Fianchetto: Speed, Belief, Guile, Caution to Win at Reconnaissance Blind Chess

Mohammad Taufeeque TAUFEEQUE@CSE.IITB.AC.IN
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Nitish Tongia NITISHTONGIA@EE.IITB.AC.IN
Department of Electrical Engineering
Indian Institute of Technology Bombay

Shivaram Kalyanakrishnan SHIVARAM@CSE.IITB.AC.IN
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

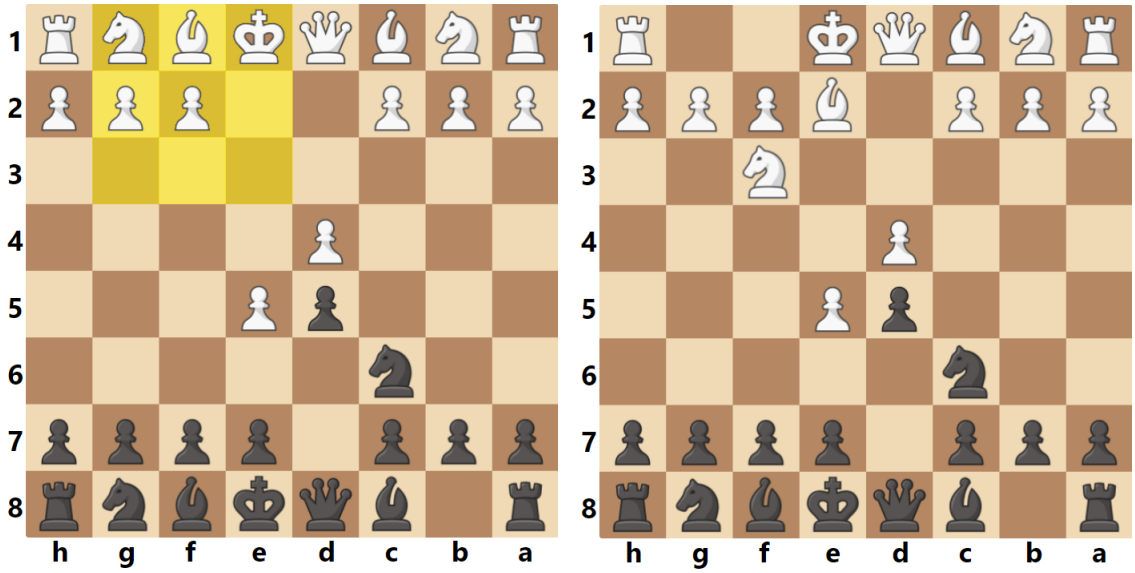
Abstract

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 also 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, Lisý, Morrill, Bard, Davis, Waugh, Johanson, & Bowling, 2017; Brown & Sandholm, 2018a) have been test beds for research on imperfect information games.

Reconnaissance Blind Chess (RBC) (Gardner, Lowman, Richardson, Llorens, Markowitz, Drenkow, Newman, Clark, Perrotta, Perrotta, Highley, Shcherbina, Bernadoni, Jordan, &



(a) Sense region (b) Sense result

Figure 1: An RBC position before the sense move on the f2 square is played, highlighting the 3x3 region that will be updated (left) along with the updated board after the sense move (right).

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, 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. 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, 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 compu-

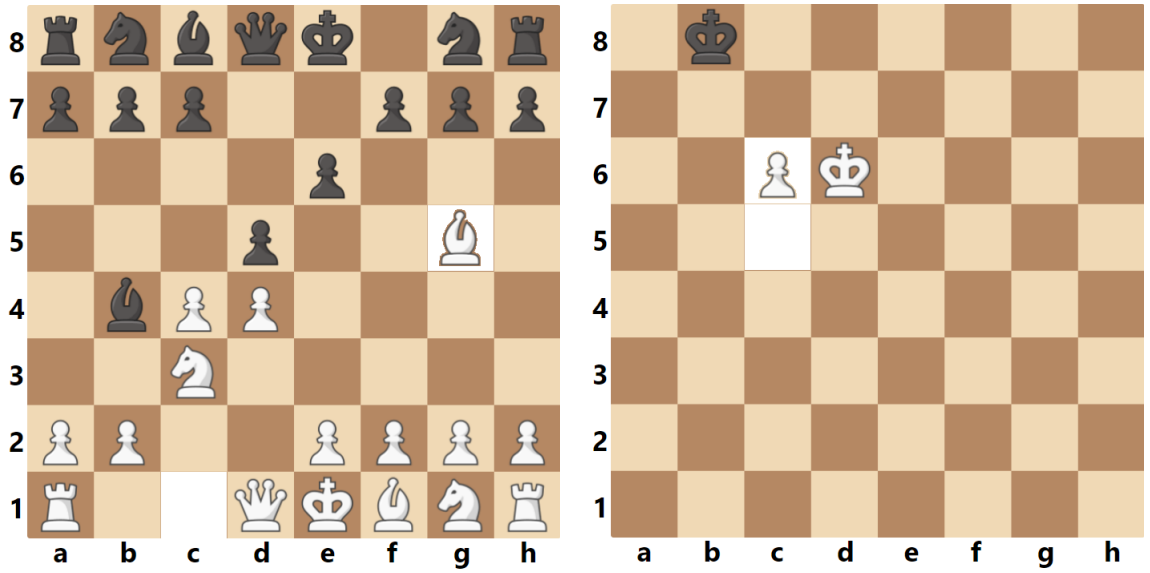
tation of equilibrium strategies (Zinkevich, Johanson, Bowling, & Piccione, 2007; Brown & Sandholm, 2018b). Moreover, it appears feasible in Poker to generalise across *belief states* for learning behaviour (Brown, Bakhtin, Lerer, & Gong, 2020), thereby removing the non-Markovian influence of hidden 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 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.

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. Markowitz et al. (2018) estimate that the number of distinct states (in perfect information games) or information sets (in imperfect information games) 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, that includes opponent's possible knowledge. Markowitz et al. (2018) estimate the number of possible opponent states for a given information set at 10^{68}—much larger than 10^{14} for 6-player poker or 1 for perfect information games like Chess or Go. This additional layer of complexity provides additional hurdles to compute ineffi-



(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.

cient algorithms like MCTS or their imperfect information variants like ISMCTS (Cowling, Powley, & Whitehouse, 2012) to find optimal policies for RBC.

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.

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).

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 4, lend strong support to the overall strength of `Fianchetto`. However, we must 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 5.

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. Randomization 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.

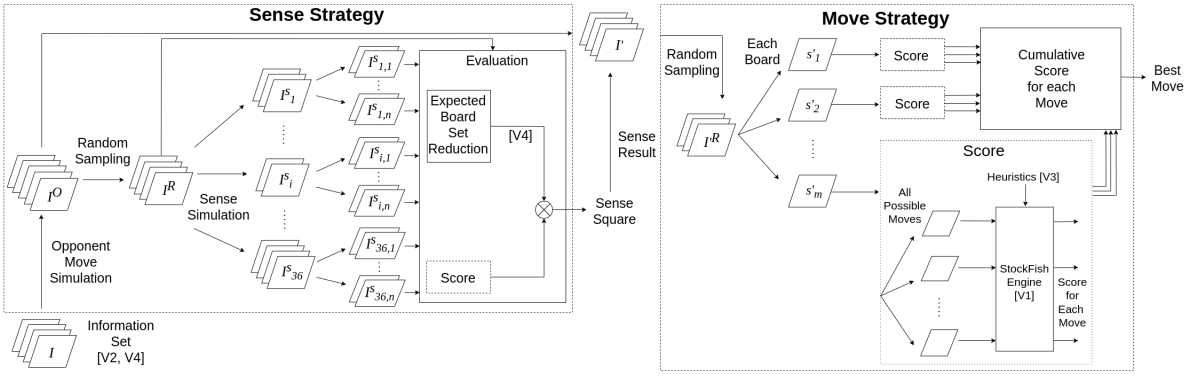


Figure 3: Control flow in `StrangeFish` (Perrotta & Perrotta, 2019). On each turn, the agent begins with the information set I from the previous turn, which the sense strategy maps to a sense square (equivalently, the reconnaissance move). Sensing prunes the information set to $I' \subseteq I$. Based 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.

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 organizers 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 minimize 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, 2021), 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 specialized 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	StrangeFish2	Oracle	trout	attacker	random	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 1: 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 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 1. 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 2, the time for a single call to Lc0 well exceeds that for a call to Stockfish. However, a more favorable 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 favor 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 parallelize 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 2 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 3 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 2: 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 3: 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 1, 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.

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.

The updated belief over the information set at every turn is used to determine the next move to play as:

$$\mathbb{P}\{a|b\} = \sum_{a} \mathbb{P}\{a|s\} \cdot \mathbb{P}\{s|b\}$$

Here $\mathbb{P}\{a|s\}$ is modeled using the Lc0 engine which provides near optimal policy for the Chess state s. This strategy of combining the belief over the information set with the optimal policy for the equivalent MDP (Chess) although works well in practice, cannot give an optimal policy for the POMDP (RBC) as it suffers from assuming that both the players have perfect information in the future by using the optimal MDP policy. We illustrate the suboptimality of this strategy through a simple demonstration in Appendix C.

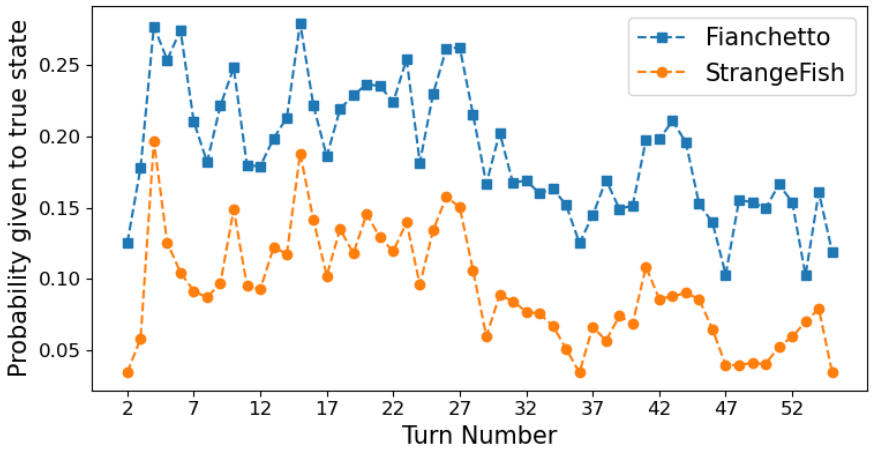


Figure 4: Probability associated with true state by `Fianchetto` (V2) and `StrangeFish`, averaged over hundreds of games played by V2 on the RBC server from 16–24 December 2021.

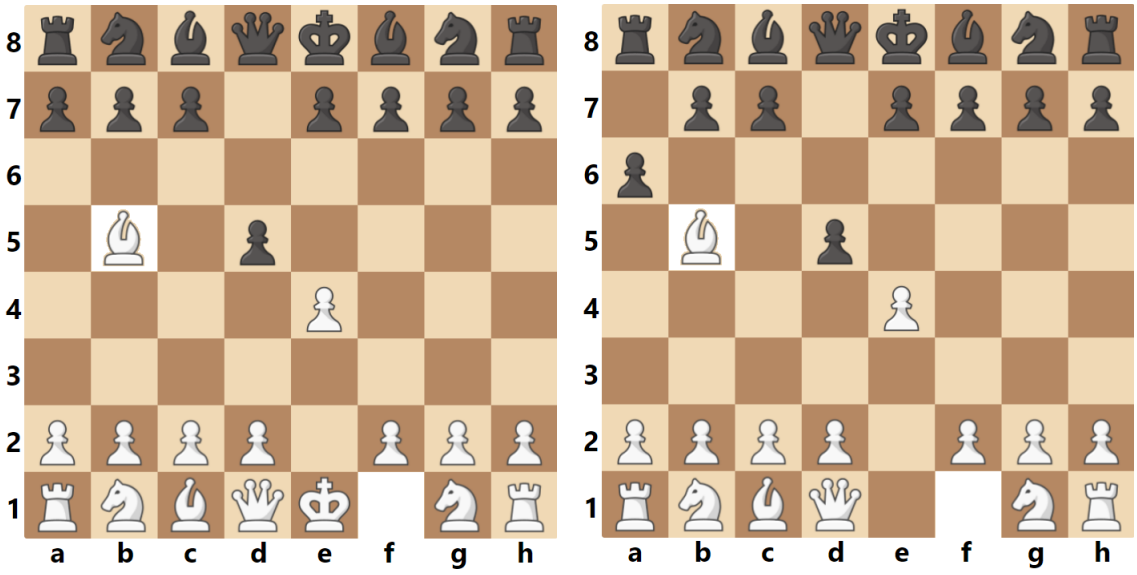
Of course, the effectiveness of our principled belief update depends on the accuracy of Lc0 as the opponent model. We do find empirical validation. Figure 4 compares the probabilities given to the true board state by `Fianchetto` and `StrangeFish` as the game progresses; observe that `Fianchetto` is consistently more accurate. Table 1 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.

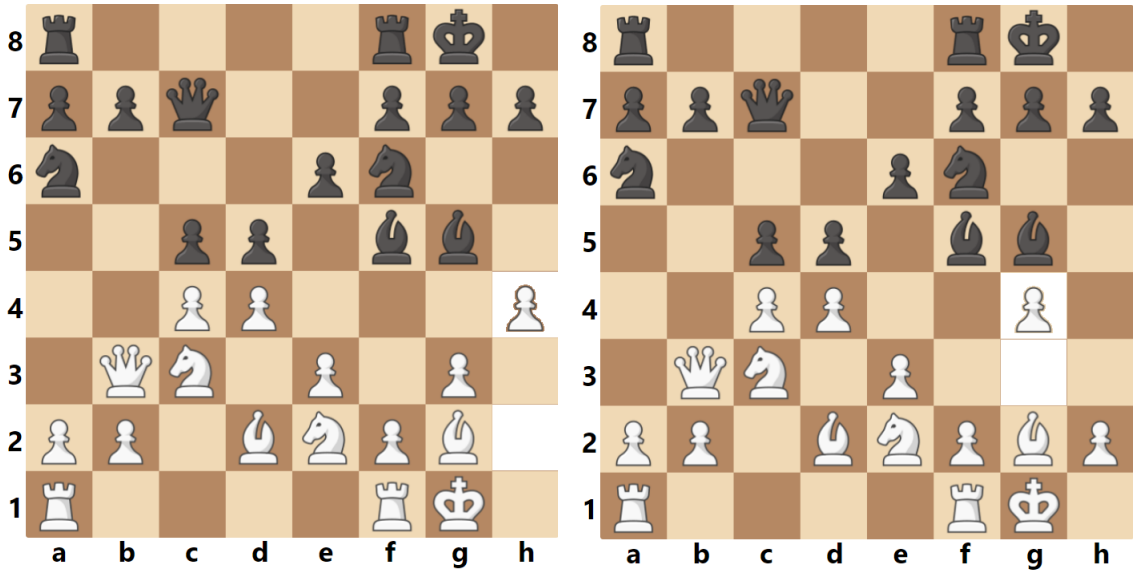


(a) Low risk (b) High risk

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 incentivize 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 incentivize moves to block the check from afar by moving



(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 that gets lesser incentive

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.

We observe from Table 1 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 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 an 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

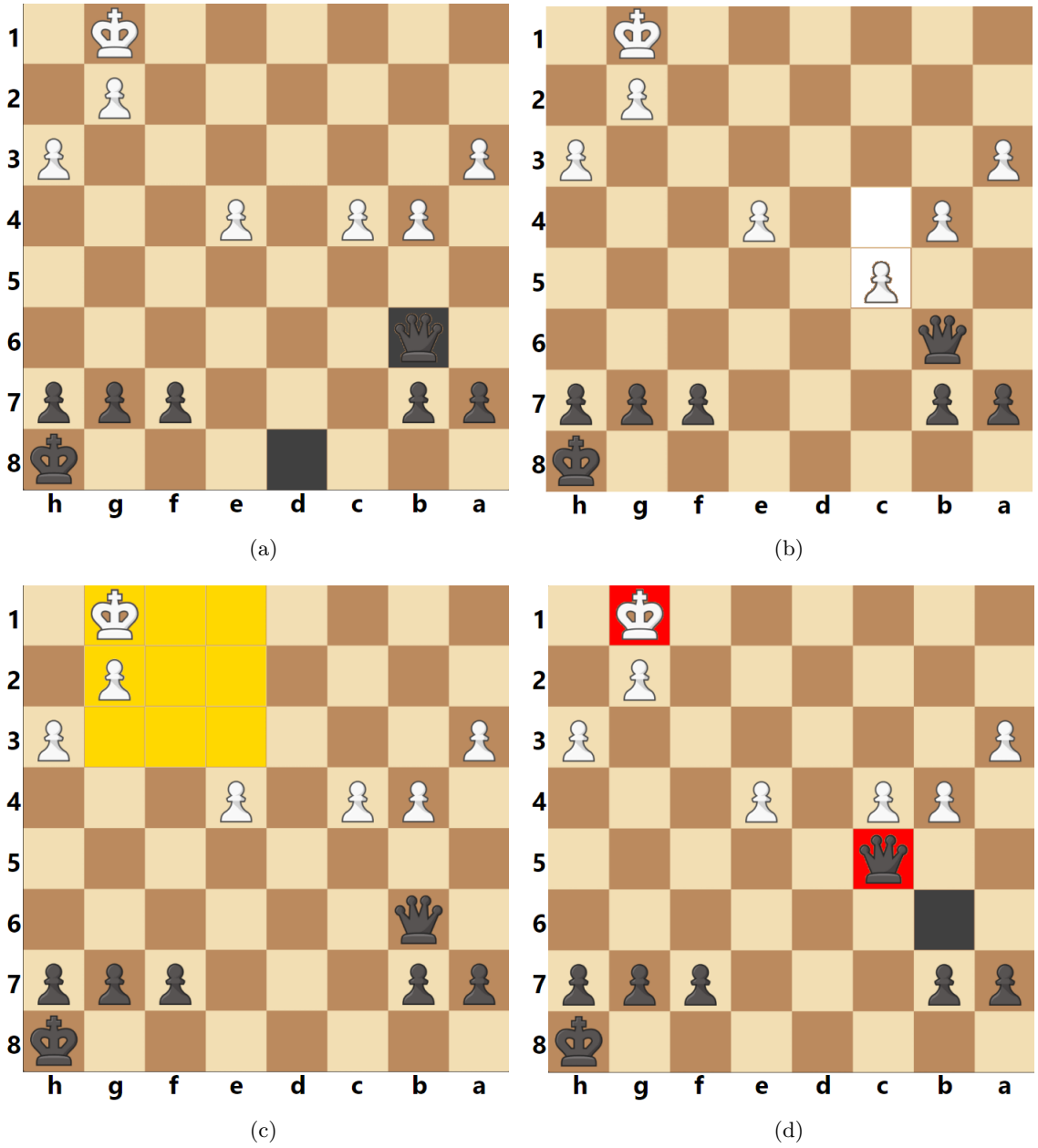


Figure 7: Suppose black moved its queen from d8 to b6 as shown in (a). To protect the king from the check, a pawn move from c4 to c5 shown in (b) is incentivized with the idea that the opponent is highly likely to sense the region where the king is in (c), (as it can be a potentially winning information) 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).

sets, almost no weight 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.

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

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

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

$$\text{sense score} \propto \text{mean sense impact} + f(size(I)) * \text{expected set reduction}$$

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

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 1 for V4 over V3.

4. 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 B. `Fianchetto`[1] topped the competition with a rating of 1759, compared to 1662 of the second-ranked `StrangeFish2` (Perrotta & Perrotta, 2019) and 1584 of the third-ranked `penumbra` (Clark, 2021). 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

1. 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.

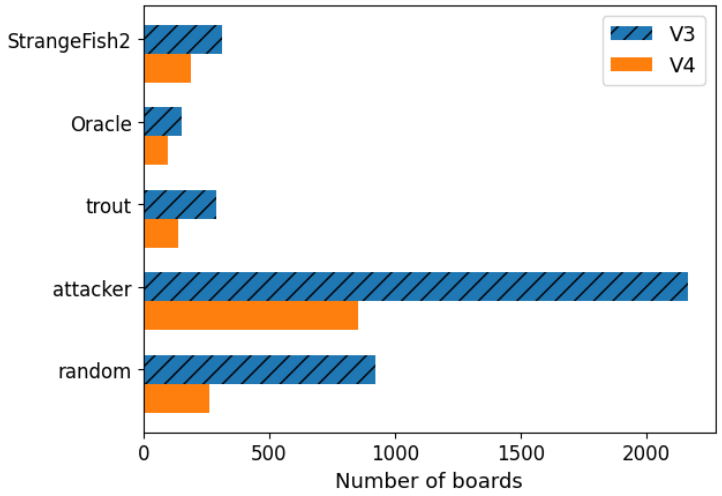


Figure 8: Average information set size in games played on the RBC server .

registered an overall win ratio of 91%. Table 4 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.

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

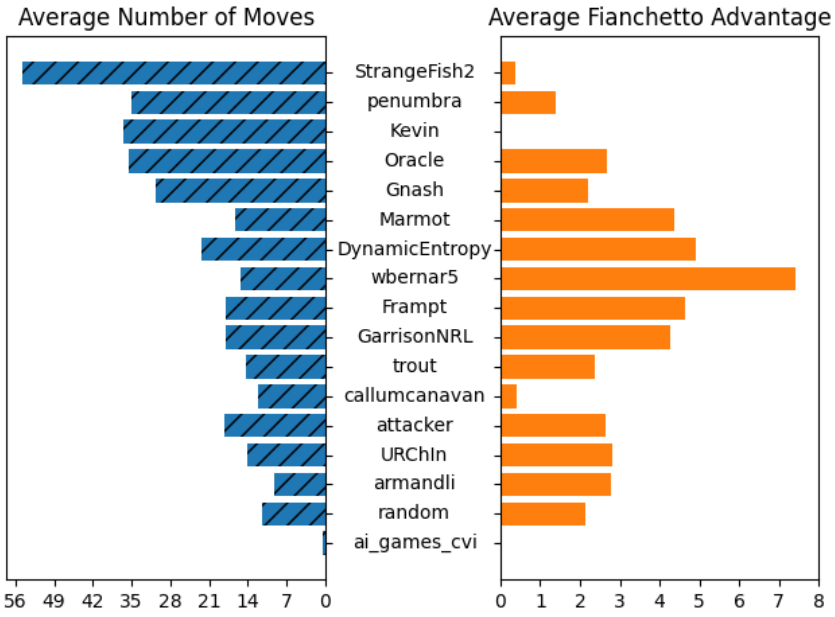


Figure 9: Averages of the number of moves per game (left panel) and `Fianchetto`'s material advantage per move (right panel) in its games against all opponents from the NeurIPS 2021 competition (arranged in decreasing order of ranking top to bottom). The `ai_games_cvi` agent never played any moves for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

Analyses performed on game logs from the tournament show trends for `Fianchetto` that are characteristics of expert play. In Figure 9, 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.

5. Future Work

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

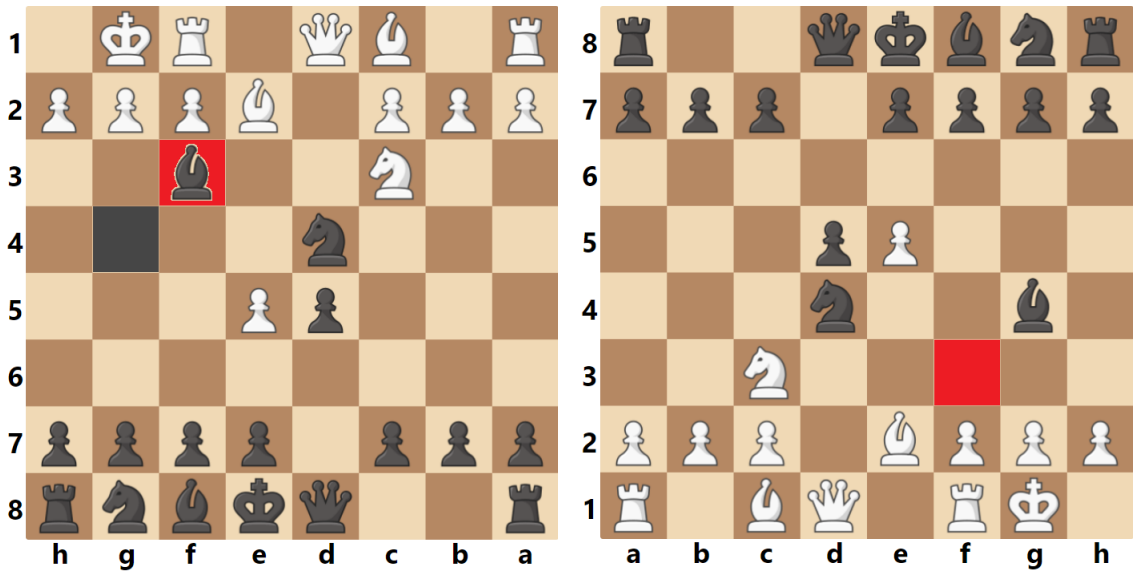
The current reliance on trained Chess evaluation functions such as Stockfish and Lc0 is more based on convenience than on principle. Given the many differences between RBC and Chess, there is a growing need for training evaluation functions specifically for RBC. Negotiating hidden state while so doing remains a technical challenge (Clark, 2021). 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, the endgame (with few pieces and an exploding information set) becomes a formidable challenge for current approaches, and merits more attention from a modeling perspective.

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 10a, 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 10b, 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 10

- 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 11 shows the winning move played by white to capture black's king. In the NeurIPS competition, each player had 15 minutes on the clock without increments to complete the game.

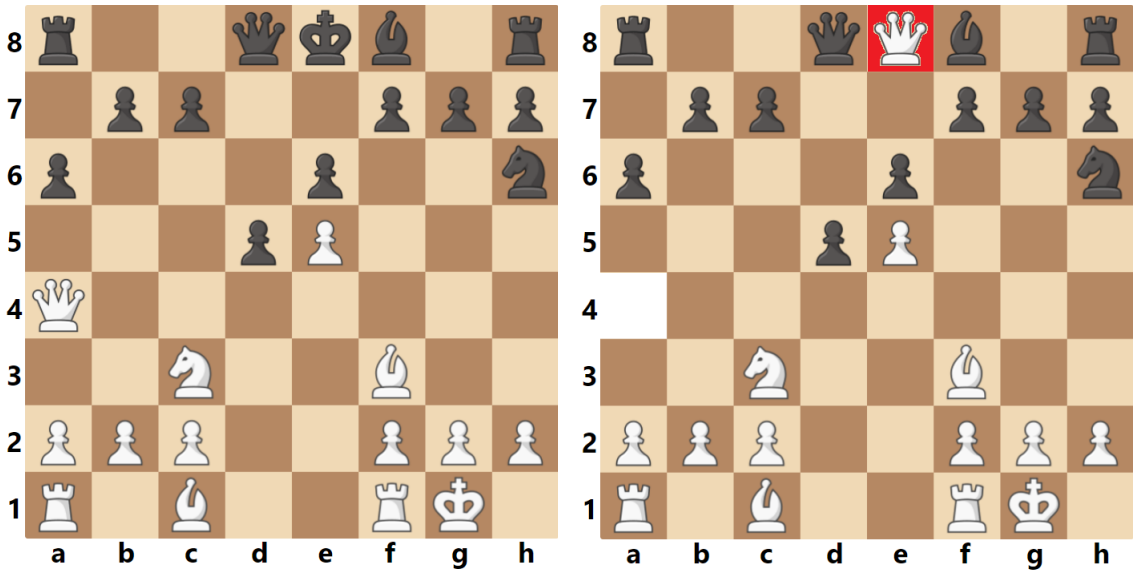


Figure 11

- 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 an opponent's piece (using a queen, bishop, or rook) results in the capture of that piece, and the sliding move is halted at the position of the capture. The transition in Figure 12 shows white's attempt to play the queen from d1 to d8, which results in capturing the black 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 13a. The same using a pawn move (from d2 to d4), as shown in 13b 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 14a or a pawn forward move (d5-d4) as shown in Figure 14b, 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 the check itself is removed.

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

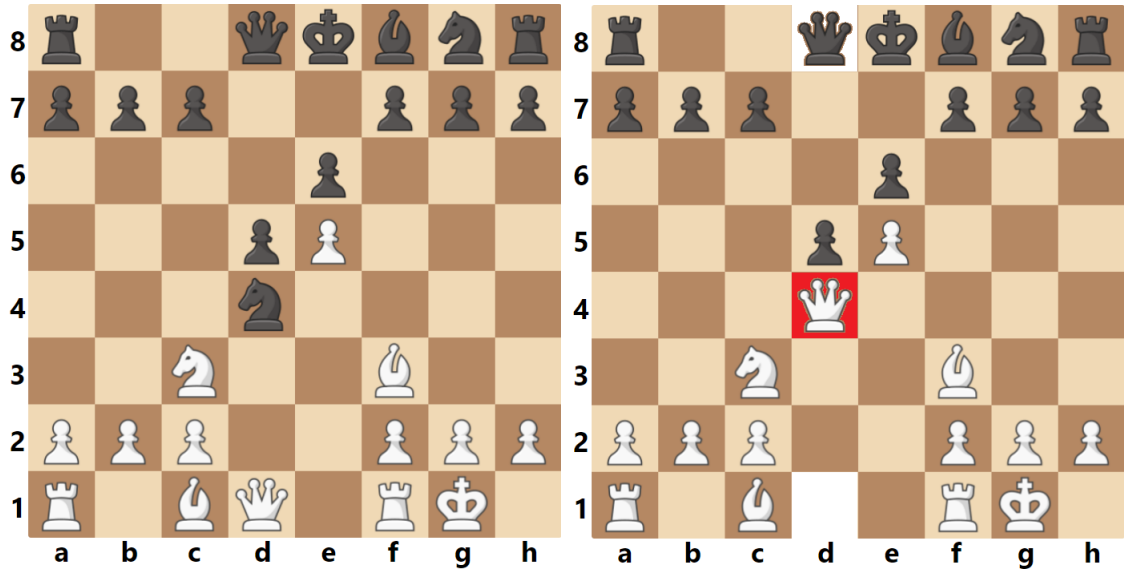
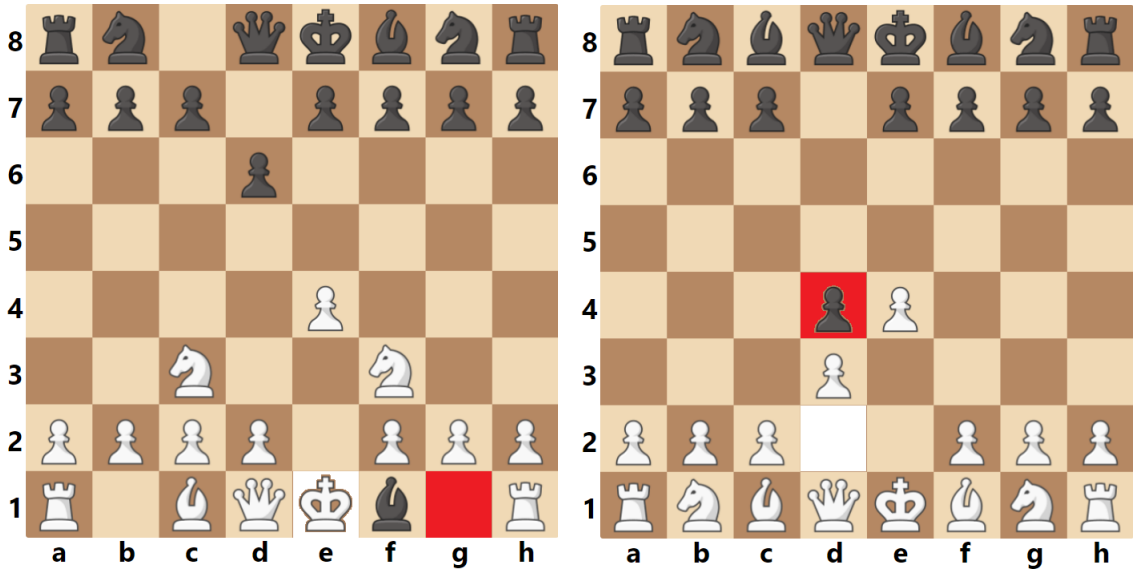


Figure 12



(a) Castle slide

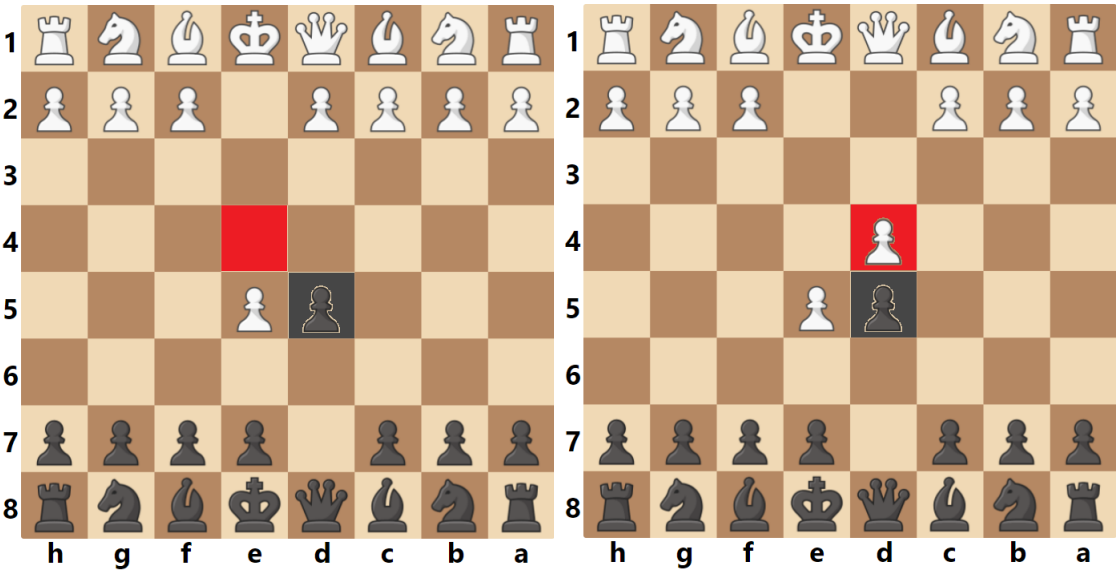
(b) Pawn slide

Figure 13

Appendix B. 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 players A and B are

$$\mathrm{E}_A = \frac{1}{1 + 10^{(R_B - R_A)/400}} \, , \, \mathrm{E}_B = \frac{1}{1 + 10^{(R_A - R_B)/400}}$$



(a) Illegal capture (b) Illegal move

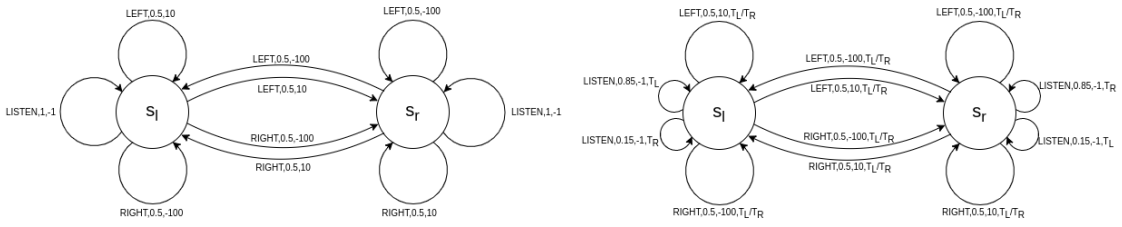
Figure 14

When a player's actual tournament scores exceed (or fall short) their expected scores, the Elo system (Coulom, 2008) takes this as evidence that the 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 C. Suboptimality Illustration through the Tiger Problem



(a) MDP (b) POMDP

Figure 15: The MDP and the POMDP for the 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 15a. The 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 observations T_L or T_R with uniform probability in any of the states.

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 the left door ($b > 0.9$), then the optimal action is RIGHT; if it believes with a 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 16 (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:

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

The above equation gives the policy in Figure 16a 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 16b.

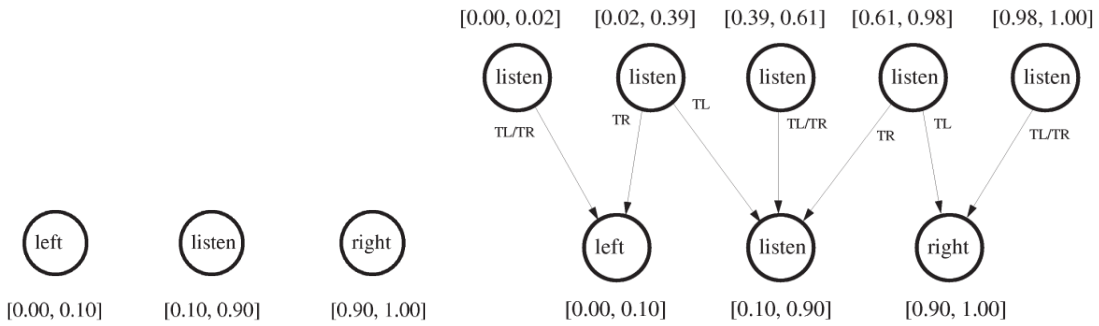
Since we know that plainly approximating the POMDP policy from the MDP policy is suboptimal, we can add our own rules to get a better POMDP policy. Consider the rule that the "LISTEN" action is usually better than other actions when $T > 1$. We incorporate

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 a 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 16: Optimal policy tree for the POMDP of the Tiger problem.

References

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, 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. (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.

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. (2021). 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.

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.

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.

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

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/.

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.

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., 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.

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.

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