Iterative Tree Search in General Game Playing with Incomplete Information

Armin Chitizadeh and Michael Thielscher

UNSW Sydney, NSW 2052 Australia
{a.chitizadeh, mit}@unsw.edu.au

Abstract. General Game Playing (GGP) is concerned with the development of programs capable of effectively playing a game by just receiving its rules and without human intervention. The standard game representation language GDL has recently been extended so as to include games with incomplete information. The so-called Lifted HyperPlay technique, which is based on model sampling, provides a state-of-the-art solution to general game playing with incomplete information. However, this method is known not to model opponents properly, with the effect that it generates only pure strategies and is short-sighted when valuing information. In this paper, we overcome these limitations by adapting the classic idea of fictitious play to introduce an Iterative Tree Search algorithm for incomplete-information GGP. We demonstrate both theoretically and experimentally that our algorithm provides an improvement over existing solutions on several classes of games that have been discussed in the literature.

Keywords: General Game Playing with Incomplete Information · Learning · Valuing information · Fictitious play

1 Introduction

Designing a player for games with incomplete information[1] has been studied in different areas. Vector minimax is a well-known and relatively general technique, which has been shown to solve the so-called strategy-fusion problem [6]. A limitation of the vector minimax technique is to rely on a specific game structure: It is limited to games in which all the moves can be seen except the first ones by the random player, e.g. where cards are being shuffled [4]. Another relatively general technique for playing incomplete-information games is Information Set Monte Carlo Tree Search (ISMCTS) [2]. This technique generalizes the well-known Monte Carlo tree search to incomplete-information games. Thanks to the use of Monte Carlo simulations, this technique performs well for large games. One of its limitation is given by the assumption that all states in an information set are equally likely. Due to this limitation, ISMCTS fails to properly model opponents or to play games with non-uniform probability distributions

[1] In game theory the term *imperfect information* is used to refer to the class of games in which players lack full information about the state of the game. On the other hand, in AI it is more common to use the expression *incomplete information* for problems in which agents lack full information. It has become customary in GGP to follow the standard AI terminology.

over possible states such as the famous Monty Hall problem [9]. Counterfactual Regret Minimization (CFR) [23] was introduced to solve the imperfect-information game of poker. This technique performs well thanks to poker specific optimisations [12]. However, its general implementation failed to perform equally well in other games with incomplete information. One of the reasons is the high computational complexity of $\mathcal{O}(I^2N)$, where I is the number of information sets and N the total number of states in the game [13]. All of the previously mentioned techniques perform well in some games, but it is believed that their success relies heavily on game-specific expertise of their developers and tailored algorithms.

More recently, AlphaZero [20] was able to learn the game of chess from scratch and through self-play to a point where it was able to beat Stockfish [16], the hitherto best chess engine. AlphaZero uses a relatively general approach compared to its predecessor AlphaGo [20]. However, AlphaZero is only applicable to complete-information two-player board games [20].

General Game Playing is concerned with the development of an AI capable of playing any arbitrary finite game effectively by just receiving the rules of the game without any human intervention. GGP programs receive the game rules in the form of the general Game Description Language (GDL), which has a Prolog-like syntax [14]. The first version of GDL was designed to only model deterministic games with full observability. Later, the extension GDL-II was developed for general game playing with incomplete information [21]. This extended general specification language allows to describe any finite game with incomplete information and randomness, for example, Poker or Backgammon, to a general game-playing system; however, designing such a system has remained a challenge.

The first successful players for GGP-II approached the problem by grounding all unknown variables and generating a set of sampled complete-information game states. They then searched each state sample and averaged the reward for moves over each of them in order to find the optimal move [17, 3]. However, searching on a set of sampled complete-information states for a game with incomplete information has its limitations, as pointed out in a recent paper on a technique called lifted HyperPlay [18]. Specifically, when searching separate complete-information samples, no extra value be put on moves that provide additional information about the current game state. This was a limitation of all previous approaches that combined search with complete-information sampling and was the main motivation for the development of the two state-of-the-art techniques in general game playing with incomplete information: HyperPlayer with Incomplete Information (HP-II) [18] and the so-called Norns algorithm [8].

HP-II uses nested players to simulate games. Through this technique, it can value knowledge-gathering moves, and it can also value moves that prevent the opponents from gaining helpful knowledge. However, this comes at the price of two main limitations of HP-II itself: high resource consumption [19] and, more importantly, shortsightedness when it comes to valuing information, as we will demonstrate in this paper. The aforementioned Norns algorithm, which uses Action-Observation Trees (AOT) to simulate a game and to determine the value of information, also suffers from high resource consumption and, more importantly, is restricted to single-player games [8].

In this paper, we introduce *Iterative Tree Search* (ITS) to overcome the limitations of HP-II and Norns. Our main motivation was to find a technique that, in principle, can correctly solve GDL-II games in general provided their search space is suitably small, rather than finding a technique that is applicable to large games. Our ITS combines the classic idea of *fictitious play* [1] with incomplete information tree search. Fictitious play learns the behavior of a rational opponent by self-playing a game several times. Incomplete information tree search is able to model an information set at every step of the game.

The main contributions of our paper are as follows: We formally introduce Iterative Tree Search as a new approach to general game playing with incomplete information. We theoretically analyse both the ITS and the HP-II algorithm and show the limitations of HP-II compared with ITS on different classes of games. We also report on experimental results with an implementation of the ITS algorithm to demonstrate its advantages over the existing techniques.

2 Background

In this section, we will briefly describe General Game Playing with Incomplete Information (GGP-II) and Fictitious Play (FP). For further details we refer the reader to [18] and [1].

2.1 General Game Playing with Incomplete Information

Players in General Game Playing (GGP) are given the rules of a game in the declarative Game Description Language (GDL) [14]. States in GDL are defined as sets of true facts. The initial state and terminal states are distinguished, and rewards are given to players at terminal states. In GDL-II, Nature is modeled as a special-purpose "random" player. This player chooses its moves at random with uniform probability, and it has the same reward values at all terminal states. Logical rules are used to describe the legal actions and their effects on a game state. In GDL-II, players' moves are hidden from each other, but players may receive "observation tokens" after each joint move [21]. The only way of learning something about the moves by other players is through these perceptions. Rules of the game explicitly describe under which conditions a player makes an observation. When the game ends, players will be notified and are given a reward value. In GGP, by convention the minimum reward value is 0 and the maximum reward value is 100. The goal of players is to achieve the maximal reward.

Formalisation In this paper, we will not be concerned with a set of GDL rules themselves but rather consider the induced game tree, including players' perceptions [21]. GDL and GDL-II allow us to describe games with simultaneous moves. For simplicity of explanation, we will use the standard transformation by which joint-move incomplete-information games in GGP-II can be converted into sequential incomplete-information games.

Definition 1. *Let $G = \langle S, R, M, \Sigma, s_0, Z, u, do \rangle$ be a game with incomplete information, where:*

- *S is a set of states;*
- *R is a set of players, and $R(s)$ is a function which given state s provides the player whose turn it is;*
- *M is a set of moves, and $M(s)$ is the list of legal moves at state s by the player*[2]*;*
- *Σ is a set of perceptions, and $\Sigma(s)$ is the list of perceptions for $R(s)$ from initial state to s;*
- *$s_0 \in S$ is the initial state of the game;*
- *$Z \subset S$ is the set of terminal states, for which we have $M(z) = \emptyset$ for any $z \in Z$;*
- *$u : Z \rightarrow \Re^{|R|}$ is the terminal utility function;*
- *$do : S \times M \rightarrow S$ is the successor function.*

To illustrate this, we look at an extended cutting wire game (ECW), adapted from a cooperative game presented in [18] and which we use later in section 4.1. Figure 1 shows the left part of the game tree. The roles are $R = \{random, cutter, teller\}$. At first, the random player arms a bomb, and only the teller sees which of two wires is used for this purpose. For the next two moves, then, the teller can either decide to tell which wire was used, or wait. Telling first costs 20 points and telling later costs 10 points for both players. Through telling, the cutter is informed about which wire he should cut to disarm the bomb. At the end, the cutter has to decide which wire to cut. Cutting the correct one gives both players 100 points (minus the aforementioned costs). Otherwise, they get 0. We use the sequence of moves as subscript to denote a state in the game tree, for example s_{rtwr}. Examples for the set of legal moves and list of percepts for a state are: $M(s_{rw}) = \{wait2, tell2\}$ and $\Sigma(s_{rtt}) = [(), red, red]$ respectively. As the cutter reaches its decision state s_{rtt}, he has received three perceptions along the path. The first perception is empty because the first action by random does not result in any perception for the cutter. The second and third both are "red" because, in this particular state, the teller has chosen $tell1$ and $tell2$. The final utility in this case $u(s_{rttr}) = 70$. The $do(S, M)$ function returns the next state, e.g. $d(s_r, wait) = s_{rw}$.

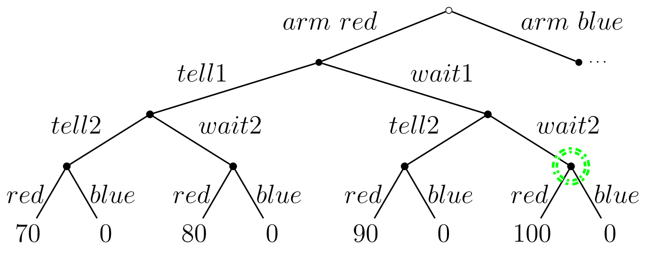

Fig. 1: Part of the game tree for the extended "cutting wire" game, with the red wire being armed. The green circle means the node is in an information set together with one other node in the other half of the game tree (because the cutter has not received any observation token).

[2] Each move is unique. Having similar names for moves at different states does not mean the moves are the same.

2.2 Fictitious Play

In game theory, fictitious play has been suggested as a learning technique [1]. It was originally designed for one step joint-move games, that is, normal form games. It is known to find a Nash equilibrium (NE) in the time-average sense for two-player zero-sum games, for games solvable with iterated strict dominance, and for so-called identical interest games as well as potential games [15]. Recently, the technique has been extended to some sequential-form games, for example, full-width extensive-from fictitious play (XFP) [10] and neural fictitious self-play (NFSP) [11]. The XFP technique learns a strategy which is realization-equivalent to the normal form fictitious play, meaning it considers a strategy as a whole. For this reason, this technique suffers from the curse of dimensionality. The NFSP technique uses sampling and neural networks to learn an NE strategy for the game. It was able to play limited Texas Hold'em successfully, but this game has no information-gathering moves.

The standard fictitious play updates a mixed policy after each iteration by averaging the previously played moves [1]. The updating algorithm can be mathematically described as follow:

$$\pi_r^{t+1} = \frac{\pi_r^t}{t} + \frac{\pi(b(\pi_{-r}^t))}{t+1} \tag{1}$$

Here, r is the player and t is the iteration index. π_r^t is the mixed policy for player r at iteration t and π_{-r} is the mixed policy for all players except r. Mixed policy π_r defines the probabilities for player r to choose a move at each state where it is this player's turn.

Given the mixed policy π_{-r} of other players, player r finds the best response as follows:

$$b(\pi_{-r})^t = \operatorname*{argmax}_{m \in M} util(m, \pi_{-r}^t) \tag{2}$$

Here, $b(\pi_{-r})$ is the best-response function which returns the best move m_r for player r given the mixed policy for the other players; and $util(m_r, \pi_{-r})$ is the reward function for player r if he plays move m given the strategy π_{-r} for the opponents. In this paper, we refer to π as a function which takes a move and returns a policy with the probability of the given move being 1 and all others being 0.

3 Iterative Tree Search

In this section, we will introduce a novel algorithm called the Iterative Tree Search (ITS) for GGP-II. ITS is an offline search, meaning it finds the best move before the game begins, and then during the game, it plays based on the pre-calculated mixed move policy. We first need to generate the incomplete-information tree and initialize its probabilities and utilities of the states; then we need to update the states' values and move probabilities iteratively, all within the given time limit before the start of the game.

3.1 Initialising the Tree

The first step in initialising the incomplete-information game tree is to represent indistinguishability of states. We use information sets for this purpose, which can be determined from the observation tokens a player has received [21].

Definition 2. *Let G be a GDL-II game as in Def. 1, then:*

- $I(s) : S \rightarrow S^*$ *is the information set function which takes a state* s *and returns a set of states. The given state and all the states in the returned set are indistinguishable by the player for whom* s *is a decision node.*

Since we are converting games into sequential form, the player for which states are indistinguishable is always the player whose turn it is. The following equation describes formally how an information set is generated:

$$I(s) = \{x \in S \ : \varSigma(s) = \varSigma(x) \wedge \xi_r(s) = \xi_r(x) \wedge r = R(s)\} \tag{3}$$

Here, $\xi_r : S \rightarrow M^*$ is the *history* function which provides the sequence of moves of player r from the start to the provided state. In our ECW example, the only non-singular information set is $I(s_{rww}) = \{s_{rww}, s_{bww}\}$. We refer to this information set as I_{ww}.

Our ITS technique assigns probabilities to moves and states in each information set. The probability of a state can be recursively calculated from the probabilities of the moves. Through iteratively updating both probabilities, we can obtain a mixed strategy for every information set in a game tree that can be used to play optimally during the game.

When initializing the tree, we provide uniform probability to the moves in a state. A mixed policy is beneficial in a sense that it can prevent the opponent from predicting our moves. In the ECW example, the first iteration will be the initial move probability of $\mu(tell1) = \frac{1}{2}$.

3.2 Iterative Probability Update

After populating the probabilities of the moves, we calculate the probability of every state in an information set to be the true state of the game.

Definition 3. *Let G be a GDL-II game as above, then:*

- $\mu : M \rightarrow [0, 1]$ *is the probability function which, given a move, returns the probability of the move to be chosen by the player.*

The initial value of μ^0 is uniform over all the moves in a state, that is, $\mu^0(m) = \frac{1}{|M(s)|}$ for all $m \in M(s)$. Based on Definition 3 we can define for player r the mixed policy π_r with move probability μ as follow:

$$\pi_r = \{\mu(m) | r = R(s) \wedge m \in M(s)\} \tag{4}$$

Definition 4. *Let G be a GDL-II game as defined, then:*

- $\rho : S \rightarrow [0,1]$ *is the probability function which for a state returns the probability of the state to be the true state in its information set.*

ρ can be calculated with the help of the probability factor.

- $\rho Factor(s') = \rho Factor(s) * \mu(m)$
  where $s' = do(s,m)$ *and* $\rho Factor(s_0) = 1$.

With this definition we can calculate the ρ of states in an information set as follows:

$$\rho(s) = \frac{\rho Factor(s)}{\sum_{s_n \in I(s)} \rho Factor(s_n)} \tag{5}$$

We then need to calculate and assign utility values to all states in the game. we extend the terminal utility function defined in a game to a function that assigns a utility to all states. The utility of each state can be calculated based on the utilities of the successor states and the probabilities of moves. The recursion will be as follow:

$$u(s) = \sum_{m \in M(s)} [u(do(s,m)) * \mu(m)] \tag{6}$$

The base cases are the utilities of terminals which are given as part of the game description.

In our ECW example, at the first iteration all $\rho Factor()$ are equal at each level. This game has a branching factor of two at all nonterminal nodes, hence $\mu^0(m) = \frac{1}{2}$ for all moves m in all states of the game. We can then calculate $\rho Factor(s_{rwt}) = \mu^0(r) * \mu^0(w) * \mu^0(t) * \rho Factor(s_0)$ which is $\frac{1}{2^3}$. As a result, the values $\rho(s_{rww})$ and $\rho(s_{bww})$ are $\frac{1}{2}$ while all the others are 1 because they all form their own singleton information set.

At the next stage, we need to calculate the value of each move in its information set and then set the move with the highest utility as the chosen one. To calculate the reward of a move we will consider all the states in the information set.

Definition 5. *Let G be a GDL-II game as defined, then:*

- $chosenMove : I \rightarrow M$ *is the move selection function which chooses the move with the highest reward for the player in an information set as:*

$$chosenMove(i) = argmax_m \Big[\sum_{s \in i} \rho(s) * u_r(s') \Big] \tag{7}$$

Here, for each s we have $s' = do(s,m)$ and $r = R(s)$. Similar legal moves in an information set have the same utility. As a result, the chosen move will be the similar move for all the states in the same information set.

In the ECW example, for the cutter the $chosenMove$ in singleton information sets will be the move that leads to the highest possible utility at termination. However, for $cut\ red$ the average utility at I_{ww} is $\rho(s_{rww}) * u(s_{rwwr}) + \rho(s_{bww}) * u(s_{bwwr})$. This is similar to the $cut\ blue$ action at I_{ww} and equals 50. In the long run, as the game is symmetric, $\rho(s_{rww})$ and $\rho(s_{bww})$ will stay the same. So $chosenMove$ switches randomly between the two moves in I_{ww}.

For the last stage, we need to update the move probability function μ as follows: Let m be $chosenMove$ and m' all other moves.

$$\mu^{t+1}(m) = \frac{(\mu^t(m) * t) + 1}{t+1} \tag{8}$$

$$\mu^{t+1}(m') = \frac{\mu^t(m') * t}{t+1} \tag{9}$$

We will now reset all utilities for non-terminal states and $\rho Factor$ values. The iteration can be continued until the end of the pre-game calculation. In this way, we can come up with an approximation of optimal moves within a given time limit.

Coming back to the ECW example, in the long run, for the cutter's states the probability of cutting the correct wire approaches 1 except for the states in I_{ww} in which the probabilities of $cut\ blue$ and $cut\ red$ stay the same. As can be seen, the computational complexity of the algorithm is linear in the number of game states.

The Iterative Tree Search is summarized as Algorithm 1 below.

Algorithm 1 Iterative Tree Search

```
 1: Generate allInfomationSets                                  ▷ using (3)
 2: $t \leftarrow 0$
 3: for all $m \in M$ do
 4:     Initialise $\mu(m)$
 5: while time allowed do
 6:     for all $s \in S$ do
 7:         $u(s) \leftarrow 0;\ \rho Factor(s) \leftarrow 0;\ \rho(s) \leftarrow 0;\ t \leftarrow t+1$
 8:     $\rho Factor(s^0) \leftarrow 1$                           ▷ $s^0$ is the initial state
 9:     for all $s \in S$ & $m \in M(s)$ do
10:         $s' \leftarrow do(s,m)$
11:         $\rho Factor(s') \leftarrow \rho Factor(s) * \mu(m)$
12:     for all $s \in S$ do
13:         $\rho(s) \leftarrow \frac{\rho Factor(s)}{\sum_{s_n \in I(s)} \rho Factor(s_n)}$
14:     for all $s \in S$ & $m \in M$ do                       ▷ For terminals, it is already set
15:         $u(s) \leftarrow u(s) + (u(do(s,m)) * \mu(m))$
16:     for all $i \in$ allInfomationSets do
17:         $chosenMove(i)$
18:         $\leftarrow \underset{m \in M(i)}{\text{argmax}}[\sum_{s \in i} \rho(s) * u_{R(s)}(do(s,m))]]$
19:     for all $I \in$ allInfomationSets do
20:         $\mu(chosenMove(I)) = \frac{(\mu(chosenMove(I)) \times t)+1}{t+1}$
21:         for all $m \in M(I)$ & $m \neq chosenMove(i)$ do
22:             $\mu(m) = \frac{\mu(m) \times t}{t+1}$
```

4 Analysis

In the following, we characterize the classes of GDL-II games that our ITS can solve. We will resort to the theory of fictitious play to this end. We also demonstrate how HP-II fails in these games and show experimental results to confirm our observations. All mentioned games have either been previously introduced in the literature or are extensions of games from the literature.

4.1 Games with Dominant Pure Strategy and Single Player Games

ITS can correctly solve games in which there exists a dominant pure strategy. Having a dominant strategy means playing one specific move at each information set guarantees the player the highest reward. Also, it means that the actions of the opponents will not affect the decision of the player. If a single-player game with incomplete information has an optimal strategy, then this will be a pure dominant strategy, as the random player has no intention of changing its strategy.

The ITS algorithm at the first iteration assigns equal probabilities to all moves from the same information set. This means that no $\mu(m)$ will ever have zero probability. As the calculation progresses, the player with a dominant strategy tends to play more of it because states on the path of a dominant strategy have the highest rewards and the probability of the parent state never changes. As a result, the probability of playing a dominant strategy increases with each iteration. So $\rho(s) * u_r(s')$ always increases and will always be the chosen move. We can use the ECW game to illustrate how ITS can correctly play this type of games and also why HP-II fails.

Example: Extended Cutting Wire The game was inspired by the "cutting wire" game originally published to motivate the HP-II technique [18]. In the original version, the teller can $tell$ or $wait$ only once. HP uses complete-information sampling and therefore fails to solve this problem (since information gains can have no value), but HP-II successfully solves the original version of the game. We have extended the cutting wire in order to show that ITS can correctly value information in deeper parts of the game tree while HP-II is "short-sighted" in this regard. ITS correctly chooses to $tell$ at the second level, where it is less costly, while the HP-II player chooses to $tell$ at the first level because of its short-sightedness when valuing information.

HP-II uses nested players to overcome the limitations of model sampling as used in HP. However, at each step it sees information-gathering moves only one level ahead. This causes the algorithm to choose $tell1$ and $wait2$ rather than $wait1$ and $tell2$. More precisely, the move selection policy of HP-II $\boldsymbol{\pi}_{hpii}$ can be described as:

$$argmax_{m \in M(s)} \left[\sum_{s' \in I(s)} eval(replay(s_0, I_{r \in R}(do(s', m)), \boldsymbol{\pi}_{hp}), \boldsymbol{\pi}_{hp}, n) \right] \quad (10)$$

We have modified the algorithm to suit sequential games.[3] At state s_r, $\boldsymbol{\pi}_{hpii}$ chooses the move based on which of $\boldsymbol{\pi}_{hp}(s_{rw})$ or $\boldsymbol{\pi}_{hp}(s_{rt})$ gives the higher expected reward. The HP-II policy, $\boldsymbol{\pi}_{hp}$, uses a Monte-Carlo search, so the move $wait1$ returns $\frac{50+90}{2}$ and move $tell1$ returns $\frac{70+80}{2}$. As a result, HP-II considers $tell1$ a better move than $wait1$. We refer to this problem as *short-sighted information valuation*.

Our new ITS algorithm can correctly value the information anywhere in the game tree. To illustrate this, recall the explanation for HP-II. As previously described, for the $arm\ red$ part of the game tree, the cutter's utilities in the long run will be: $u(s_{rtt}) = 70$, $u(s_{rtw}) = 80$, $u(s_{rwt}) = 90$ and $u(s_{rww}) = 50$. Then using equations (7) and (6) we obtain $u(s_{rw}) = 90$ and $u(s_{rt}) = 80$. As a result, the $chosenMove(I(s_r))$ will be $wait1$. Analogously, we can show that $chosenMove(I(s_b))$ is $wait1$ too.

To validate our claims we have run ITS with the extended cutting wire game. The graph in Figure 2 shows the probability of the $tell$ move at the two different stages of the game during the first 1,000 iterations. If the probability of the telling action is high in any state, then the probability of waiting is low and vice versa. As can be seen from the graph, the probability of choosing the telling action twice quickly converges to zero at early iterations. The probability of the first, more costly telling move also drops to almost zero in less than 1,000 iterations. In fact, after less than 200 iterations ITS will very likely choose to wait first and then to tell.

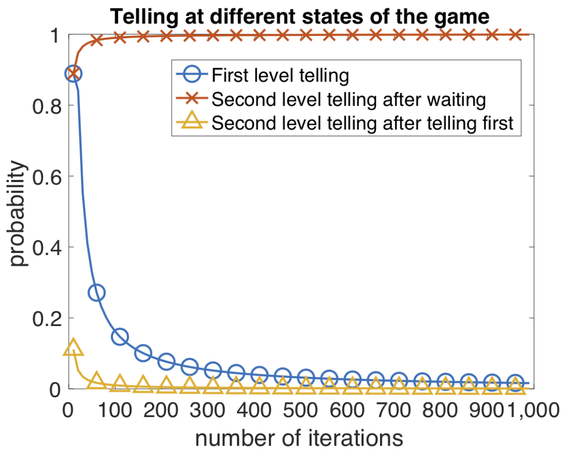

Fig. 2: Probability of $tell$ at different states during the first 1,000 iterations in the extended cutting wire game.

[3] The $replay$ function replays from the initial state to the given state. In the sequential ECW game, where the only information set belongs to the $secondPlayer$, the $replay$ function will be reduced to a simple information set function.

4.2 Non-Locality Problem

Frank and Basin [5] have formalized and analyzed the problem of *non-locality*. Non-locality happens when an algorithm only considers children of a state to find the best move for that state. Our ITS algorithm models the opponent, which can be shown to lessen the impact of this problem.[4]

Example: The game in Fig. 5 [6] To show the ability of ITS algorithm to solve games that exhibit the non-locality problem, we consider the motivating example from [5]. In this game, the first move by random is only visible to the $secondPlayer$ while players' moves are visible to each other. The random choice places the game in a particular world, and the utilities for the players depend on their moves and the world they are in.[5] Figure 3 depicts the game tree for this game and shows $firstPlayer$'s optimal strategy. This strategy guarantees that the first player always receives 1 in w_1. ITS is able to always play correctly at states a and e from the first iteration on because these are the dominant moves. Figure 4 shows the mixed strategy of the player at state d after 100 iterations. After less than 100 iterations, our implemented ITS was also able to play left at d with a probability of 99%.

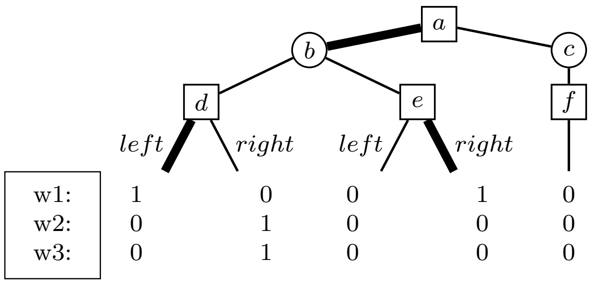

Fig. 3: Game with non-locality problem represented in world model.

4.3 One-Step Joint-Move Two-Player Zero-Sum Games

We show that ITS reduces to fictitious play for this category of games, and since fictitious play is known to solve these games, ITS can solve them too. For ITS we convert this category of games into a two-step sequential game with incomplete information. All moves of the $firstPlayer$ lead to states that are all in the same information set. To show that the ITS algorithm works similar to fictitious play for this class of games, we will show that the updating policy and the move selection of fictitious play is similar to ITS. We refer to the player who moves first as the $firstPlayer$ and call the opponent

[4] Frank and Basin[7] have introduced the “Vector MiniMax” technique, which also just lessens, rather than completely avoids, the impact of non-locality.

[5] This game can, of course, be straightforwardly axiomatized in GDL-II as a GGP-II game [22].

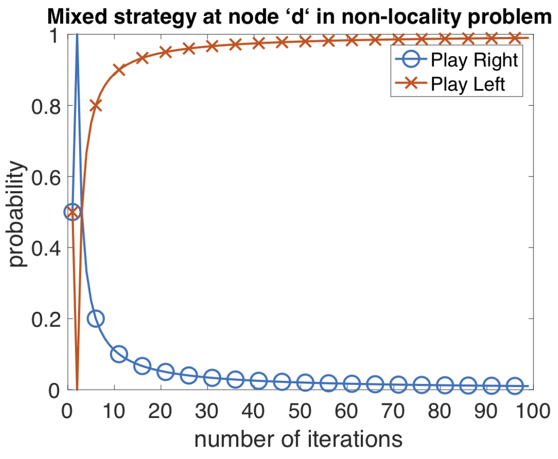

Fig. 4: Mixed strategy at state d in the non-locality game during the first 100 iterations.

the $secondPlayer$. For both players, updating the mixed policy π via equation (1) is identical to updating μ of moves in ITS, that is, equations (8) and (9).

For the move selection policy of ITS we need to consider each player separately. With regard to choosing the best move for the $firstPlayer$ in ITS, if we combine equations (7) and (6), then for $chosenMove()$ we will get:

$$argmax_{m_1}\Big[\sum_{s\in i}\rho(s)*\sum_{m_2\in M(s')}[u(do(s',m_2)*\mu(m_2))]\Big] \tag{11}$$

Here, s is the initial state s_0 and is in a singleton information set; s' is the state after the initial state, referred to as s_{m1} in what follows. $\rho(s_0)$ is always equal to 1. We also substitute $do(s', m_2)$ with s_{m1m2}, which is a terminal state with a fixed reward. Also, to simplify notation, we replace $M(s')$ with M_2. For $chosenMove()$ we then obtain the following:

$$argmax_{m1}\Big[\sum_{m2\in M_2}[u(s_{m1m2}*\mu(m_2))]\Big] \tag{12}$$

Considering the relation of mixed move policy and mixed policy described in equation (4), this equation is indeed equal to the best respond equation (2) for fictitious play.

With regard to choosing the best move for the $secondPlayer$ in ITS, the action of the $firstPlayer$ changes the probability of state $\rho(s)$. Since the $secondPlayer$'s decision states are all in the same information set and are all generated from the initial state, $\rho Factor(s_{m1}) = \mu(m1)$. So $\rho(s_{m1}) = \frac{\mu(m1)}{\sum_{m\in M_2}\mu(m)}$. By definition the denominator is equal to 1. By substituting these in equation (7), for $chosenMove()$ we obtain:

$$argmax_{m2\in M_2}\Big[\sum_{m1\in M_1}\mu(m1)*u(s_{m1m2})\Big] \tag{13}$$

which is identical to equation (12) with $m1$ being replaced by $m2$. This completes the proof that ITS reduces to basic fictitious play for this category of the games and that, therefore, ITS is able to correctly play any one-step, joint-move zero-sum game with two players. As an example, we look at Biased Penalty Kick, which is a common game in game theory to show the ability of an algorithm to find a Nash equilibrium (NE).

Example: Biased Penalty Kick This is a well-known game to illustrate opponent modeling and the value of playing a mixed strategy: If the kicker shoots right and the keeper catches, the keeper gets 60; otherwise, the kicker gets 60. If the kicker shoots left the rewards will instead be 40. There is no pure NE, and the optimal mixed strategy in a NE for the kicker is to shoot 40% right and 60% left. For the goalkeeper, it is jumping 40% to the left and 60% to the right.

While both HP and HP-II only deliver pure strategies, ITS can solve this problem after just a few iterations as this game is a one-step joint-move game. To verify this claim we have run our implemented ITS algorithm on this problem. Figure 5 shows the mixed strategy of the goalkeeper for the first $10,000$ iterations. ITS quickly finds the correct probabilities for both players.

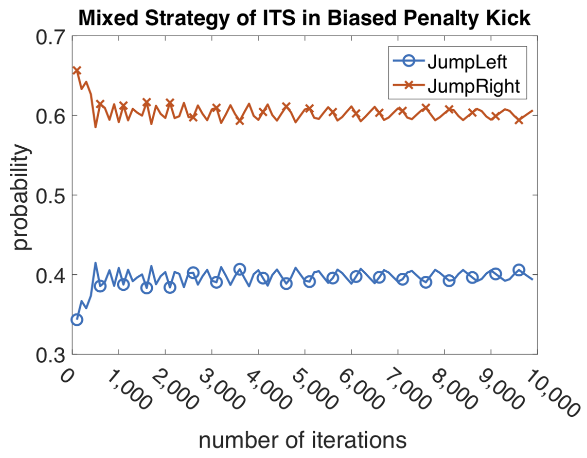

Fig. 5: Mixed strategy of the goal keeper for the biased penalty kick game for the first 10,000 iterations.

4.4 Move Separable Games

In this category of games, each player is only responsible for moves in one stage of the game. This means that when the $secondPlayer$ begins to move after a series of moves by the $firstPlayer$, then after the $secondPlayer$ has made their moves the game ends, and also the $firstPlayer$ cannot interrupt the $secondPlayer$'s course of actions. If the random player exists in the game, its actions are visible to the player of the corresponding stage of the game. In this class of games, the $secondPlayer$ may or may not be able to see some or all of the actions performed by the $firstPlayer$. We will show that these games will be reduced to a game where the $firstPlayer$ chooses a joint-move game for both players to play. HP-II fails to solve games in this class as it can only find a pure strategy, whereas again the ITS algorithm can solve this category of games.

First, consider the simpler case of games where the $secondPlayer$ cannot see the $firstPlayer$'s moves. The probability of a state to be the true state in the information set of the $secondPlayer$ depends on the sequence of the $firstPlayer$'s moves. This sequence can be considered as one single, combined move, whose probability is the same as the frequency with which the corresponding sequence is chosen. The game can therefore obviously be reduced to a one-step, joint-move game that is solvable by the ITS algorithm.

Now consider the case when the $secondPlayer$ can see some of the actions of the $firstPlayer$. The $firstPlayer$ can lead the $secondPlayer$ into one of possible information sets. The $\rho(s)$ of the states in each information set can be changed by the unobserved moves of the $firstPlayer$. Thus a game of this type can be reduced to a game where the $firstPlayer$ chooses a subgame of a one-step, joint-move game with the highest NE payout for himself among all subgames and then play the subgame corresponding to the NE strategy. As described in the previous section, ITS can solve this subgame and determine the payout for each NE. Choosing the subgame with the highest NE payout then just requires a simple search. In this way ITS can solve Move Separable Games. We end our analysis with an example of a game from the literature that falls into this category.

Example: Banker and Thief This game was used in the HP-II paper [18] to show the ability of HP-II to value the withholding of information. There are two banks in this game, one of which has a faulty alarm system. The owner of the bank that is faulty has to decide to distribute \$100 between the two banks in \$10 notes. The thief can see the distribution of the money between the banks but not which of the two is faulty. If the thief decides to rob the faulty bank then he succeeds in getting the money, otherwise the banker receives all the money left in his bank at the end of the game.

Using HP-II, the banker places \$40 in the faulty bank and \$60 in the other, implicitly making the assumption that the thief is greedy and will choose to rob the bank with \$60, which means the banker wins. This strategy was considered as the winning strategy in HP-II paper [18]. We claim that this is, in fact, a sub-optimal strategy as the banker wrongly assumes the thief to be greedy. Indeed, the thief might well assume the banker to assume that he is greedy, and hence he will decide to go after the \$40. The best strategy for this game must, therefore, be a mixed strategy so that the thief becomes indifferent to choosing a bank. Only then is the (mixed) strategy a Nash equilibrium. Different distributions lead to different NE with different expected payout. The highest expected payout for the banker is a \$50-\$50 distribution with an expected payout of \$25, while the expected payout for the \$40-\$60 distribution is just \$24. Since this is a Move Separable Game, ITS can solve it in contrast to HP-II.

To prove our claim, we have run our ITS algorithm with this game, but to make it more challenging we added two extra safe banks. The banker can then choose a distribution of his money in $10-chunks among four banks. There is a total of 287 ways to do so. Table 1 shows the probabilities for some mixed strategies for the banker in the case when the first bank has been selected as faulty by the random player. The theoretical analysis for this variant of the game shows that the optimal strategy is to put $50 in a faulty and $50 in any of the safe banks. As can be seen from the table, this is what the ITS algorithm will do in 94% of the times after one million iterations. Figure 6 shows how the probabilities for some of the 287 strategies evolve.

Money distribution	50 50 0 0	50 0 50 0	50 0 0 50	0 50 50 0	all others
Probability of choosing	35.67%	27.5%	30.88%	0%	5.95%

Table 1: Probability of choosing a money distribution by ITS in the banker and thief game.

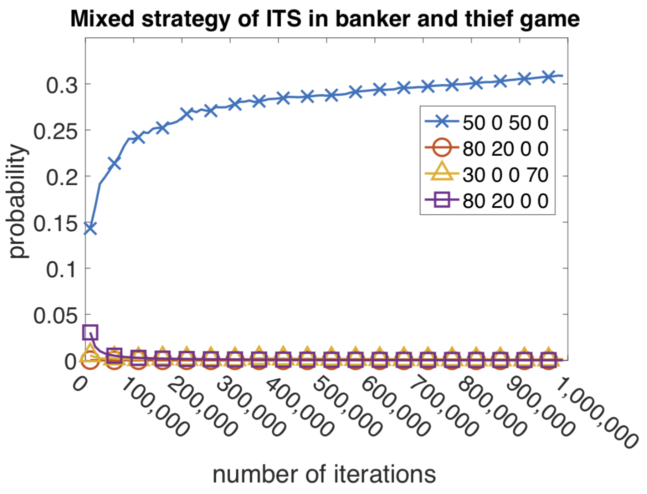

Fig. 6: The probability change toward equilibrium for four strategies in the banker and thief game when the faulty bank is the first one.

5 Conclusion

We have introduced the Iterative Tree Search (ITS) algorithm as a significant improvement over state-of-the-art algorithms, in particular HP-II, for general game playing with incomplete information. While HP-II is short-sighted on valuing information, our ITS algorithm has been shown to correctly value information in a game by gathering information at the lowest possible cost that promises the highest benefit. An ITS-based general game player is also able to withhold information from opponents and to play a NE on a number of classes of games. Moreover, HP-II is not able to compute mixed strategies, so it fails to find the best strategy in games that require opponent modeling. With ITS we can overcome these limitations by iteratively self-playing the game using an incomplete-information tree and thus learn the expected behavior of a rational opponent.

ITS has shown strong performance in playing small games. However, it failed to perform well in larger games such as poker. A related direction for future work is the addition of sampling techniques to ITS in order to break the curse of dimensionality.

References

1. Brown, G.W.: Iterative solution of games by fictitious play. Activity analysis of production and allocation **13**(1), 374–376 (1951)
2. Cowling, P.I., Powley, E.J., Whitehouse, D.: Information set monte carlo tree search. IEEE Transactions on Computational Intelligence and AI in Games **4**(2), 120–143 (2012)
3. Edelkamp, S., Federholzner, T., Kissmann, P.: Searching with partial belief states in general games with incomplete information pp. 25–36 (2012)
4. Frank, I., Basin, D.: A theoretical and empirical investigation of search in imperfect information games. Theoretical Computer Science **252**(1-2), 217–256 (2001)
5. Frank, I., Basin, D.A.: Search in games with incomplete information: A case study using bridge card play. Artif. Intell. **100**(1-2), 87–123 (1998). https://doi.org/10.1016/S0004-3702(97)00082-9, https://doi.org/10.1016/S0004-3702(97)00082-9
6. Frank, I., Basin, D.A., Matsubara, H.: Finding optimal strategies for imperfect information games. In: AAAI/IAAI. pp. 500–507 (1998)
7. Frank, I., Basin, D.A., Matsubara, H.: Finding optimal strategies for imperfect information games. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA. pp. 500–507 (1998), http://www.aaai.org/Library/AAAI/1998/aaai98-071.php
8. Geißer, F., Keller, T., Mattmüller, R.: Past, present, and future: An optimal online algorithm for single-player GDL-II games. In: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). pp. 357–362 (2014). https://doi.org/10.3233/978-1-61499-419-0-357, https://doi.org/10.3233/978-1-61499-419-0-357
9. Gill, R.: Monty hall problem. International Encyclopaedia of Statistical Science pp. 858–863 (2010)
10. Heinrich, J., Lanctot, M., Silver, D.: Fictitious self-play in extensive-form games. In: International Conference on Machine Learning. pp. 805–813 (2015)
11. Heinrich, J., Silver, D.: Deep reinforcement learning from self-play in imperfect-information games. CoRR **abs/1603.01121** (2016), http://arxiv.org/abs/1603.01121
12. Johanson, M.B.: Robust strategies and counter-strategies: Building a champion level computer poker player. In: Masters Abstracts International. vol. 46 (2007)
13. Long, J.R., Sturtevant, N.R., Buro, M., Furtak, T.: Understanding the success of perfect information monte carlo sampling in game tree search. In: AAAI (2010)
14. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General game playing: Game description language specification (2008)

15. Monderer, D., Shapley, L.S.: Potential games. Games and economic behavior **14**(1), 124–143 (1996)
16. Romstad, T., Costalba, M., Kiiski, J., et al.: Stockfish: A strong open source chess engine, https://stockfishchess.org/, retrieved May 1st, 2018
17. Schofield, M.J., Cerexhe, T.J., Thielscher, M.: Hyperplay: A solution to general game playing with imperfect information. In: AAAI. Citeseer (2012)
18. Schofield, M.J., Thielscher, M.: Lifting model sampling for general game playing to incomplete-information models. In: AAAI. pp. 3585–3591 (2015)
19. Schofield, M.J., Thielscher, M.: The scalability of the hyperplay technique for imperfect-information games. In: AAAI Workshop: Computer Poker and Imperfect Information Games (2016)
20. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815 (2017)
21. Thielscher, M.: A general game description language for incomplete information games. In: AAAI. vol. 10, pp. 994–999. Citeseer (2010)
22. Thielscher, M.: The general game playing description language is universal. pp. 1107–1112. Barcelona (Jul 2011)
23. Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: Advances in neural information processing systems. pp. 1729–1736 (2008)