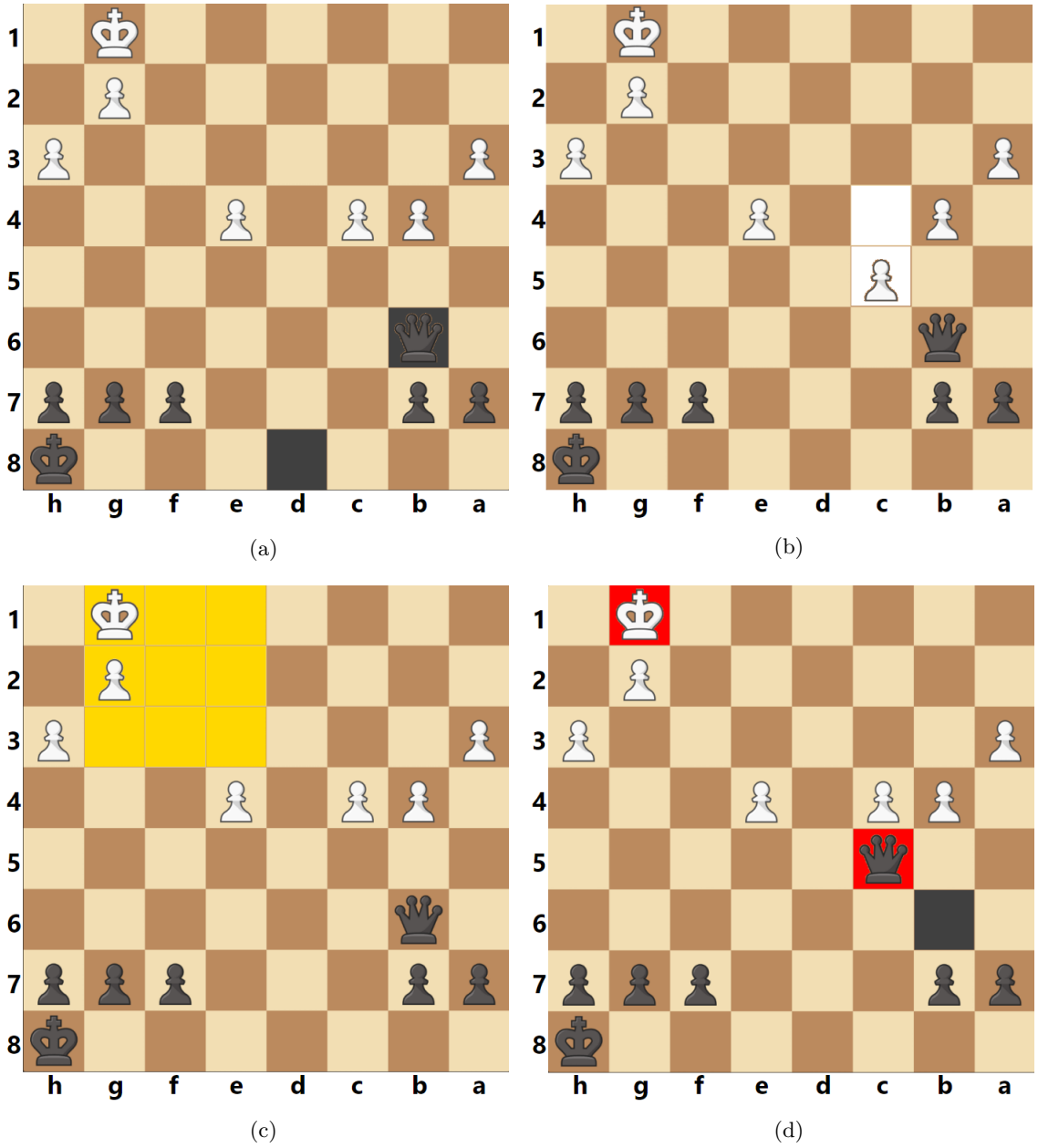


Figure 7: Supoose black moved its queen from d8 to b6 as shown in (a). To protect the king from the check, a pawn move from c4 to c5 shown in (b) is incentivised with the idea that the opponent is highly likely sense the region where the king is in (c), (as it can be a potentially winning information) and finding it still being in the line of attack would then try to capture it, but in the process it will fall for losing its own piece as shown in (d).

Dynamic information set reduction. One additional change we make to contain the information set size is while calculating the score for the sense square. Unlike `StrangeFish`'s static average between the move score resulting from the sense and the expected information set size reduction, we incorporate a weighted average of the two with a *dynamic* weight

associated with the information set reduction score. When the information set size becomes large, we (disproportionately) encourage sensing that will reduce the information set size. With small information sets, the agent seeks to play the best possible move based on the available information. For each square

sense score $\propto$ mean sense impact + $func(size(I))$*expected set reduction

$$func(x) = \begin{cases} 1 & x < 5000 \\ 1 + 49 * \dfrac{(x - 5000)}{45000} & 5000 < x < 50000 \\ 50 & x > 50000 \end{cases}$$

Figure 8 provides definitive evidence that the average information set size against literally every opponent becomes smaller when we incorporate the changes described above (in V4). Observe the pronounced gains against `attacker`, which repeatedly plays the *pass* move once its mainline strategy terminates. Modest performance gains are also observed in Table 3 for V4 over V3.

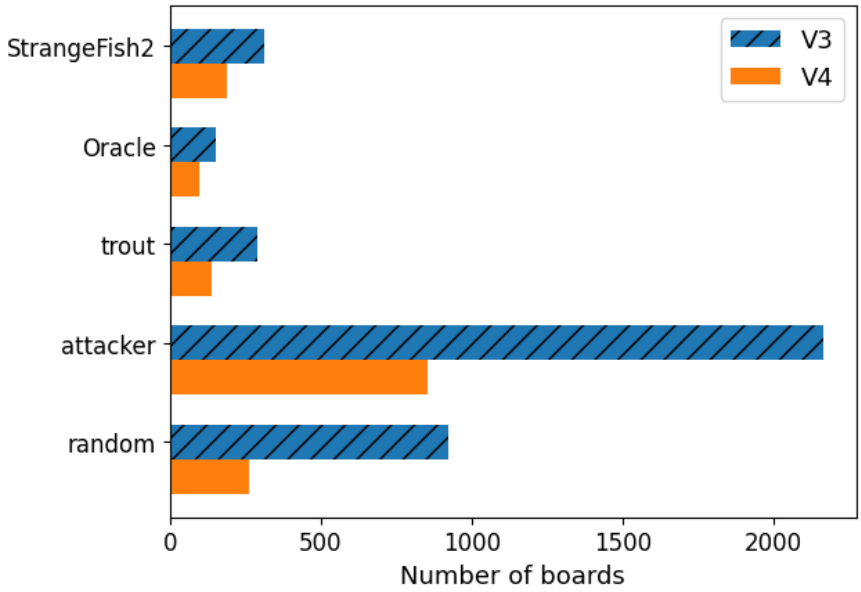


Figure 8: Average information set size in games played on the RBC server (which are also reported in Table 3).

4. State-based scoring system

LCzero is a state-of-the-art chess engine that uses a deep neural network to evaluate positions. The current move score in LCzero is defined as a weighted sum of the move score and probability of the board s in the information set I.

- Generate a list of moves to evaluate and denote it as set A.

- Define $T(s, a)$ as transition function $T : (S, A) \rightarrow S'$ to depict next board position on playing action a in state s.
- Currently the compound score for move $a \in A$ for the Information set I is defined as

$$score(a) = \sum_{s \in I} P(s)Q(s, a)$$

  where the $Q(s, a)$ is the output of LCzero engine. The drawback with this schema is that the $Q(s, a)$ is inconsistent meaning that a bad move was being given a better score because LCzero's $Q(s, a)$ are not accurate for moves that are not going to happen in a normal play of chess game i.e less represented in training games. Hence the bishop blunder is given a much more negative Q score than a queen blunder(which are the less represented/bad moves on that board position).
- Hence we came up with a more consistent state-based scoring system where we define the score for every move a as $score(a) = \sum_{s \in I} P(s)V(s')$ where $s' = T(s, a)$.We noticed that $V(s) \in [-1, 1]$ is much more consistent compared to Q(s, a) as seen below. To further improvise the move score and take into opponent's perspective also use a min-max-based approach
- Also generate an opponent list of moves for every board position s'(in reply to our move a) and denote it as B_a.
- The improvised scoring system is defined as

$$score(a) = \sum_{s \in I} P(s)min_{s''_a \in S''_a} V(s''_a)$$

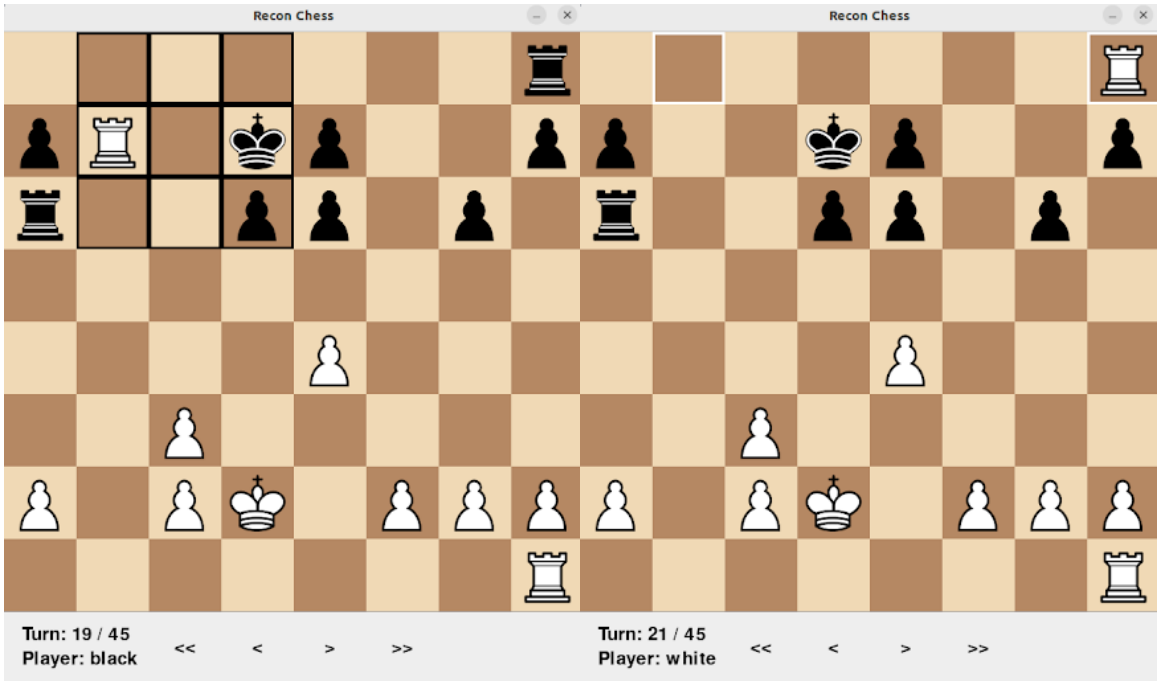
  where $s' = T(s, a), S''_a : [s''_a = T(s', b_a)\ for\ b_a \in B_a])$ and b_a is generated as opponents' perspective moves in reply to our move a.
- In the corner case where suppose it is white to play and the black king is already in check instead of evaluating this position using the LCzero engine, we assign $V(s) = -1$.
- Eventually, choose the maximum compound score as

$$move = argmax_{a \in A} score(a) = argmax_{a \in A} \sum_{s \in I} P(s)min_{s''_a \in S''_a} V(s''_a)$$

  .

5. Inconsistency Examples

During the Tournament 2022 Fianchetto instead of playing king from d7 to c6 which is the best move in the position, played king from d7 to e8 blundering the rook on h8 on the next move. White rook gave a check on b8 skewering the black rook and king and capturing on the next move after the black king moved back to d7. The move scores $Q(s, a)$ for $d7e8 : 0.8854877948760986, d7c6 : 0.4970516562461853$ which shows that d7e8 blunder is given a much higher score compared to d7c6 the best move, whereas the state based score



(a) Original Position (b) Blunder

Figure 9: Inconsistency example 1

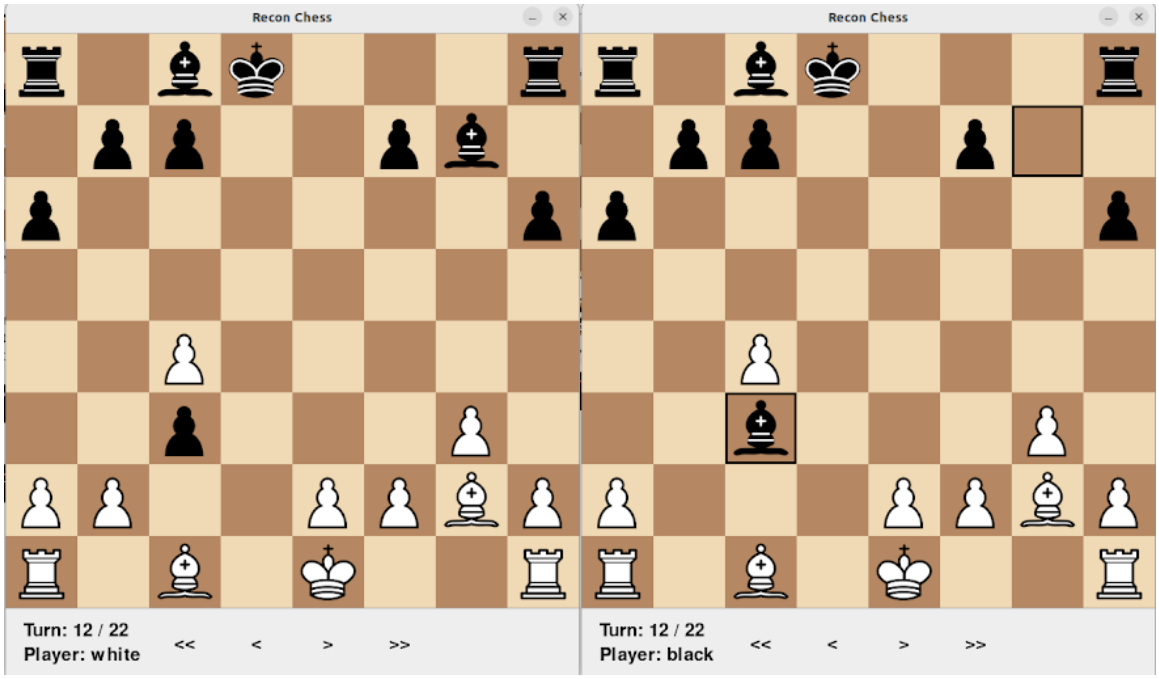
$V(s)$ for $d7c6 : -0.83482993, d7e8 : -0.96879441$ is much more consistent giving the best move d7c6 a higher score.

Another example is where instead of playing the best move rook from a1 to b1 it played pawn b2 takes c3. It is a blunder since the bishop from g7 takes c3 and comes with a check followed by the rook capture on a1. The move scores $Q(s, a)$ for $a1b1 : 1.5521409511566162, b2c3 : 2.047072172164917$ which shows that b2c3 blunder is given a much higher score compared to a1b1 the best move, whereas the state based score $V(s)$ for $a1b1 : -0.2059148, b2c3 : -0.68169534$ is much more consistent giving the best move a1b1 a higher score.

6. NeurIPS 2021 Tournament

The NeurIPS 2021 RBC competition was conducted during 21–23 October 2021 as a round-robin tournament among 18 competing agents, with each pair of agents playing 60 games against each other (equally split as black and white). The eventual rankings were based on the Bayesian ELO rating (Coulom, 2008), which is explained for the reader's convenience in Appendix C. `Fianchetto`[2] topped the competition with a rating of 1759, compared

2. The version of `Fianchetto` that competed in the tournament has two minor changes from V4: (1) it used the then-latest weights published on-line for our LCO network (ID w4737df84), and (2) it included some hand-coded "special cases" for the reconnaissance move. Subsequent tests have shown that the performance of the competition version is virtually indistinguishable from that of V4; at any rate its complete source code is a part of our public release.



(a) Original Position (b) Blunder

Figure 10: Inconsistency example 2

to 1662 of the second-ranked `StrangeFish2` (Perrotta & Perrotta, 2019) and 1584 of the third-ranked `penumbra` (Clark, 2021a). Indeed `StrangeFish2` and `penumbra` are updated versions of the agents that won the 2019 and 2020 competitions, respectively. Not only did `Fianchetto` win at least two-thirds of its games against these and every other opponent, it registered an overall win ratio of 91%. Detailed statistics from the NeurIPS 2021 competition are provided in Appendix **??**. Table 6 shows `Fianchetto`'s win-loss record against the other 17 agents that took part in the NeurIPS 2021 RBC competition. Agents are ranked based on the Bayesian Elo rating.

Analyses performed on game logs from the tournament show trends for `Fianchetto` that are characteristic of expert play. In Figure 11, we observe a positive correlation between the average number of moves per game and the strength of the opponent. The same figure also shows that `Fianchetto` enjoys a positive material advantage against all the opponents. Note that although `Kevin`, the opponent against whom the advantage is least (0.06, hence not visible), is ranked fourth in the tournament, `Kevin` has the best head-to-head record against `Fianchetto`. Games against many low-ranked players end quickly, well before a substantial advantage is accumulated.

7. Future Work

In this paper, we explain the dominant performance of `Fianchetto` in the NeurIPS 2021 RBC competition in terms of its innovations to different aspects of game play. Our justifica-

Agent	ELO rating	`Fianchetto`'s wins-losses
`StrangeFish2`	1662	41-19
`penumbra`	1584	40-20
`Kevin`	1544	40-20
`Oracle`	1503	51-9
`Gnash`	1454	49-11
`Marmot`	1315	56-4
`DynamicEntropy`	1299	59-1
`wbernar5`	1219	58-2
`Frampt`	1208	59-1
`GarrisonNRL`	1140	59-1
`trout`	1127	59-1
`callumcanavan`	1066	60-0
`attacker`	1049	60-0
`URChIn`	854	60-0
`armandli`	777	60-0
`random`	753	60-0
`ai_games_cvi`	288	60-0
Overall	–	931-89

Table 6: NeurIPS 2021 RBC tournament results. `Fianchetto`'s ELO rating from the tournament was 1759.

tions double up as proof that there is still a long way to go to build optimal, non-exploitable agents for RBC. Immediate steps could focus on addressing current weaknesses.

The current reliance on trained Chess evaluation functions such as Stockfish and Lc0 is more based on convenience than on principle. Given the many differences between RBC and Chess, there is a growing need for training evaluation functions specifically for RBC. Negotiating hidden state while so doing remains a technical challenge (Clark, 2021a). To some extent, rollout-based policies can offset the deficiencies of the evaluation function (Ciancarini & Favini, 2010); we are yet to incorporate this useful idea into `Fianchetto`. From the perspective of the game itself, endgame (with few pieces and an exploding information set) becomes a formidable challenge for current approaches, and merits more attention from a modeling perspective. We hope that the public release of our code base can help foster research in all these directions.

Researchers developing agents for RBC face the practical challenge of not having a large, diverse set of opponents to test against.

conclusion early days more relevant

bounded optimality russell and others. far far away.

Neural fictitious self-play (NFSP) is an approach which can be applied to two-player zero-sum games with imperfect information (Heinrich & Silver, 2016) and it avoids hand-crafted game abstractions.

Trains Q, when even roll-outs are untrustworthy.

why brown and sandholm twice?

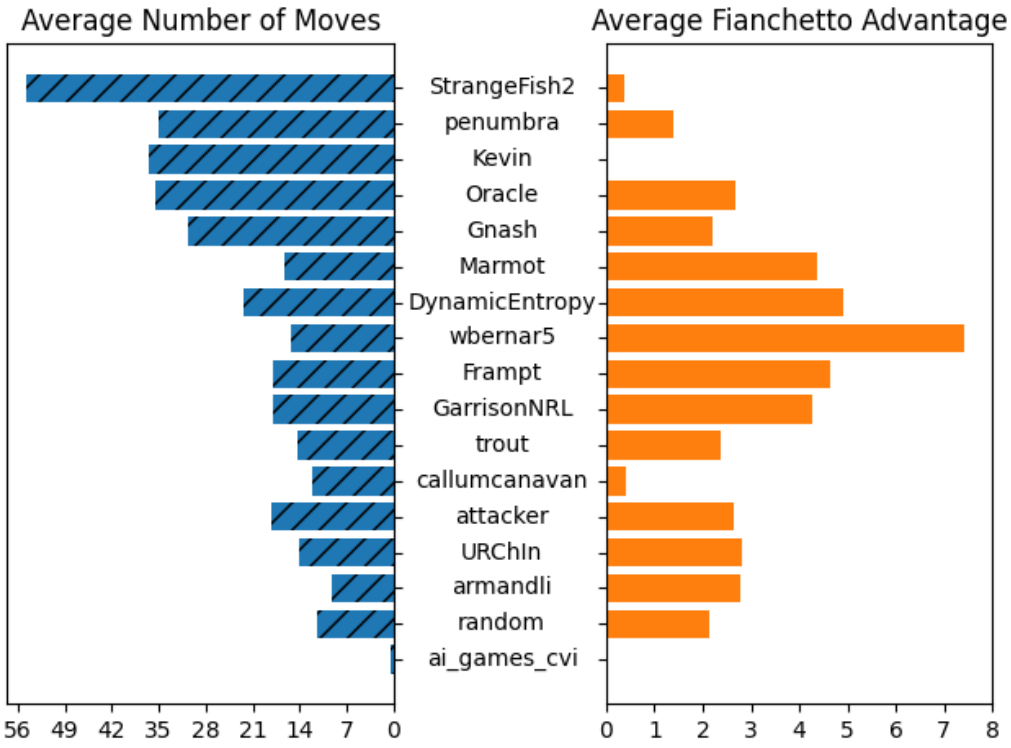


Figure 11: Averages of the number of moves per game (left panel) and `Fianchetto`'s material advantage per move (right panel) in its games against all opponents from the NeurIPS 2021 competition (arranged in decreasing order of ranking top to bottom). The `ai_games_cvi` agent never played any moves. Material advantage is based on the following points: Pawn 1, Bishop/Knight 3, Rook 5, Queen 9.

8. Introduction

Reconnaissance Blind Chess (RBC) is a Chess variant in which players only have partial visibility of the opponent's pieces. Before each move, a player can view the opponent's pieces only inside a 3x3 region of their choice. Each player receives no information about checks to their king or the identity of captured pieces, forcing the victory condition to be the enemy king's capture. Thus, RBC is a game of imperfect information. In this project, we have worked on improving Fianchetto - an RBC agent developed by Mohammad Taufeeque, Nitish Tongia, and Shivaram Kalyanakrishnan that achieved first place in the NeurIPS 2021 RBC Tournament. We have developed a new version of Fianchetto (hereafter referred to as version 5 or V5) to compete in the NeurIPS 2022 RBC Tournament, held in October. Fianchetto achieved second place in the tournament, with an aggregate win-draw-loss record of 837-168-15.

9. Background and Previous Work

The first four versions of Fianchetto were developed last year, with version 4 being the final version that took part in the 2021 tournament. A succinct description of how Fianchetto

V4 works is given below. A detailed description can be found in the manuscript OurAgent: Speed, Belief, Guile, Caution to Win at Reconnaissance Blind Chess. At each turn, Fianchetto V4 chooses a 3x3 region to sense. The sense region is decided based on two scores which are weighted together - the mean sense impact and the expected board set reduction. The mean sense impact is a measure of how much critical information is present in a sense region, whereas the expected board set reduction is a measure of how much the size of the information set decreases due to a sense region choice. The sense region that maximises this weighted score is chosen. Now, the agent chooses a move to be played based on the sensed information. It models the opponent according to Leela Chess Zero (Lc0) (Pascutto & Linscott, 2021) and gives a probability to each board in the information set. A smaller information set is sampled according to this probability. The agent simulates every legal RBC move on each board in the sampled set and gets a score according to its evaluation function. The agent computes the minimum, maximum and average score for each move across the sampled board set, which is weighted together in a convex sum. A move is chosen randomly from the highest-scoring moves according to this weighted sum. In a game of RBC, each player gets 15 minutes to play their moves. This year's iteration of the NeurIPS Tournament introduced two rule changes. Firstly, in addition to the 15 minutes, players get a bonus increment of 5 seconds per move. Secondly, games can now end in a draw according to the 50-move rule in chess. These changes allow for more computationally intensive strategies, which we have tried in our experiments. We conducted our experiments on the John Hopkins University RBC server, which allows bots to play practice games against each other. The server uses four baseline bots - Oracle, Trout, attacker and random to benchmark the performance of agents.

10. Experiments, Observations and Results

We analysed hundreds of games played by Fianchetto V4 to find and analyse weaknesses and limitations in its play. Fianchetto V5 attempts to fix some of these limitations, three of which are mentioned below. Chapter 4 explains experiments that did not sufficiently improve performance to be a part of V5.

10.1 Checkmate Detection

In our experiments on the RBC server, we noticed that Fianchetto V4 occasionally lost to the baseline bots Oracle and Trout in the same sequence of five moves [Figure 12]. This was happening due to a limitation in the evaluation function. The evaluation function penalises the agent for playing moves that put the agent's king in check or checkmate but it does not penalise moves that allow the opponent to deliver checkmate on the next turn. We changed the evaluation function to give a -200 penalty for moves that blunder checkmate in one. This change fixed the problem - we no longer lost to the above bots in this move sequence. Unfortunately, this change required a 1 step lookahead which was too computationally expensive to do on every move, and we noticed an increase in losses due to timeout. We decided to perform checkmate detection only against the opponents Oracle and Trout, during the first 10 moves of the game. This led to an increased win rate against Oracle (78% to 83% over 60 games) and a perfect score against Trout.

Figure 12: Example of 5 move loss to Oracle. Here Fianchetto does not sense the white bishop's arrival at the c4 square, leading to checkmate and forced king capture.

10.2 Null Move Probability Boosting

RBC allows players to play a pass or a null move where they can choose not to move any piece. This is a strategic move that prevents the opponent from getting any new information, causing their information set to become exponentially large. In a chess position, the opponent may have roughly 30 legal moves on average, and them playing a null move may only eliminate 10 of these moves. This would cause the information set to grow by a factor of 20 for every null move played. This particularly becomes a problem against opponents like attacker and Trout who start playing only null moves after a certain move threshold. We noticed that we were losing some games to these opponents due to timeout, as processing this large information set became computationally infeasible. We solved this problem by increasing the probability of the opponent having played a null move if the information set crosses a certain threshold, leading to a perfect score against attacker in all subsequent experiments.

10.3 Defensive Play

One of the major limitations of V4 is that it models opponents according to Lc0. While this proved to be an extremely useful strategy in last year's iteration of the competition, this year it was exposed as a weakness by strong opponents such as Kevin and Châteaux. The reason for this is that Fianchetto gives a low probability to bad chess moves, assuming that the opponent will not play them. Such moves may be good in RBC if not detected by the opponent [Figure 13].

While we do not yet have a solution to this problem, V5 achieves improvements by giving a larger weight to the minimum score of a move on a board in the information set. V4 gives a weight of 0.7 to the average score of a move over all boards in the information

Opponent	Weight for min move	Weight for avg move
StrangeFish2	0.7	0.3
Kevin	0.8	0.2
ROOKie	0.8	0.2
JKU-CODA	0.8	0.2
uccchess	0.9	0.1
Marmot	0.9	0.1
Châteaux	0.99	0.01

Table 7: Opponent-specific weights used for defensive play.

Figure 13: Example of a game between Fianchetto (white) and Châteaux (black). Here Châteaux plays Qd1 - a move that gives up the queen in chess. However, it is not detected by Fianchetto as it is a low-probability chess move, leading to the king's capture on the next move.

set and a weight of 0.3 to the minimum score, whereas V5 uses opponent-specific weights [Table 7]. The result is more defensive play - V5 is better at sensing low-probability threats.

11. Other Experiments

11.1 Fast Divide Factor Analysis

We observed that we could achieve a better performance against certain bots by sampling a much smaller fraction of the information set according to the modelled probabilities. We theorize that for bots whose playing style is more similar to chess engines our model is more accurate, allowing us to save on computational time by dividing the size of the information set to be sampled by a fast divide factor (DF). We ran experiments for different values of a DF and found the best values for each opponent [Table 8].

Opponent	Fianchetto win rate for different fast divide factors		
	1	**25**	**50**
StrangeFish2	0.63	0.55	0.68
Oracle	0.78	0.73	0.82
ROOKie	0.93	0.80	0.85
JKU-CODA	0.88	0.93	0.92
attacker	1	0.91	0.89
Trout	1	0.88	0.91
random	1	1	1

Table 8: Win rates of Fianchetto for different values of fast divide factor against various opponents. Each cell in the table is an average across approximately 50 games.

These results, however, are not very reliable as some of the opponents were under development and in constant flux during the experiments. As we neared the tournament, opponents like ROOKie and StrangeFish2 seemed to change their strategy considerably, causing the results to no longer be valid. Additionally, it is possible that the results have a high amount of variance as the number of games played against each opponent was not sufficient. For these reasons, we chose not to include the fast divide factor analysis in Fianchetto V5.

11.2 Threat Avoidance Strategy

The idea is to try to achieve positions with the minimum number of possible opponent threats. A position with a high number of possible opponent threats is extremely difficult to play as there are several choices of sense regions that could potentially reveal critical information. Additionally, the definition of a threat in chess and RBC may not be the same as some bad chess moves are actually good RBC moves. The optimal strategy becomes unclear with increasing uncertainty in positions. We tried to give more weightage to moves that lead to safer positions, i.e. positions with fewer possible enemy threats.

We defined a threat as an enemy move that attacked or captured the agent's pieces. Checks also fall into this category as they attack the agent's king. Failure to sense threats would lead to loss of material or the loss of the game. In order to quantify the loss of material, we computed the Static Exchange Evaluation (SEE) [Chess Programming Wiki] at the square of each capture.

This strategy had many weights and parameters that required tuning, and we could not find a setting that improved the performance of V4. More work is required to evaluate the efficacy of this strategy.

Fianchetto (1644)	**Score**
StrangeFish2 (1762)	21-34-5
Kevin (1623)	31-29
Châteaux (1621)	18-42
ROOKie (1551)	37-18-5
Oracle (1465)	49-9-2
Marmot (1329)	52-8
JKU-CODA (1283)	51-9
DynamicEntropy (1194)	58-2
SomeRegret (1184)	55-5
Trout (1116)	60-0
attacker (1099)	59-1
GarrisonNRL (1039)	57-3
uccchess (1025)	49-8-3
random (893)	60-0
arandombot (598)	60-0
srcork (590)	60-0
uccch (581)	60-0
Overall	**837-168-15**

Table 9: NeurIPS 2022 Tournament Results. The Bayesian ELO rating of each bot is shown in parenthesis next to its name.

12. NeurIPS 2022 RBC Tournament

The NeurIPS 2022 RBC Tournament was held virtually from 19th-22nd October 2022. 18 agents competed and played 60 games against each agent. Each agent was ranked according to a Bayesian ELO rating [Coulom, 2008]. Fianchetto finished in second place with an ELO rating of 1644 [Table 9].

12.1 Known Limitations and Issues

The tournament exposed a number of weaknesses of Fianchetto. The biggest weakness is the vulnerability to sneak attacks as elaborated on in chapter 3.3. This was thoroughly exploited by the bots Châteaux and Kevin. Although we were able to counter Kevin to some extent through defensive play, we were never quite able to defend perfectly against Châteaux's aggressive play. A limitation to playing defensively is that the idea breaks down in open positions. There are too many possible threats to sense in the endgame, for example. Fianchetto resorts to sensing near its king at every turn, leading to it having no information about the location of enemy pieces. There are also cases where Fianchetto senses correctly but still plays an incorrect move [Figure 14]. This is likely due to an unknown bug in the code.

At a broader level, there needs to be a change in the assumption that opponents can be modelled by chess engines, as optimal strategies in RBC deviate considerably from optimal

Figure 14: Example of Fianchetto V5 (white) blundering its queen against StrangeFish2 (black). The highlighted sense region is the 3x3 grid centred on d7 and the move played by white was Qxc6. Fianchetto captures the pawn despite having information that it is protected by the enemy queen.

strategies in chess. Perhaps attempts could be made to design a custom evaluation function for RBC.

13. Conclusion and Future Work

The first step before proceeding further would be to design quality tools for debugging and visualization of RBC games. It would be extremely useful to know why the agent chose a particular sense region and move, by visualizing the scores given to every board in the information set. Developing such tools is part of our future work. Next, we plan to work on addressing the limitations and issues with the goal of building a more robust agent.

Abstract

In this seminar, we discuss the challenges and potential improvements for Fianchetto, an Artificial Intelligence (AI) agent designed to play Reconnaissance Blind Chess (RBC), an imperfect information variant of Chess. Despite its strong performance in recent NeurIPS RBC Tournaments, Fianchetto's strategy remains susceptible to exploitation. To address this issue, we survey existing Game Theory techniques as potential solutions for enhancing Fianchetto's decision-making process under uncertainty. We provide an overview of RBC and Fianchetto's existing strategy, followed by an introduction to relevant Game Theory concepts, with a focus on Partially Observable Markov Decision Processes and Imperfect Information Extensive Form Games. We review existing agents and algorithms for similar problems. The objective of this research is to identify effective approaches that can be applied to Fianchetto and other AI agents facing similar challenges in imperfect information games.

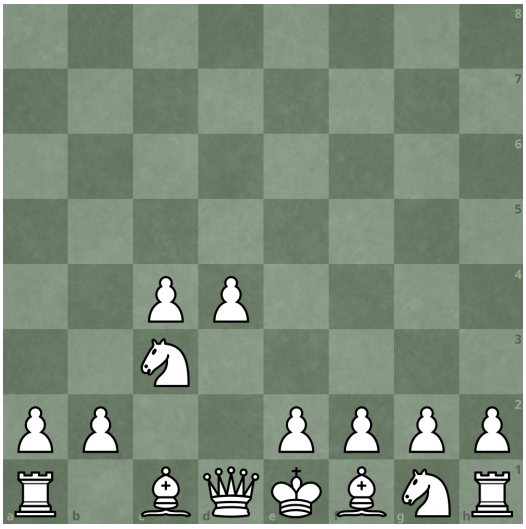


Figure 15: Players cannot see their opponent's pieces.

Figure 16: Players can choose a 3×3 region on the board and receive information about the opponent's pieces inside that region.

14. Background

In this section, we describe the necessary theory to model games such as RBC. We start by giving a brief overview of the rules of RBC and then describe how Fianchetto works. Next, we explain two separate approaches to model RBC - as an Imperfect Information Extensive Form Game (IIEFG) (a Game Theoretic approach) and as a POMDP (a Reinforcement Learning approach). In order to define an IIEFG, we first define the perfect information setting, i.e. an Extensive Form Game.

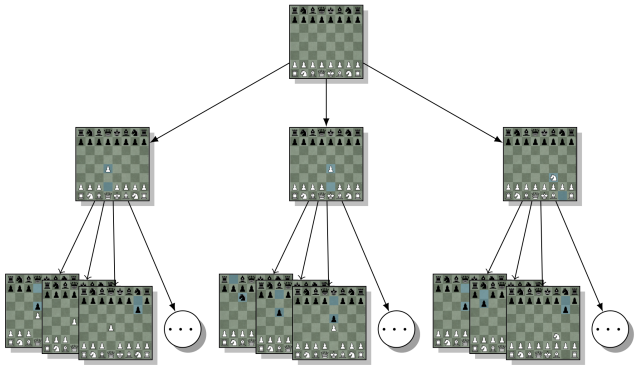


Figure 17: Game Tree of Chess, an Extensive Form Game.

14.1 The Rules of Reconnaissance Blind Chess

RBC follows the rules of Chess with respect to the setting and movement of the pieces. In RBC however, players cannot see their opponent's pieces or the moves that the opponent makes, as shown in Figure 15. The only information available about the position is through the selection of a 3×3 region on the board before each move. A depiction of the same is given in Figure 16. Players must use this information to build a belief about the true state and attempt to fulfil the objective of the game - capturing the opponent's king. Players are not notified about checks and the identity of captures - only that a capture has taken place.

In addition to the moves of Chess, RBC also introduces a null move. Players can choose not to move any of their pieces or 'pass'. The advantage of playing a null move is that it can cause the opponent's uncertainty about the position to increase - due to the fact that the opponent is not receiving any new sense information. Similar to the 50-move rule in Chess, a game of RBC ends as a draw when there is no pawn push or capture for 50 moves. Players get 15 minutes of clock time each to make their moves, with an increment of 5 seconds every time they make a move. Players may lose by timeout when they run out of time.

14.2 A Brief Description of Fianchetto's Policy

Fianchetto models RBC as a Partially Observable Markov Decision Process (POMDP) (Åström, 1965) by assuming that the opponent is a part of the environment. This assumption implies that we fix a stochastic policy for the opponent and use it to calculate a belief probability distribution across states in the information set at each turn. Fianchetto uses its own policy to model the opponent. As noted above, this is a poor way to model opponent play - the opponent's play may significantly deviate from Fianchetto's strategy.

Fianchetto's policy can be described as follows. Each action consists of two parts: a sense action and a move action. The selection of the sense region depends on the combination of two weighted scores: the mean sense impact and the expected board set reduction. The mean sense impact evaluates the amount of important information within a sense region, while the expected board set reduction measures the decrease in the size of the information

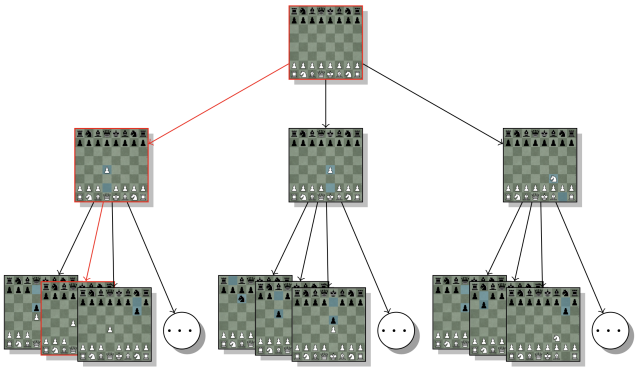


Figure 18: A path from the root to any leaf specifies a game of chess.

set due to a chosen sense region. The sense region with the highest weighted score is ultimately selected.

Using the information received from sensing, Fianchetto computes the states that comprise the information set and assigns them a probability using a POMDP belief update. Fianchetto uses Leela Chess Zero (Lc0) (Pascutto, Gian-Carlo and Linscott, Gary,), a chess engine, to score all possible moves on each board in a sampled subset of boards in the information set. Each score is also weighted by the board probability of the board on which the move was made. These scores are combined using a convex combination of the minimum, maximum and average scores across the subset. The moves that score above a certain threshold are chosen, and a move is randomly picked from this set.

14.3 Extensive Form Game

An Extensive Form Game (EFG) (Maschler, Solan, & Zamir, 2013b) consists of n players sequentially making moves and can be specified by a game tree with a root, as shown in Figure 17. Each node defines the state of the game the players are currently in. Each sequence of moves by the players specifies a path from the root to the leaf in this game tree, as shown in Figure 18. This sequence of moves defines a single game. Each leaf node in the game tree must have a defined n-tuple of payoffs for each player. Players are aware of the state they are in, the payoffs defined at the leaves, the actions available to each player at each state and the knowledge that each player has at every state.

14.4 Imperfect Information Extensive Form Game

An IIEFG (Maschler, Solan, & Zamir, 2013c) modifies an EFG with the following assumptions. Players may not be aware of the opponent's move, or their own state in the game tree. They use a set of observations to narrow the game state down to a member of a set of nodes at the same level in a game tree, known as an information set, as shown in Figure 19. Players are however aware of the payoffs available at the leaf nodes of the game tree. Thus an EFG is an IIEFG where each information set has only one member. This formulation can be used to exactly describe RBC.

The solution to an IIEFG is given by a Perfect Bayesian Equilibrium (PBE) (Maschler, Solan, & Zamir, 2013a). This is a refinement to the Nash Equilibrium concept in Normal

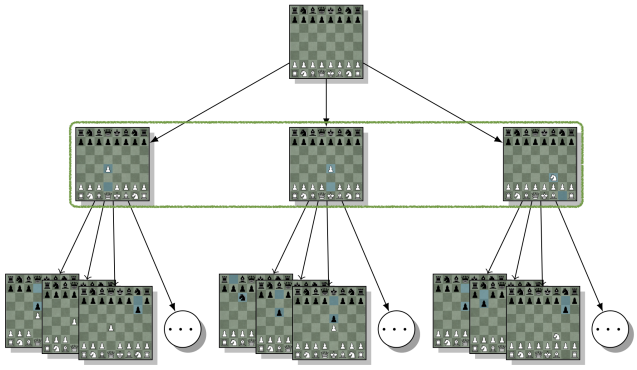


Figure 19: Information Set in an Imperfect Information Extensive Form Game.

Form Games (NFG) (Maschler, Solan, & Zamir, 2013d), which gives a strategy for each player that no player can deviate from to improve their payoff. Every EFG can be converted into an NFG representation, however, this representation has an action space that is exponential in the number of states of the game. Hence for games with exponentially large game trees such as Chess or Poker, approximating the PBE is a tough challenge.

14.5 Partially Observable Markov Decision Process

A Partially Observable Markov Decision Process (POMDP) is a Markov Decision Process (MDP) where the agent does not have access to the true state it is in. Instead, it must rely on a series of observations to choose its actions. More formally:

An MDP can be specified by defining the following:

- $\boldsymbol{\mathcal{S}}$: a set of states,
- $\boldsymbol{\mathcal{A}}$: a set of actions,
- $\boldsymbol{\mathcal{T}}$: a transition function from $\boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \to \boldsymbol{\mathcal{S}}$,
- $\boldsymbol{\mathcal{R}}$: a reward function from $\boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \to \mathbb{R}$, and
- $\boldsymbol{\gamma}$: a discount factor.

A POMDP extends this definition by defining:

- $\boldsymbol{\Omega}$: a set of observations, and
- $\boldsymbol{\mathcal{O}}$: an observation function from $\boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \to \boldsymbol{\Omega}$.

POMDP solution techniques typically also define $\boldsymbol{\mathcal{B}_t}$: a belief probability distribution over the set of states at time step $\boldsymbol{t}$. These definitions are for a single agent and its interactions with its environment. In the context of RBC however, we have two agents playing

against each other in a zero-sum game. To model this problem by a POMDP, we assume that the second agent is a part of the environment and its behaviour is captured within the belief distribution. After each move in RBC, we make a sense move that provides us with an observation regarding the position of the opponent's pieces. This allows us to update the belief distribution according to our model of the opponent's play. This is why we require accurate opponent modelling to improve the performance of Fianchetto.

In the following sections, we look at existing agents for EFGs and IIEFGs for games like Poker and RBC. We also look at solution techniques for POMDPs, EFGs and IIEFGs.

15. Existing Work for Extensive Form Games

15.1 Fictitious Self-Play

(Heinrich, Lanctot, & Silver, 2015) introduces full-width Extensive-form Fictitious Play (XFP) and Fictitious Self-Play (FSP). These methods build upon the concept of Fictitious Play, which was first proposed by (Brown, 1951) as a technique to determine the Nash Equilibrium in Normal-Form Games (NFGs).

Fictitious Play is an iterative process in which players calculate their best responses to the average strategies of their opponents. This method is guaranteed to converge to the Nash Equilibrium over time. XFP is an extension of Fictitious Play that adapts the original concept for application to EFGs. As a full-width approach, XFP must examine all states within the game tree during each iteration, resulting in a linear computational complexity in the number of states, per iteration. Importantly, XFP maintains the same theoretical convergence guarantees as its predecessor, Fictitious Play. However, XFP has too high of a computational cost to be useful for most applications.

FSP attempts to mitigate these issues using Neural Networks and Machine Learning techniques. An agent gathers experience by repeatedly playing games against a version of itself, and stores these games in memory. The policy used by the agent is a mixture of its best response strategy and its average strategy. The agent uses Reinforcement Learning to update the approximate best response strategy to its opponent's moves. The agent then uses Supervised Learning on the experience of its own play to update its average strategy. Thus, FSP is a versatile technique to find the Nash Equilibrium for EFGs that can be used with numerous Reinforcement Learning algorithms.

16. Existing Work for POMDPs

16.1 Penumbra - Neural Networks for RBC

Penumbra (Clark, 2021b) is an RBC agent that won the 2020 NeurIPS RBC Leaderboard Challenge and finished third in the 2021 NeurIPS RBC Tournament. A retrained version of Penumbra named Châteaux finished fourth in the 2022 NeurIPS RBC Tournament. The authors of Penumbra introduce a novel technique known as Deep Synoptic Monte Carlo Planning (DSMCP). Here RBC is modelled as a POMDP, with the opponent's fixed policy assumed to be a part of the environment. Penumbra uses features for abstracting states in an Information Set in RBC. Abstraction is done using 104 hand-crafted features known as bitboards.

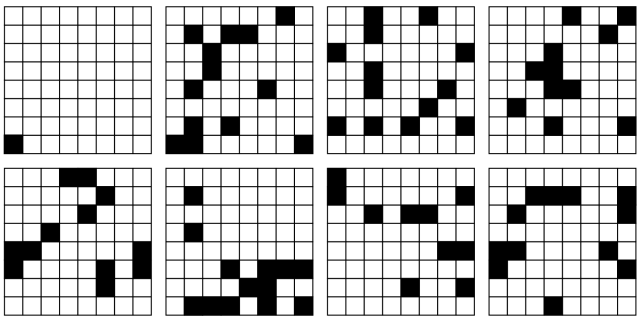


Figure 20: An example of 8 random bitboards for some information set.

Bitboards are 8×8 chessboards with a bit for each square, as depicted in Figure 20. Each bitboard is used to answer a question across all nodes in the current information set such as "What are the possible locations of opponent pawns?". These features are input to a residual neural network with multiple output heads for different opponents. This gives a different policy for each opponent. DSMCP mixes a bandit algorithm policy with the neural network policy to select a move. Thus, DSMCP seeks to find the best response against a known opponent, rather than finding a Nash Equilibrium strategy that works against all opponents.

16.2 Using Observations to Train a Policy Network for RBC

(Bertram, Fürnkranz, & Müller, 2022) presents JKU-CODA, an agent for RBC that attempts to approximate the POMDP as an MDP. JKU-CODA learns a policy Neural Network that directly maps observations to actions. The agent does not attempt to model the state it is in while playing. The agent is trained in two stages - first by training a Neural Network using Supervised Learning on the games of Penumbra, and second by optimising this network by self-play with Reinforcement Learning. A Reinforcement Learning algorithm known as Proximal Policy Optimization (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) is used here. Although JKU-CODA does not achieve a particularly high ELO (Wikipedia,) in RBC, this work shows that it is possible to play RBC without explicitly modelling the game state.

16.3 A solution to POMDPs using a Policy for the Fully-Observable Case

An agent that greedily chooses the best action for the most probable state will not do well on POMDPs where multiple states have a high probability of being the true state. Here the belief probability distribution is multimodal. (Sulyok & Karacs, 2022) presents an approach for solving such multimodal POMDPs, and gives the mathematical background for the same. This approach makes the important assumption that we have a near-optimal policy for the fully observable underlying MDP and that the full belief calculation at each stage is computationally feasible. The second assumption implies that the opponent's policy is either known, or that opponent plays the game with full observability.

The approach calculates a mixture policy π_{po} for the POMDP as a one-step approximation to the Q-value function for the fully-observable MDP with optimal policy π_o. The

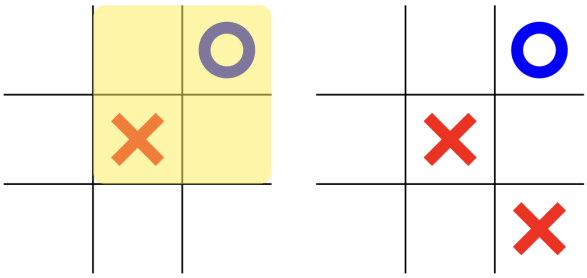


Figure 21: Reconnaissance Blind Tic Tac Toe: A smaller Imperfect Information Extensive Form Game than Reconnaissance Blind Chess. Players have to sense a 2×2 region in the Tic Tac Toe Board.

Q-value of the mixture policy is given as an expectation across the belief distribution of the optimal Q-value function. Given a belief distribution β, a state s and an action a:

$$\tilde{Q}^{\pi_{po}}(\beta, a) := \mathop{\mathbb{E}}_{s\sim\beta} Q^{\pi_o}(s, a)$$

The policy π_{po} is then given by:

$$\pi_{po}(a \mid \beta) := \begin{cases} 1 & \text{if } a = \arg\max_{a' \in a} \widetilde{Q}^{\pi_o}(\beta, a') \\ 0 & \text{otherwise} \end{cases}$$

This work demonstrates this approach for the game of Reconnaissance Blind Tic Tac Toe. This is an imperfect information version of Tic Tac Toe where players cannot see each other's moves and have to sense a 2×2 region of the board before each turn. An example is shown in Figure 21. For this game, the tree is not very large and the complete belief distribution can be computed at every turn, assuming that the opponent has full visibility of the position. The results of this game show that the mixture policy outperforms a greedy policy that assumes a unimodal POMDP belief distribution. In order to extend this work to larger games like RBC, an analysis must be done about belief approximation techniques.

17. Existing Work for Imperfect Information Extensive Form Games

17.1 Liberatus - Champion Poker Agent

Texas Holdem Poker, the most popular version of Poker, is a card game consisting of four bidding rounds. Before each round of bidding, players are given more information about their hand, starting from two private cards dealt to each player to five public cards revealed over the course of the game. Players must gauge the strength of their hands from the information available, building a belief of what state in the game tree they are likely to be in. Poker can be formulated as an IIEFG with a tree of depth four, where each node represents a possible distribution of cards to all players.

Libratus (Brown & Sandholm, 2017b) was the first agent to achieve superhuman performance in two-player Texas Holdem Poker. It is impressive to note that the methods used in Libratus assume no domain knowledge about Poker and can be extended to other IIEFGs. Following are the key ideas of Libratus responsible for its success.

Figure 22: Abstraction in Bet Sizes: A bet size of $100 and a bet size of $102 can be considered equivalent.

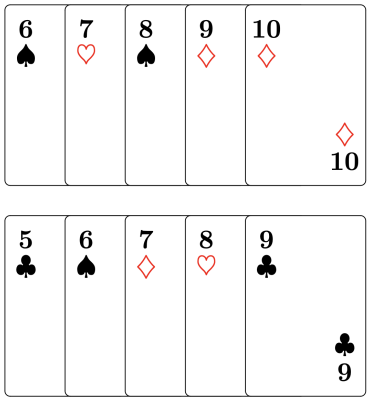


Figure 23: Abstraction in States: A 10-high straight and a 9-high straight can be considered equivalent.

17.1.1 Abstraction of States in Poker

In order to tackle the problem of a large game tree, nodes are grouped together in buckets based on certain features. These abstractions are round-specific and help reduce the size of the problem. Figure 22 shows discrete bet sizes, an example of an abstraction used in Poker. There is not much of a difference between a bet of $100 and $102 and hence they are considered equivalent. Another example is shown in Figure 23. Certain hands in Poker are roughly equivalent such as a 10-high straight versus a 9-high straight and are treated as such.

17.1.2 Concept of a Blueprint Strategy

A Blueprint Strategy describes an optimal action for each state in the entire abstracted game tree. This strategy is computed through self-play using a modified version of Monte Carlo Counterfactual Regret Minimization (MCCFR) (Lanctot, Waugh, Zinkevich, & Bowling, 2009). Libratus follows this Blueprint Strategy only for the first two rounds of the game. In the next two rounds, the Blueprint Strategy is only used to estimate the expected reward while refining the strategy using Nested Safe Subgame Solving (Brown & Sandholm, 2017a).

17.1.3 Nested Safe Subgame Solving

After two rounds of the game, the player is in a subtree of the original game tree where the root node is an information set. This is known as a subgame. The solution to this subgame must consider the possibility of the agent being in any of the nodes in the information set of the root node. For rounds three and four, new abstractions of the game are created and the Blueprint Strategy is refined. Now, using this refined strategy, the probabilities of the player being in the nodes in the information set of the root node are recalculated. Using these probabilities, a solution to this subgame is computed.

18. Challenges in adapting approaches for Fianchetto

The first challenge is that of game state abstraction. The techniques of using a Blueprint Strategy and refining it using Nested Safe Subgame Solving can only work in Poker due to abstractions that lower the size of the game tree. RBC is an even more vast game than Poker, thus it is imperative to design abstractions for chess for these methods to be of use. However, it is unclear how to manually design abstractions for a game as complex as chess. Penumbra approached this problem by using features and a neural network to abstract game states, but manually designing an appropriate and minimal set of features is a challenging task - one that is not easily generalizable. Furthermore, even if a powerful abstraction were designed, the algorithms used in Libratus may not scale, as RBC has a game tree of depth usually greater than fifty, compared to Poker's fixed depth of four.

The second challenge is of coming up with a strategy that can work well for unseen opponents. Penumbra does well against known opponents but it may be easily exploited by unseen opponents. It remains unknown if it is possible to design an agent with a strategy that does well against any opponent. Further reading is required to tackle these problems.

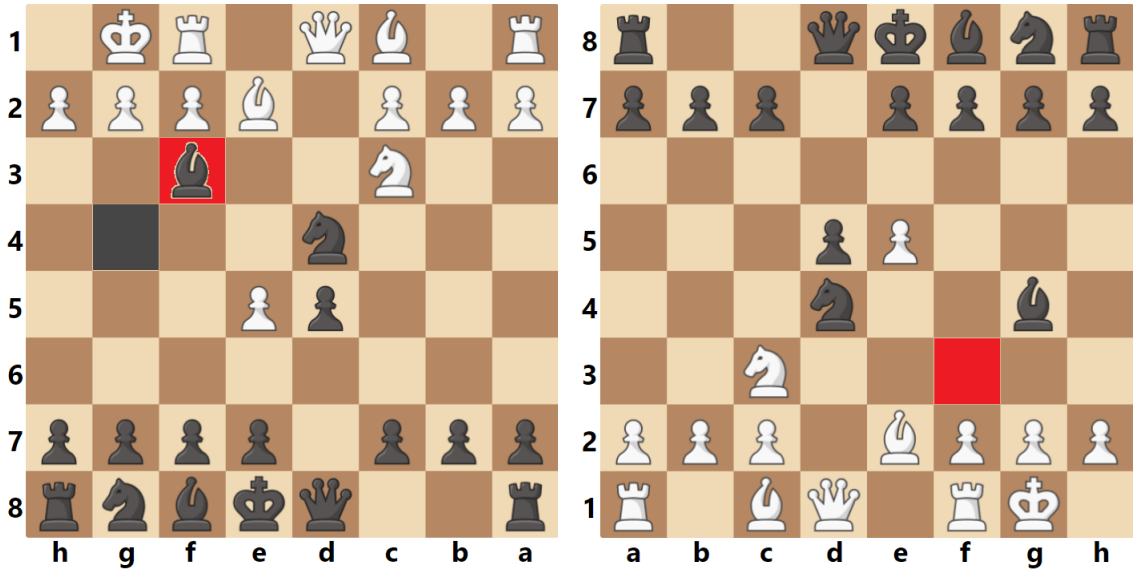
19. Conclusions and Future Work

We have surveyed solution techniques for Partially Observable Markov Decision Processes and Imperfect Information Extensive Form Games. We notice some common themes in most of these approaches. Firstly, many of these approaches use Neural Networks trained by self-play and/or supervised learning. Policy networks that map observations to actions such as in JKU-CODA could be useful as evaluation functions in search by performing rollouts. Networks that learn an opponent's policy such as in Penumbra can be used for opponent modelling. Secondly, we see that finding abstractions for states in the game tree is crucial in dealing with large games such as Chess. In the case of RBC, it is most likely that a feature-based abstraction technique must be used; however, work must be done towards finding a technique to learn these abstractions. Thirdly, we see the need to look into sampling-based belief estimation techniques in order to compute more accurate belief updates for large information sets in RBC. We will focus on implementing some of these ideas in the future.

Appendix A. Rules of RBC

RBC was designed with the intention to keep it as close to standard Chess as possible. The addition of uncertainty and active sensing makes the rules of RBC differ from the rules of Chess in the following ways.

- Opponent pieces are not visible directly to the player.
- Before each turn, the players sense a 3x3 grid on the board to obtain the true board state position for that region without providing any information to the opponent about this sense. As shown in Figure 1a, black sensed at f2 square to obtain the information of white's pieces in that region as illustrated in 1b.
- On capturing a piece, the player is informed only that a capture has occurred but not the piece which is captured. As shown in 24a, black is informed that a capture has been made at f3 but black is not informed which piece was captured at that square.
- If one of the player's pieces is captured, only the position of the captured piece is revealed and not the piece which made the capture. As shown in 24b, white is only informed that a capture has occurred at f3 but white is not informed whether that capture was made by black's knight or bishop.



(a) Black captures

(b) White's observation

Figure 24

- The notion of check and mate is removed as the player is not informed regarding any check.
- The game is won by capturing the opponent's king or when the opponent runs out of time. The transition in Figure 25 shows the winning move played by white to capture

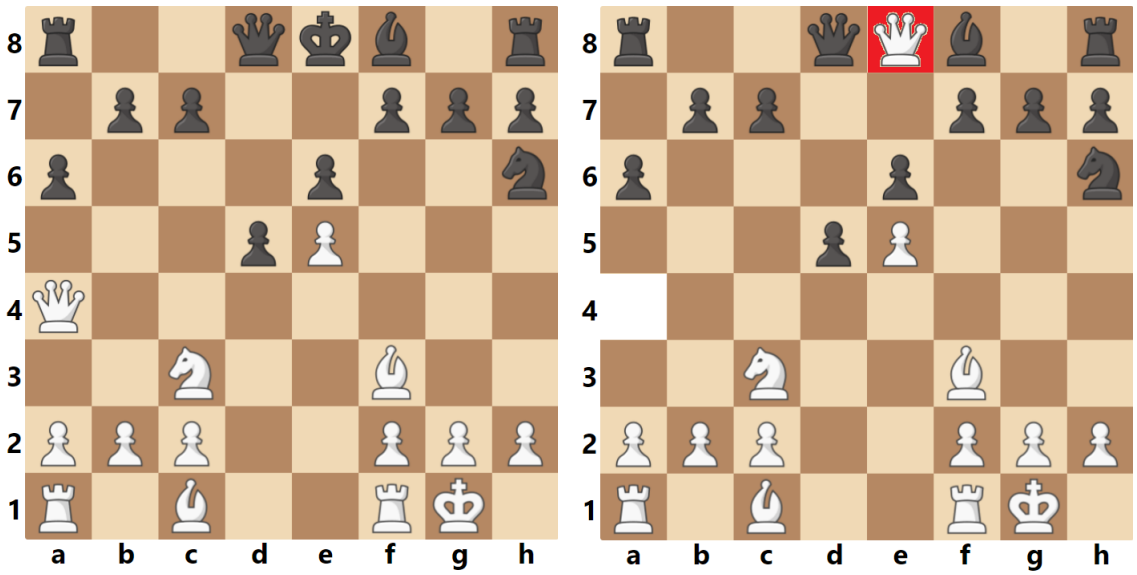


Figure 25

black's king. In the NeurIPS competition, each player had 15 minutes on the clock without increment to complete the game.

- All rules associated with stalemates or automatic draw conditions are eliminated. Hence, an RBC game never ends in a draw; it must end in favour of one player either by king capture or by timeout. In either of the cases, both the players are notified that the game has ended.

- A sliding move over a opponent's piece (using a queen, bishop or a rook) results in the capture of that piece and the sliding move is halted at the position of the capture. The transition in Figure 26 shows white's attempt to play the queen from d1 to d8, which results in capturing the black's knight at d4 which is blocking the attempted move.

- A sliding move over an opponent's piece using a King (from e1 to g1) results in no move as shown in 27a. The same using a pawn move (from d2 to d4), as shown in 27b results in one step forward movement of pawn.

- If a player attempts an illegal move such as a pawn attack (d5-e4) as shown in Figure 28a or a pawn forward move (d5-d4) as shown in Figure 28b, they are notified that the move could not be completed. Moves that place the player's own king in check are allowed (as also castling through check) as the notion of check itself is removed.

- RBC also adds a *pass* (or a null) move where the player moves nothing.

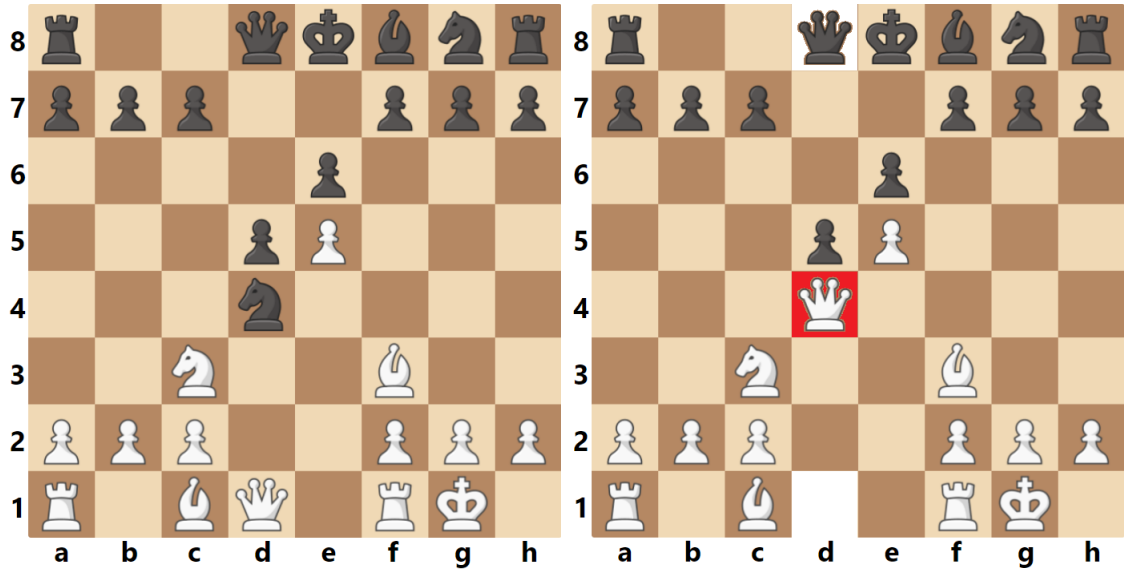
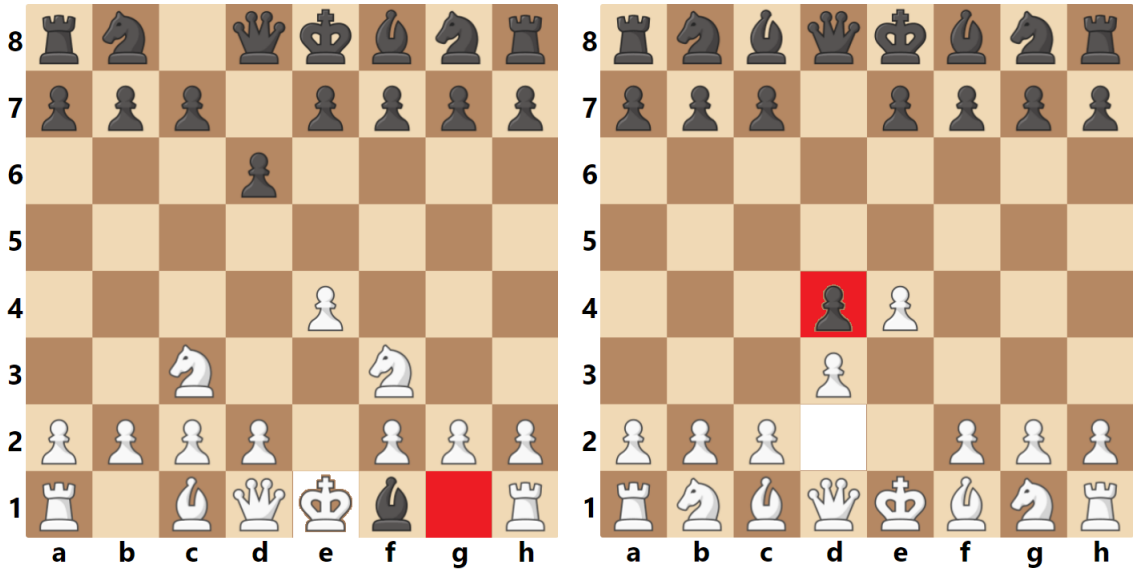


Figure 26



(a) Castle slide

(b) Pawn slide

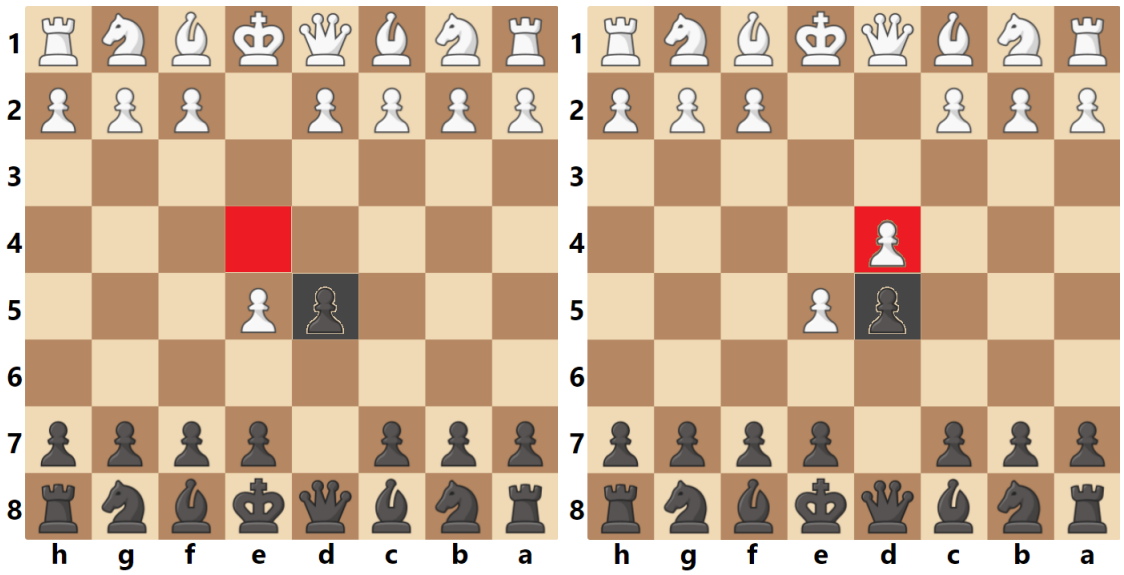
Figure 27

Appendix B. Chess versus RBC

Appendix C. Bayesian ELO Rating System

If player A has a rating of R_A and player B a rating of R_B, the exact formulae for the expected score of player A and B are

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}} \, , \, E_B = \frac{1}{1 + 10^{(R_A - R_B)/400}}$$



(a) Illegal capture (b) Illegal move

Figure 28

When a player's actual tournament scores exceed (or fall short) their expected scores, the Elo system (Coulom, 2008) takes this as evidence that player's rating is too low (or too high), and needs to be adjusted upward (or downward). Elo uses a simple linear adjustment proportional to the amount by which a player over-performed or under-performed their expected score. The maximum possible adjustment per game is called the K-factor. Now suppose player A scored S_A points, then the formula for updating the player's rating is

$$R_{A'} = R_A + K \cdot (S_A - E_A)$$

This rating update was performed for each bot after each game of the tournament. The K-factor used for all the bots throughout the tournament was $K = 40$. All the bots started the tournament with ELO Rating of 1200.

Appendix D. Related Work

Game Size Comparison

A widely adopted method by Shannon (1950) compares the game size based on the number of possible states that a player can encounter in a game. As shown in Table 10, the number of states in RBC is in the same order of No-Limit Poker and Go which is way larger than Chess or Limit Heads-Up Texas Hold'em Poker. The average number of possible states in the information set also serves as a good metric for comparing the complexity of games as it represents the difficulty in evaluation of a perceived state. Table 11 (Markowitz et al., 2018) shows the approximate values for several games. The average number of states in the information set for RBC (considering the different possible states based on opponent knowledge) exceeds that of Six-player-No-Limit Poker (Guo et al., 2020).

Game	Size
Texas Hold'em Poker	10^{13}
Chess	10^{43}
RBC	10^{139}
No-Limit Poker	10^{162}
Go (19×19)	10^{170}

Table 10: Approximate size of games

Game	Game States
Heads-Up No-Limit	1083
Chess	1
RBC	1.3×10^{68}
No-Limit Poker	6.4×10^{14}
Go (19×19)	1

Table 11: Approximate number of possible opponents states in an information set

Different Approaches for perfect and imperfect information games

There have been a lot of advancements in many perfect and imperfect information based games in the last few decades. Many perfect information games like Checkers, Connect-Four and Go are now solved (Schaeffer, Burch, Björnsson, Kishimoto, Müller, Lake, Lu, & Sutphen, 2007; Allis, 1988) or considered solved (Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, et al., 2016).

Achieving optimality for imperfect information games becomes far more difficult as the entire game tree (including all the unreachable states) also needs to be considered for approximating an optimal action. Hence, Monte Carlo tree search (MCTS) approaches (Browne et al., 2012), which were an essential part for algorithm of AlphaZero (Silver et al., 2018), cannot be directly applied to imperfect information games. Variants like Information set MCTS (ISMCTS) (Cowling et al., 2012) is developed to work for imperfect information games but it does not provide any optimality guarantees. The large game size of RBC practically rules out the state of the art optimal solution approximation approaches like counterfactual regret minimization (CFR) (Zinkevich et al., 2007). Unlike for games like poker (Brown & Sandholm, 2018b), it is very difficult to come up with abstractions that reduce the size of the original game significantly.

Several AI approaches have been developed for chess in recent years, for example —Google DeepMinds's AlphaZero algorithm (Silver et al., 2017) and the Stockfish engine (Romstad et al., 2021). But since they do not account for uncertainty, they can't be applied directly to RBC without modification. Neural fictitious self-play (NFSP) is an approach which can be applied to two-player zero-sum games with imperfect information (Heinrich & Silver, 2016) and it avoids handcrafted game abstractions.

Libratus and DeepStack (Brown & Sandholm, 2018b; Moravčík et al., 2017)) leverage CFR and game-abstraction techniques to find Nash equilibrium strategies for sub-games of reduced complexity (Brown & Sandholm, 2018b). This "local regularity" is not shared by

RBC (since a piece at a position or at an immediate adjacent position has very different implications) and hence the abstraction techniques are of no use.

Public vs Private Information

The well known poker systems (Brown & Sandholm, 2018b; Moravčík et al., 2017) used the idea of reduction in game size by decomposing imperfect information games into subgames (Burch et al., 2014; Brown & Sandholm, 2018b; Šustr et al., 2019). These techniques operate on the public information of the game tree. Such a useful decomposition is not possible for a game like RBC as it involves very little public information at any point. The opponent is not provided with any information of the sense move, and whatever information that is gained by the player through sense is completely private.

The AI agent for Poker ReBeL (Brown et al., 2020) converts imperfect information games into continuous state space perfect information games with public belief states as nodes. This approach though powerful, is pointless for games like RBC which have substantial hidden or private information.

The major difference between RBC and most of the other common imperfect information games like scrabble (Sheppard, 2002; Richards & Amir, 2007), poker (Moravčík et al., 2017; Brown & Sandholm, 2018a), Bridge (Ginsberg, 1999, 2001), Dou Dizhu (Whitehouse et al., 2011), and Battleship (Silva & Vinhas, 2007; Clementis, 2013) is the absence of public information. In each of these games, the uncertainty reduction is caused through obtaining some extra public information. On the other side, in RBC, the information gained through sensing move is completely private to the player. Almost nothing except the starting position is common knowledge for the players.

Games similar to RBC

The blind variant of chess Kriegspiel (Wetherell et al., 1972; Favini, 2010; Ciancarini & Favini, 2007, 2010; Russell & Wolfe, 2005), is the most similar game to RBC. In Kriegspiel, the information regarding opponent's pieces location is gained through umpire's feedback. Similarly, the Go equivalent of Kriegspiel for chess is called Phantom Go (Cazenave, 2005; Wang et al., 2017). Dark chess is another blind chess variant which omits explicit sensing thereby resulting in more uncertainty. Banqi is another interesting game of perfect information with a chance element where neither player can know anything more than the other player at any point. The ability to sense portions of the board in RBC gives players more ability to manage uncertainty, which may make the game more realistic for many scenarios.

Are there other games which have perfect and imperfect information game counterpart

- simulation soccer
- Go vs Phantom Go

Appendix E. Details of Other RBC Agents

The RBC competition has been running from three consecutive years, as part of NeurIPS competition track in 2019 and 2021 and as an online competition in 2020. In these three years, many agents have participated in the competition. Here we describe the approach

of most successful agents from all the previous edition of the competition and additional agents against which we benchmarked our agent.

`StrangeFish`: was the winner of the NeurIPS 2019 competition on RBC (Gardner et al., 2020). The source code of the agent is publicly available and we used it as the baseline for our agent. A detailed description regarding the sense strategy and move strategy of this baseline is provided in the section 2.

`penumbra`: won the 2020 online competition on RBC (Clark, 2021a). Like `Fianchetto`, penumbra also maintains all possible board states. It uses Gibbs sampling for opponent state sampling and plays according to finite length playouts starting at samples of belief state constructed using an unweighted particle filter.

`LaSalle`: bot and its successor, the La-Q bot, maintain a separate probability distribution for each piece, reflecting the agent's belief about where that piece might be located on the board. LaSalle and La-Q achieved the second rank in the online competition in 2019 and 2020 respectively (Highley et al., 2020; Blowitski & Highley, 2021). Both these agents did not participate in the NeurIPS 2021 competition on RBC.

`Oracle`: same as in `Fianchetto`, Oracle tracks all possible opponent board states. The sensing strategy tries to minimize the number of possible board states in expectation. It assumes each board to be equiprobable with some minor heuristics regarding the possibility of a check to our king. The moving strategy greedily picks the move that is recommended by the Stockfish chess engine on most of the boards. The bot is developed by the organizers at the Johns Hopkins University Applied Physics Laboratory (Gardner et al., 2020).

`random`: choose move and sense actions uniformly at random.

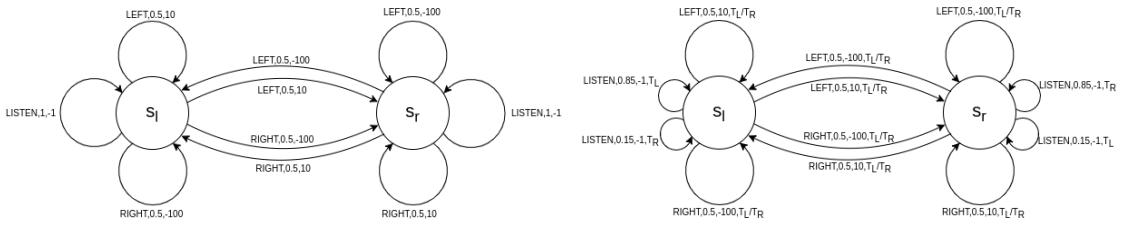
`trout`: uses the last observation of each square to maintain a singular board-state and uses Stockfish chess engine for move action. For positions with a past capture or a possibility of the capture, it senses over the capture square, else, it randomly selects the sensing region which does not contain any of its own pieces.

`attacker`: plays a random opening from a fixed number of hard-coded attacking openings and starts playing randomly if it is not able to continue playing the opening it chose (possibly due to piece capture or by playing all the moves that were hard-coded).

Trout, Attacker and Random are developed by the organizers at the Johns Hopkins University Applied Physics Laboratory and are available on the reconchess python library (Gardner et al., 2020).

Appendix F. Tiger Problem

One of the classic problems in the POMDP literature is the tiger problem (Kaelbling, Littman, & Cassandra, 1998). The MDP of the tiger problem is shown in Figure 29a. The



(a) MDP (b) POMDP

Figure 29: Tiger problem

MDP contains two states - s_l denoting that a tiger is behind the left door and s_r denoting that the tiger is behind the right door. The agent, at every turn, has to take one of the three actions - LEFT, RIGHT, or LISTEN which signify that the agent opens the left door, the right door or none of the door. Opening the door containing the tiger gives a large negative reward -100 and opening the other door gives a positive reward of +10. The LISTEN action gives a reward of -1. Opening of a door randomly places the tiger behind one of the doors and the next turn starts. However, the LISTEN action keeps the agent in the same state, i.e., the tiger stays behind the same door. The POMDP of the problem provides observation T_L with probability 0.85 and T_R with probability 0.15 when the state is s_l and the action is LISTEN, and conversely for the state s_r. On the other hand, the LEFT and RIGHT action both provide any of the observation T_L or T_R with uniform probability in any of the state.

The optimal policy for the MDP for any horizon length is trivial, since at every step the agent knows the state which tells the door behind which the tiger is, the agent can choose the opposite action - LEFT if the state is s_r and RIGHT if the state is s_l. The optimal value for any horizon $T > 0$ is then $100 * T$. Computing the optimal policy for the POMDP is however a little tricky. The problem is to compute the optimal policy given the belief b of the agent that the tiger is behind the left door. For horizon $T = 1$, if the agent believes with a high probability that the tiger is behind left door ($b > 0.9$), then the optimal action is RIGHT; if it believes with high probability that the tiger is behind the right door ($b < 0.1$), then the optimal action is LEFT. If it is uncertain about where the tiger is ($0.1 < b < 0.9$), then the optimal action is LISTEN. For $T = 2$, the first optimal action is to always LISTEN, no matter what the belief of the agent is and then on the next step take the optimal action according to the policy of $T = 1$ with the updated belief (after the observation from the LISTEN action). As explained by Kaelbling et al. (1998), this is because even if the agent opens a door on the first step, it will have to listen on the second step since the updated belief will be $b = 0.5$ after the first step (since the tiger is randomly behind any of the doors). Therefore, it is better to LISTEN on the first step and take a more informed decision on the next step. The complete policy for both the time steps is given in Figure 30 (Kaelbling et al., 1998).

Now let's compute the policy for the POMDP assuming that we only have the optimal policy for the MDP. In this case, for a given belief $(b, 1 - b)$, we can approximate the Q value for the belief by using the optimal Q_{MDP} value of the MDP:

$$Q_{POMDP}(b, a, t) = b * Q_{MDP}(s_l, a) + (1 - b) * Q_{MDP}(s_r, a)$$

Is there horizon in defining Q here, for MDP and POMDP? Can we show that for alternative choice of Q (inexact), we can do better by this aggregation rule?

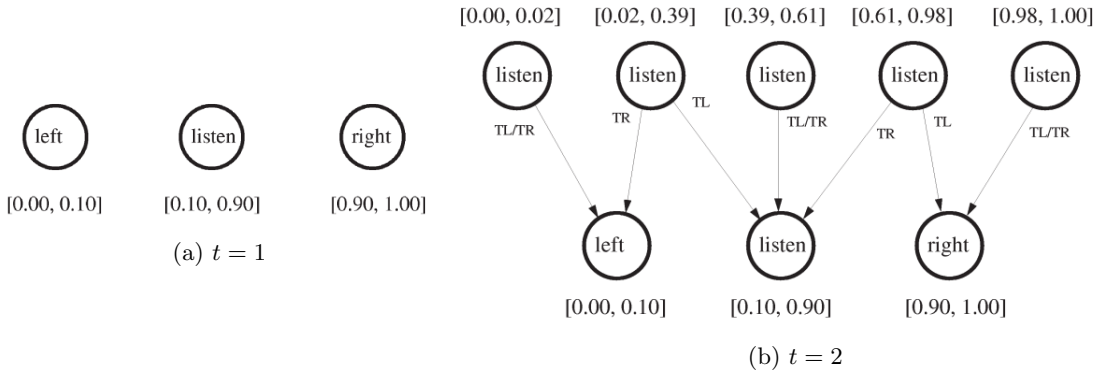
The above equation gives the policy in Figure 30a for all $T \geq 1$. This happens because the equation assumes that after one time step, the agent "knows" the exact state and executes the optimal policy for the MDP. This assumption is wrong for all $T > 1$ and therefore, the above approach gives a suboptimal policy for all $T > 1$. This can be seen by the example of $T = 2$ where the optimal policy is given in Figure 30b.

Since we know that plainly approximating the POMDP policy from the MDP policy is suboptimal, we can add our own rules to get a better POMDP policy. Consider the rule that the "LISTEN" action is usually better than other actions when $T > 1$. We incorporate this rule by adding an incentive given by a constant C to the Q-value as:

$$Q'_{POMDP}(b, a, t) = \begin{cases} Q_{POMDP}(b, a, t) + C, & \text{if } a = \text{LISTEN } \& \ t > 1 \\ Q_{POMDP}(b, a, t), & \text{otherwise} \end{cases}$$

We can see that if given a high enough incentive C, Q'_{POMDP} can give the exact optimal policy for horizon $T = 2$ by always choosing to "LISTEN" at the first time step and then following the optimal policy at the last time step.

This illustration shows that solving a POMDP problem by using the optimal solution of the corresponding MDP leads to sub-optimal policy. However, we can get better policies by adding rules that incentivize certain actions in certain situations.



(a) $t = 1$

(b) $t = 2$

Figure 30: Optimal policy tree for the POMDP of tiger problem

Appendix G. Rebuttal

Review 2, Factual error 1:

The reviewer comments: "It is not clear whether the proposed enhancements do contribute to the results."

Kindly note that Table 1 is precisely meant to demonstrate the incremental contribution of each proposed enhancement, measured by the win-loss ratio against (1) the StrangeFish (2019) baseline, as well as (2) opponents that were available on the RBC server after the competition ended. The last row of the table (for V4) shows the overall score when all the proposed enhancements are employed. Also note the specific statistics that we have

provided in individual cases. The point of batched board evaluations (in V1) is to increase the throughput of search, which is affirmed by data in Table 2. The purpose of opponent modeling and Bayesian updating in V2 is to predict the hidden state more accurately; Figure 2 clearly shows improved prediction accuracy over Strangefish.

Review 2, Question 1:

Although current techniques achieve practically optimal performance in perfect information games such as Chess and Go, the challenge of negotiating imperfect information in games is still a stumbling block for AI. For example, while one might reasonably expect Monte Carlo Tree Search to benefit decision time planning, Ciancarini and Favini [2010] report that just a 1-step rollout outperforms full-trajectory rollouts in Kriegspiel. The novel idea of neural "synopsis" [Clark, 2021] did result in a winning RBC agent in 2020, but this same agent lost to OurAgent 20-40 in 2021. The well-publicised Hidden Information Games Competition (2021) did not receive even a single entry even after being open for over half a year. In summary, the AI research community is yet to discover definitive, practicable approaches in the domain of imperfect information games. We expect our contributions RBC (an especially challenging instance due to having virtually no public information) to generalise within this domain.

- The high-level lesson from our solution is the need to optimise and integrate modules for search, opponent modeling, and uncertainty management. It appears unlikely that any single technical idea will become a silver bullet.
- GPUs are routinely used for parallel computing in areas such as deep learning. We show that GPUs can offer the same convenience to on-line decision making in games.
- Many imperfect information games $G_{\text{imperfect}}$ have well-understood perfect information counterparts G_{perfect} (like RBC has Chess). Naturally, one might want to transfer learned knowledge from G_{perfect} to $G_{\text{imperfect}}$. Our experience (see Table 3) warns against the peril of overfitting, demonstrating that a *poorer* evaluation function for G_{perfect} might actually be better for $G_{\text{imperfect}}$.
- The idea of a mathematically-grounded belief update (in V2) is applicable to literally every task with partial observability. Rather than abandon it altogether due to issues of complexity, we use it effectively—in its canonical form in the early stages of the game, and by regulating the information set size (by mixing a uniform distribution with our opponent model) when it becomes too large. This approach could be used in general for Bayesian updating in tasks with large state spaces.
- Although the concrete rules we incorporate in V3 are specific to RBC, the general principle is that partial observability opens up many opportunities in game play to exploit strategically. That bad moves in Chess might be good ones for RBC (and vice versa) could very well apply to other $(G_{\text{imperfect}}, G_{\text{perfect}})$ pairs.

Review 2, Question 2:
Please see our comments under "Review 2, Factual error 1" and our response under "Review 4, Question 2".

Review 3, Question 1:

Please see our response to "Review 2, Question 1".

Review 4, Question 1:

Let I be the information set (that is, the set of boards consistent with history) after the agent's sense move. Both StrangeFish and OurAgent require a weight $w(s')$ associated with each state $s' \in I$ in order to select an action. StrangeFish sets $w(s')$ based on the StockFish evaluation of s'. On the other hand, OurAgent sets $w(s')$ to be the actual belief (a probability) associated with s'. This belief is obtained by a nested computation involving the opponent's action a from each state s in the preceding belief state b. Hence, our action selection step performs a non-trivial computation for each $s \to a \to s'$ transition, whereas StrangeFish does so only for each s'. Since each player's "think time" is restricted to 900 seconds per game, our extra layer of computation is feasible only because Lc0 can process multiple boards in parallel, and specify probabilities for all actions simultaneously.

Review 4, Question 2:

Indeed there are dependencies between our versions. Without the speedup provided by V1, it would not be possible to perform the belief update in V2 in the available time. Moreover, it is because we use a proper belief state that we have to regulate its size in V4. The RBC-specific changes in V3, alone, are independent of the other versions. The most important contributor to OurAgent's success is indeed the Bayesian update in V2 (in turn facilitated by the speedup of V1).

References

Allis, L. V. (1988). A knowledge-based approach of connect-four. *J. Int. Comput. Games Assoc.*, *11*, 165.

Åström, K. (1965). Optimal control of markov processes with incomplete state information i. *Journal of Mathematical Analysis and Applications*, *10*, 174–205.

Bertram, T., Fürnkranz, J., & Müller, M. (2022). Supervised and reinforcement learning from observations in reconnaissance blind chess..

Blowitski, K., & Highley, T. (2021). Checkpoint variations for deep q-learning in reconnaissance blind chess. *Journal of Computing Sciences in Colleges*, *37*(3), 81–88.

Brown, G. W. (1951). Iterative solution of games by fictitious play. In Koopmans, T. C. (Ed.), *Activity Analysis of Production and Allocation*. Wiley, New York.

Brown, N., Bakhtin, A., Lerer, A., & Gong, Q. (2020). Combining deep reinforcement learning and search for imperfect-information games. *CoRR*, *abs/2007.13544*.

Brown, N., & Sandholm, T. (2017a). Safe and nested subgame solving for imperfect-information games. *CoRR*, *abs/1705.02955*.

Brown, N., & Sandholm, T. (2017b). Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, *359*, eaao1733.

Brown, N., & Sandholm, T. (2018a). Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, *359*(6374), 418–424.

Brown, N., & Sandholm, T. (2018b). Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, *359*(6374), 418–424.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1–43.

Burch, N., Johanson, M., & Bowling, M. (2014). Solving imperfect information games using decomposition. In *Twenty-eighth AAAI conference on artificial intelligence*.

Campbell, M., Hoane Jr., A. J., & Hsu, F.-h. (2002). Deep Blue. *Artificial Intelligence*, *134*(1–2), 57–83.

Cazenave, T. (2005). A phantom-go program. In *Advances in Computer Games*, pp. 120–125. Springer.

Ciancarini, P., & Favini, G. P. (2007). Representing kriegspiel states with metapositions.. In *IJCAI*, pp. 2450–2455.

Ciancarini, P., & Favini, G. P. (2010). Monte carlo tree search in Kriegspiel. *Artificial Intelligence*, *174*(11), 670–684.

Clark, G. (2021a). Deep synoptic monte-carlo planning in reconnaissance blind chess. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., & Vaughan, J. W. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 34, pp. 4106–4119. Curran Associates, Inc.

Clark, G. (2021b). Deep synoptic monte-carlo planning in reconnaissance blind chess. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., & Vaughan, J. W. (Eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 4106–4119.

Clementis, L. (2013). Supervised and reinforcement learning in neural network based approach to the battleship game strategy. In *Nostradamus 2013: Prediction, Modeling and Analysis of Complex Systems*, pp. 191–200. Springer.

Coulom, R. (2008). Whole-history rating: A Bayesian rating system for players of time-varying strength. In *Computers and Games*, pp. 113–124. Springer.

Cowling, P. I., Powley, E. J., & Whitehouse, D. (2012). Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(2), 120–143.

Favini, G.-P. (2010). The dark side of the board: advances in chess kriegspiel..

Gardner, R. W., Lowman, C., Richardson, C., Llorens, A. J., Markowitz, J., Drenkow, N., Newman, A., Clark, G., Perrotta, G., Perrotta, R., Highley, T., Shcherbina, V., Bernadoni, W., Jordan, M., & Asenov, A. (2020). The first international competition in machine Reconnaissance Blind Chess. In *NeurIPS 2019 Competition and Demonstration Track*, pp. 121–130. PMLR.

Ginsberg, M. L. (1999). Gib: Steps toward an expert-level bridge-playing program. In *IJCAI*, pp. 584–593. Citeseer.

Ginsberg, M. L. (2001). Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, *14*, 303–358.

Guo, Z., Wang, X., Qi, S., Qian, T., & Zhang, J. (2020). Heuristic sensing: An uncertainty exploration method in imperfect information games. *Complexity*, *2020*.

Heinrich, J., Lanctot, M., & Silver, D. (2015). Fictitious self-play in extensive-form games. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 805–813. JMLR.org.

Heinrich, J., & Silver, D. (2016). Deep reinforcement learning from self-play in imperfect-information games. *ArXiv*, *abs/1603.01121*.

Highley, T., Funk, B., & Okin, L. (2020). Dealing with uncertainty: a piecewisegrid agent for reconnaissance blind chess. *Journal of Computing Sciences in Colleges*, *35*(8), 156–165.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1), 99–134.

Lanctot, M., Waugh, K., Zinkevich, M., & Bowling, M. (2009). Monte carlo sampling for regret minimization in extensive games. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., & Culotta, A. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 22. Curran Associates, Inc.

Markowitz, J., Gardner, R. W., & Llorens, A. J. (2018). On the complexity of Reconnaissance Blind Chess. *CoRR*, *abs/1811.03119*.

Maschler, M., Solan, E., & Zamir, S. (2013a). *Equilibrium refinements*, p. 251–299. Cambridge University Press.

Maschler, M., Solan, E., & Zamir, S. (2013b). *Extensive-form games*, p. 39–74. Cambridge University Press.

Maschler, M., Solan, E., & Zamir, S. (2013c). *Game Theory*. Cambridge University Press.

Maschler, M., Solan, E., & Zamir, S. (2013d). *Strategic-form games*, p. 75–143. Cambridge University Press.

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

Pascutto, G.-C., & Linscott, G. (2021). Leela Chess Zero.. URL: https://lczero.org/.

Pascutto, Gian-Carlo and Linscott, Gary. Leela chess zero. http://lczero.org/.

Perrotta, G., & Perrotta, R. (2019). StrangeFish..

Richards, M., & Amir, E. (2007). Opponent modeling in scrabble.. In *IJCAI*, pp. 1482–1487.

Romstad, T., Costalba, M., & Kiiski, J. (2021). Stockfish - open source chess engine.. URL: https://stockfishchess.org/.

Russell, S., & Wolfe, J. (2005). Efficient belief-state and-or search, with application to kriegspiel. In *IJCAI*, Vol. 19, p. 278.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., & Sutphen, S. (2007). Checkers is solved. *Science*, *317*(5844), 1518–1522.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms..

Shannon, C. E. (1950). Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *41*(314), 256–275.

Sheppard, B. (2002). World-championship-caliber Scrabble. *Artificial Intelligence*, *134*(1–2), 241–275.

Silva, D. C., & Vinhas, V. (2007). An interactive augmented reality battleship game implementation.. pp. 213–219. Citeseer.

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., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354–359.

Sulyok, A. A., & Karacs, K. (2022). Towards using fully observable policies for pomdps..

Turing, A. M. (1953). Chess. In Bowden, B. V. (Ed.), *Faster than thought: A symposium on digital computing machines*, pp. 288–297. Sir Isaac Pitman & Sons Ltd.

Šustr, M., Kovařík, V., & Lisý, V. (2019). Monte carlo continual resolving for online strategy computation in imperfect information games. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, p. 224–232, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Wang, J., Zhu, T., Li, H., Hsueh, C.-H., & Wu, I.-C. (2017). Belief-state monte carlo tree search for phantom go. *IEEE Transactions on Games*, *10*(2), 139–154.

Wetherell, C. S., Buckholtz, T. J., & Booth, K. S. (1972). A director for Kriegspiel, a variant of Chess. *The Computer Journal*, *15*(1), 66–70.

Whitehouse, D., Powley, E. J., & Cowling, P. I. (2011). Determinization and information set monte carlo tree search for the card game dou di zhu. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pp. 87–94. IEEE.

Wikipedia. Elo rating system — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Elo\%20rating\%20system&oldid=1090925763.

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2007). Regret minimization in games with incomplete information. *Advances in neural information processing systems*, *20*.