Recursive Monte Carlo Search for Imperfect Information Games

Timothy Furtak and Michael Buro
University of Alberta, Edmonton, T6G 2E8, Canada
Email: {furtak|mburo}@cs.ualberta.ca

***Abstract*—Perfect information Monte Carlo (PIMC) search is the method of choice for constructing strong AI systems for trick-taking card games. PIMC search evaluates moves in imperfect information games by repeatedly sampling worlds based on state inference and estimating move values by solving the corresponding perfect information scenarios. PIMC search performs well in trick-taking card games despite the fact that it suffers from the strategy fusion problem, whereby the game's information set structure is ignored because moves are evaluated opportunistically in each world. In this paper we describe imperfect information Monte Carlo (IIMC) search, which aims at mitigating this problem by basing move evaluation on more realistic playout sequences rather than perfect information move values. We show that RecPIMC — a recursive IIMC search variant based on perfect information evaluation — performs considerably better than PIMC search in a large class of synthetic imperfect information games and the popular card game of Skat, for which PIMC search is the state-of-the-art cardplay algorithm.**

I. Introduction

In recent years AI research in the area of imperfect information games has flourished. For instance, in 2008 Poker program "Polaris" defeated a group of six strong human players in two-player limit Texas hold'em in a duplicate match setting [1], and in 2009 "Kermit" reached expert playing strength in Skat, a popular trick-based card game similar to Bridge [2]. The considerable progress in Poker is due to new techniques such as counter factual regret minimization for approximating Nash equilibrium strategies in smaller, abstract versions of the game [3], whereas in Skat fast perfect information Monte Carlo (PIMC) search combined with explicit state inference and heuristic state evaluation elevated programs to the next level.

Both approaches have distinct advantages and disadvantages. For instance, solving abstracted game versions off-line leads to fast on-line move computation because, essentially, available moves only have to be sampled from pre-computed probability distributions. However, finding good game abstractions that allow us to approximate move distributions in the original game well isn't trivial. A property of Poker is that the set of legal moves in each game state doesn't depend on the cards players are holding. Therefore, we only need to consider state abstractions (such as hand-strength bucketing) and we don't have to deal with move abstractions.

By contrast, in trick-based card games such as Bridge, Spades, and Skat, legal moves are defined by the cards players hold. Therefore, abstracting such games with the intent of using pre-computed move probabilities is non-trivial. PIMC search deals with this problem by sampling game states in accordance with observed moves and private state information, revealing game states to all players, and then evaluating moves by using search algorithms tailored for the perfect information setting. The obvious drawback is that this method blatantly ignores players' ignorance and, consequently, for example has no concept of information gaining moves [4]. However, on the positive side, perfect information search algorithms such as alpha-beta search can be quite fast and, therefore, it may be possible to compensate for PIMC's shortcomings with regard to imperfect information game aspects by tactical performance.

In this paper we focus on PIMC search and show how it can be improved by using move evaluators that are based on actual game play rather than analyzing perfect information scenarios. We begin by formalizing PIMC search and discussing related work. We then proceed by introducing imperfect information Monte Carlo search and presenting performance evaluations using a synthetic game tree model and the game of Skat. Finally, we conclude the paper with a summary and suggestions for future research.

II. Related Work

PIMC search is one of the most widely used algorithms for imperfect information games. It was first proposed by Levy [5] in the context of Contract Bridge, and successfully implemented in Ginsberg's Bridge program GIB [6]. PIMC search has also been applied to other trick-taking card games such as Skat [2], [7], Hearts, and Spades [8]. To this day, the world's best Bridge and Skat programs are based on PIMC search. Given this success, the question arises why PIMC search performs so well even though it ignores seemingly essential imperfect information game-aspects. The first to study this question were Frank and Basin [4], who identified two types of problems PIMC search suffers from: strategy fusion (i.e., cherry-picking moves in nodes belonging to the same information set), and non-local dependencies between potentially distant nodes in the tree. Long et al. [9] present arguments why PIMC search is very successful in games in which, during the course of the game, more and more information is revealed (like in trick-based card games), but doesn't perform as well in other imperfect information games.

Another relevant part of AI literature deals with Monte Carlo Tree Search (MTCS) applied to imperfect information games, in which PIMC's top-level Monte Carlo search is replaced by UCT [10] variants acting on information sets, rather than game states [11], [12], [13], [14]. We will discuss how UCT is generalized to searching over information sets in the experimental section, where we compare it with our new recursive imperfect information Monte Carlo search method. Although in Skat, UCT was unable to defeat PIMC search

978-1-4673-5311-3/13/$31.00 ©2013 IEEE

```
PIMC (InfoSet I, int N)
for each m ∈ Moves (I) do
    let val[m] = 0
end
for each i ∈ {1..N} do
    let x = Sample(I);
    for each m ∈ Moves(I) do
        let val[m] += PerfInfoValue(x, m);
    end
end
return argmax_m {val[m]}
```

Algorithm 1: PIMC search pseudo-code.

[11], in Hearts it proved to be stronger than the previous best known computer Hearts players [12].

Lastly, Cazenave's nested Monte Carlo search for single-agent problems [15] is also highly relevant to this paper. One can think of the recursive IIMC algorithm presented here as a generalization of nested Monte Carlo search applied to imperfect information games with more than one player.

III. PIMC Search

The PIMC search algorithm presented below returns a move with the highest perfect information evaluation, given information set I and the number of samples N. An information set is a set of game tree nodes (sometimes also called *worlds*) the player to move in the current state of the game cannot distinguish, assuming we represent the game in extensive form. Function `Moves` returns the set of moves available to the player to move in a given information set I. Note that all move sets are identical across all nodes in I. Function `Sample` samples a node from I according to some state inference mechanism. This inference mechanism can take into account the game history, as known by the player to move. Function `PerfInfoValue` computes the perfect information value of making a move — again, with respect to the current player.

The strength of PIMC hinges on the quality of the state inference module used by `Sample` and the speed of `PerfInfoValue`. In the best case, when only one node is to be considered and all players share that knowledge, the game turns into a perfect information game, and thus, PIMC computes optimal moves. Furthermore, sampling more nodes improves move value estimates.

The shown basic algorithm can be improved in many ways: In a tournament setting, checking a fixed sample size N can be replaced by an anytime algorithm that continues sampling until a given time threshold is reached. Also, if `PerfInfoValue` is fast, PIMC can easily be parallelized by assigning sampling and node solving to different cores or computers in a network. In the case where computing perfect information values takes considerable time, one can resort to parallelizing this task as well, for instance by using an efficient parallel alpha-beta search algorithm. Finally, it seems to be wasteful to continue evaluating moves that are likely inferior, if so implied by the sampling history. Using importance sampling, applying UCB [16] at the root node, or utilizing budget allocation algorithms such as OCBA [17] may improve basic PIMC performance in a given domain.

```
IIMC (InfoSet I, int N, Player P)
for each m ∈ Moves (I) do
    let val[m] = 0
end
for each i ∈ {1..N} do
    let x = Sample(I);
    for each m ∈ Moves(I) do
        let v = FinishedGameValue(x, m, P);
        let val[m] += v;
    end
end
return argmax_m {val[m]}

FinishedGameValue (Node x, Move m, Player P)
let y = MakeMove(x, m);
while y is not a terminal node do
    let y = MakeMove(y, ComputeMove(P, I(y)));
end
return value of y in view of the player to move in x
```

Algorithm 2: Imperfect Information Monte Carlo search pseudo-code.

IV. Imperfect Information Monte Carlo Search

In general one may consider replacing PIMC's perfect information move evaluation with an arbitrary evaluation function. In the imperfect information Monte Carlo (IIMC) algorithm shown above, we pass on an additional parameter: a player module used to finish playing out (imperfect information) games. This playout occurs immediately after sampling a node and making the particular move we wish to evaluate. This computation is encapsulated in function `FinishedGameValue`. Note that the player module can range from random players, over rule-based systems, to quite sophisticated systems that execute PIMC or IIMC searches themselves. *Recursive IIMC* thus refers to an IIMC player using IIMC as a playout module, with the "recursion level" denoting the maximum recursive depth. For instance we may define `R0` as PIMC search, `R1` as IIMC calling PIMC, `R2` as IIMC calling `R1`, etc. In even more general settings, one could specify player modules as a collection of individual players, which then would allow us to bias games to either exploit observed move biases or cooperate with partners in multi-player settings. Another important point is that IIMC suffers less from strategy fusion compared to PIMC provided we do not leak game state information to the player in the course of finishing games.

The concept of running MCTS on information sets — which collect game states the player to move can't distinguish — is quite similar to IIMC [11], [13]. In this setting the usual nodes of the MCTS tree instead correspond to information sets, and playouts involve sampling a world at the root and then playing the MCTS move for previously observed states. Outside of the tree a fast rule-based agent or random player can be used to complete the game. One important difference to IIMC lies in the fact that running MCTS on information sets leaks game state information implicitly, for if, at the root, we sample worlds consistent with the current player's view, and play out those worlds according to information set constraints, the other players will never see actions inconsistent

with the true hand — their strategies will implicitly converge to "knowing" it. Moreover, the zero leakage "solution" of sampling inconsistent worlds is so detached from the real world as to be tactically hopeless for all but small endgames. By contrast, IIMC, at least at the top-level, does not leak game state information because the agents' playout policies do not adapt across playouts. Another difference is that IIMC can use its playout module's inference mechanism in `ComputeMove` at any node during the search (because game tree nodes imply their game histories), whereas MCTS players we are aware of either sample nodes only at the root or simply assume uniform distributions for hidden state variables.

As an example of IIMC, consider an IIMC player that uses PIMC player P as its player module. We call the resulting player "Recursive P" or RP for short. RP has two parameters: number N of top-level nodes it samples, and number M of nodes it samples in the process of finishing games. For any top-level move m and sampled node x, RP completes the game by repeatedly calling `PIMC` $(I(y), M)$, which computes a move in information set $I(y)$ by sampling worlds and evaluating moves using perfect information search. Naturally, RP will be slow, as its runtime is roughly $C \cdot N \cdot M$ at the beginning of the game, for a suitable constant C that estimates the average time it takes to solve positions along game trajectories. However, RP will suffer less from strategy fusion. So, it will be interesting to see for which choices of N and M RP can surpass P (which itself samples K nodes). Perhaps it is even possible to show that RP surpasses P's saturation point — the point at which increasing the number of sampled nodes K no longer increases P's performance significantly. In this case RP can benefit from hardware acceleration, but P cannot. We'll address these questions and more in the following sections.

V. IIMC Performance in Synthetic Trees

To investigate the asymptotic behaviour of the IIMC algorithm with PIMC as the base player, we used the generative tree model proposed by Long et al. [9]. This model defines a family of binary two-player zero-sum synthetic game trees parameterized by their *bias*, *correlation*, and *disambiguation* (BCD). Bias and correlation are properties defined in terms of the parents of the terminal nodes — they do *not* affect the correlation/bias of higher level tree branchings directly. Correlated leaves have the same values for their left and right actions. Non-correlated leaves have player-to-move left/right action values of $(1, -1)$ or $(-1, 1)$, chosen uniformly at random. Bias affects *only* correlated leaves, and assigns left/right values of $(1, 1)$ with probability $bias$, and $(-1, -1)$ otherwise. Disambiguation is a global property that affects how quickly an information set shrinks. A disambiguation value of $dis = 0$ results in maximum confusion, with information sets maintaining their full size throughout the game, while $dis = 1$ results in a perfect-information game.

We divided the BCD parameter space into a number of triplets. For each triplet we generated several thousand synthetic trees and computed the policies produced by all recursion levels of IIMC (each level consumes one ply of the game tree), with `R0` = PIMC, `R1` = IIMC(`R0`), etc. We also refer to `R1` as RecPIMC. At each information set we exhaustively sample all consistent worlds, assigning equal weight to each (i.e., uniform inference). Where the playout/PIMC evaluations are the same, we assign equal weight to each move, otherwise the policy deterministically selects the move with the highest evaluation. Because each recursive policy uses the evaluations of the previous iteration, we may efficiently compute each policy in time proportional to the game tree size. For consistency with Long et al. [9], we chose a tree height of 8, with 8×8 nodes at the root (a *chance* node where each player can distinguish only 8 information sets). Experiments with tree heights of 7 and 9 showed no substantial differences.

To measure the exploitability of each `R`* policy, we compute a corresponding best response player — this is simply the expectimax strategy, given a fixed opponent strategy. To eliminate any first-player bias, we evaluate with players in both positions and report the sum of the scores (thus the maximum possible exploitability is 4). Because the trees are sufficiently small and we have each player's policy, we compute the game theoretic score for each tree directly, rather than sampling playouts. In Fig. 1 we show the exploitability for `R0` and `R1` at various disambiguation levels. For low-disambiguation (poker-like) games, `R1` performs notably worse than `R0`. The precise reason for this is unclear, but we will discuss some possibilities.

First, consider the family of BCD trees with $dis = 0$ and $corr = 0$ ($bias$ is irrelevant when $corr = 0$). The leaf player always has one move which wins and one which loses. Thus, to an `R0` player, the leaf player always has a winning strategy (it "knows" the true world and can always guess correctly). However, since all paths appear to lead to wins (losses), `R0` views all moves equally, *except at the leaves*. At the leaves, the `R0` player chooses the move which wins most often on average (over the nodes in the current information set).

Now consider the `R1` player on the same BCD family. Because `R1` is performing playouts using `R0` as a playout module, it sees (at all nodes) that not all moves lead to wins. Thus the `R1` policy becomes biased towards certain regions of the tree. It may be that it is simply the presence of deterministic moves which is allowing it to be exploited. Forcing `R`* players to select an arbitrary (but fixed) move when both actions appear equal may shed light on the issue, since the number of deterministic actions would be more similar.

A second reason may be that the lack of an inference model is affecting our results. The nature of such an effect is non-obvious, but the possibility of its existence cannot be ignored. That said, any negative effects of using `R1` seem to disappear once disambiguation exceeds 0.3. Based on the measurements by Long et al. [9], disambiguation in trick-taking card games such as Hearts and Skat can be expected to lie somewhere in the region of 0.6, with some noise due to structural discrepancies between the generative model and the real games. Thus we expect to see significant reductions in exploitability for this class of games by adding a single level of recursion to existing PIMC agents.

To gain a fuller understanding of what each level of recursion provides, Fig. 2 shows the pairwise `R`* performance, and the exploitability of each player. The largest drop in exploitability comes from the first level of recursion, with minor improvements occurring until `R5`. This is promising, since going beyond `R1` or `R2` with current commodity hardware seems unlikely. There are clear signs that `R1` is exploiting `R0`, perhaps due to a good implicit opponent model, but as it is less exploitable itself, this seems a very fair tradeoff.

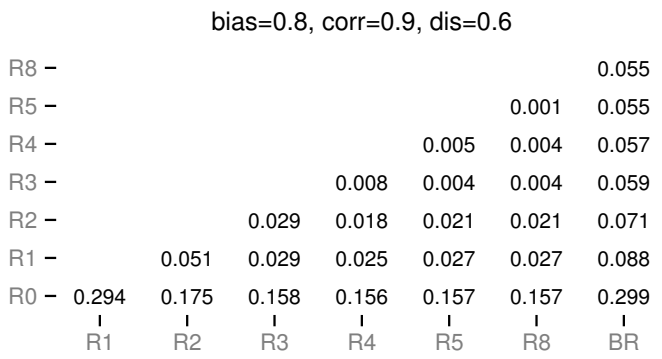

Fig. 2: Exploitability details for the BCD-tree parameter space resembling Hearts. Scores are with respect to the column player.

VI. IIMC Performance in Skat

In this section we conduct several experiments with the purpose of measuring the playing strength of PIMC-/IIMC-based players in a real game, under various world-sampling parameter settings. We start by giving a short introduction to the game of Skat — the application area we chose. We then discuss the Skat AI systems we used in our experiments, describe the experimental setup, and finally present and interpret the tournament results we obtained.

A. Skat

The following brief overview of Skat is based on the description given in [2]. We have omitted many of Skat's more detailed rules from this summary. For an in-depth description, we refer the reader to www.pagat.com/schafk/skat.html.

Skat is a popular trick-taking card game for three players. It is played by more than 40,000 players worldwide organized in the International Skat Players Association (www.ispaworld.org). Skat uses a short 32-card playing deck, which is similar to the standard 52-card deck, but without ranks 2 through 6. A hand begins with dealing 10 cards to each of the players, and the remaining two cards (the *Skat*) dealt face-down. Playing the hand consists of two phases: bidding and cardplay. In the bidding phase, players compete to become the *soloist*, a position analogous to the declarer in Bridge. Unlike Bridge, there are no permanent alliances of players between hands, but the two players who lose the bidding will become partners for the remainder of the current hand. The soloist will then usually pick up the Skat and discard any two cards face-down back to the table. The soloist then announces the *game type*, which will determine the trump suit and how many points the soloist will earn if she wins the game. There is one game type for each of the four suits (♢♡♠♣), in which the named suit and the four jacks form the trump suit. These four types are referred to as *suit games*. Other game types include *grand*, in which *only* the four jacks are trump, and *null*, in which there is no trump and the soloist attempts to lose every trick. Cardplay begins after the soloist's game announcement, and consists of 10 tricks, which are played in a manner similar to Bridge and other trick-taking card games. The soloist's objective is to take 61 or more of the 120 available *card points*.

From an AI perspective, Skat has many challenging aspects. In the bidding phase it constitutes a full-fledged three-player game that turns into a one-versus-two team game in the cardplay phase, in which coordination among the defenders is essential. Moreover, Skat is not even a zero-sum game, which complicates computing Nash equilibrium strategy profiles even further. Unlike many other popular card games, Skat contains hidden information not only from chance moves, but also from

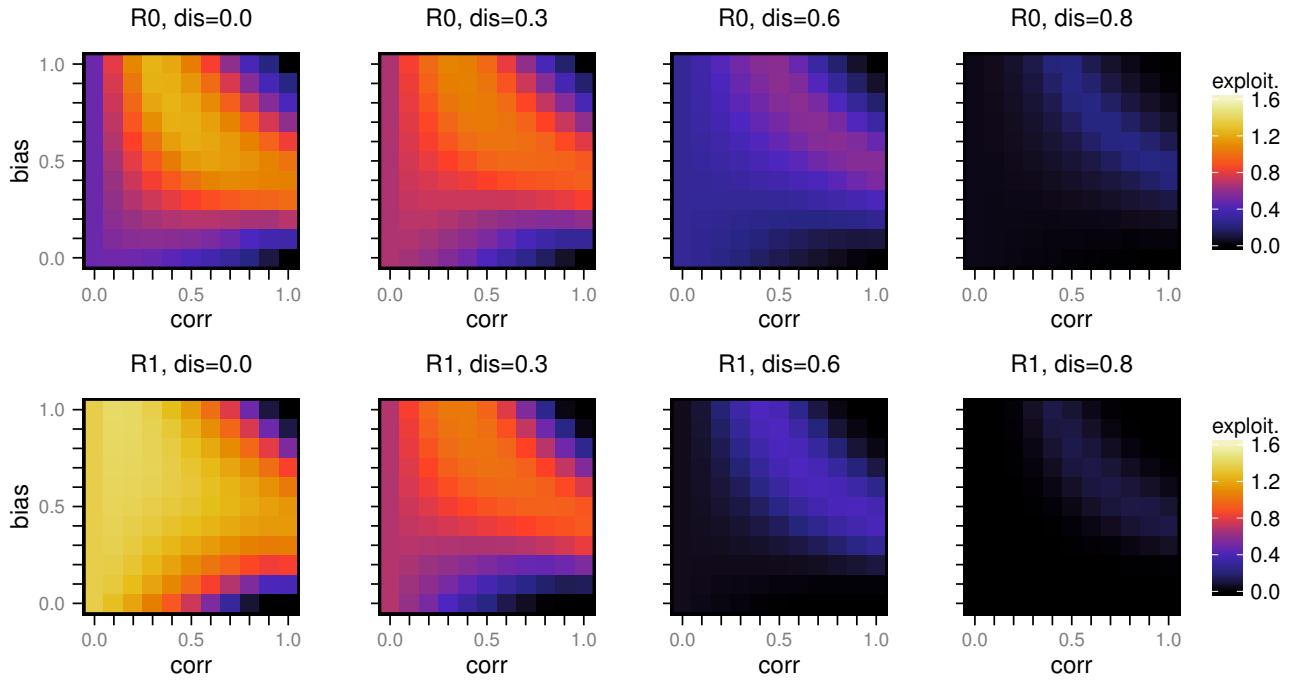

Fig. 1: Exploitability of PIMC (R0) and RecPIMC (R1) on synthetic trees with varying levels of disambiguation.

player moves in form of the soloist's two-card discard. Lastly, as with many imperfect information games, there is a need for players to infer state properties, which creates opportunities for signalling and information sharing that would not exist in a two-player game.

B. Skat AI Systems

Skat program Kermit has reached expert playing strength in 2008 [2] by combining fast PIMC search with card inference and heuristic state evaluation whose parameters were fitted using millions of Skat games played by human players. Kermit's card inference is based on tracking void suits and learned histograms for important hand features such as suit length and high-card distributions given players' maximum bids. This inference module is used to sample information set nodes which are then passed on to a fast perfect information solver. To speed up solving, pre-computed trick-6 and trick-7 endgame tables (partial and full) are used, as well as ProbCut forward pruning, payoff-similarity, and a fastest-cut-first heuristic [18], [19]. With these improvements Kermit's suit game solver is capable of computing exact cardpoint scores for all legal moves at a rate of approximately 68 worlds per second (at the start of the cardplay phase — 30 cards to play) on a single Intel i7 core running at 4 GHz. With forward pruning this number increases to 237 all-move approximate evaluations per second (K exact evaluations are playing-strength equivalent to K approximate evaluations).

The second Skat AI system we use in our experiments is XSkat. XSkat is a free software Skat program written by Gunter Gerhardt (www.xskat.de). It is essentially a rule-based system that does not perform search at all and plays very quickly, which makes it well suited for our recursive PIMC search framework. The third and last Skat program we consider here is Bernie, which runs the UCT MTCS algorithm on information sets, choosing consistent worlds uniformly at random and using XSkat as playout module [11].

C. Experimental Setup

We measured the cardplay strength of the different players using a common set of 18,962 games produced by 3 Kermits bidding, discarding, and declaring. There were initially 20,000 deals, but some led to all players passing during bidding. These passed games are not included in the scores presented. All games were scored using the widely used Fabian-Seeger tournament scoring method.

When sampling consistent worlds from an information set, Kermit's inference module takes the bidding history and game declaration (but not cardplay) into account to bias worlds. Our recursive players use this inference module when sampling worlds at the top level.

Cardplay matches were run on heterogeneous hardware, with differences in speed not affecting our results. When we list timing information it is with respect to an Intel i7 core running at 4 GHz. In the interest of speed we reuse a player's move selections across matches when encountering a previously seen information set. Correctness is not affected, as inference does not consider the names of the other players. Moreover, we expect that, by correlating games in this manner, we increase the statistical significance when varying one player.

TABLE I: Recursive XSkat (RXSkat, with and without world inference) versus UCT on information sets (Bernie). RXSkat samples 160 worlds, while Bernie uses 1,600 playouts. The confidence interval is two standard deviations. The results indicate that RXSkat is considerably stronger than Bernie.

Soloist	Defenders	Soloist Score
Bernie	$\text{RXSkat}_{\text{infer}=0}$	61.34 ± 1.30
$\text{RXSkat}_{\text{infer}=0}$	Bernie	74.14 ± 1.12
Bernie	$\text{RXSkat}_{\text{infer}=1}$	53.99 ± 1.97
$\text{RXSkat}_{\text{infer}=1}$	Bernie	74.37 ± 1.58

D. Information Set UCT vs. RecPIMC

In this subsection we compare the playing strengths of information set UCT player Bernie and IIMC player RXSkat which both use the same fast player module — XSkat.

Our recursive XSkat agent (RXSkat) samples consistent worlds (either with or without Kermit world inference), and uses XSkat completions to evaluate the effect of playing each possible move. We gave Bernie a budget of 1600 playouts and RXSkat a budget of 160 worlds (with up to 10 legal moves in each world). Thus, Bernie always uses at least as many XSkat playouts, although starting from varying depths. RXSkat is on the order of 35 times faster than Bernie, although presumably the UCT implementation could be improved. XSkat is fast enough that playing entire hands with three RXSkat players takes on the order of 1 second, using one CPU core.

Bernie's lack of informed world inference puts it at a disadvantage compared to RXSkat which uses Kermit's inference module — especially considering that the model of the bidding matches the actual process used for generating hands. Thus, we include results when RXSkat's inference module is disabled and replaced with uniform random as well.

To examine the difference between Bernie and RXSkat we use pair games, where copies of both agents take the roles of soloist and defender, playing each deal from both perspectives. Given Skat's three player nature, it is non-trivial to extrapolate tournament scores from pair matches.

Assuming equivalent bidding, we expect a single player to take the role of soloist one third of the time, or 12 hands in a 36-hand tournament list. Thus, an increase of 10 tournament points as soloist corresponds to 120 list points. However, stronger soloist play also has the effect of decreasing the number of hands successfully defended, effectively reducing the expected score of the other players.

We show in Table I the expected declarer points for each matchup. Not shown is the change in expected defender tournament points, which goes from 7.05 to 4.81 in favour of $\text{RXSkat}_{\text{infer}=0}$ versus Bernie. Over 24 hands (as defender), this corresponds to RXSkat gaining 53.76 list points from defender wins alone. Combined with an expected gain of $12.8 \times 12 = 153.6$ declarer points, RXSkat can acquire over 200 points more than Bernie in a list, due to improved cardplay. A 200 point difference in a 36-deal list is roughly what separates World Championship calibre players from average club players. Recursive XSkat is therefore considerably stronger than Bernie, which is based on information set UCT.

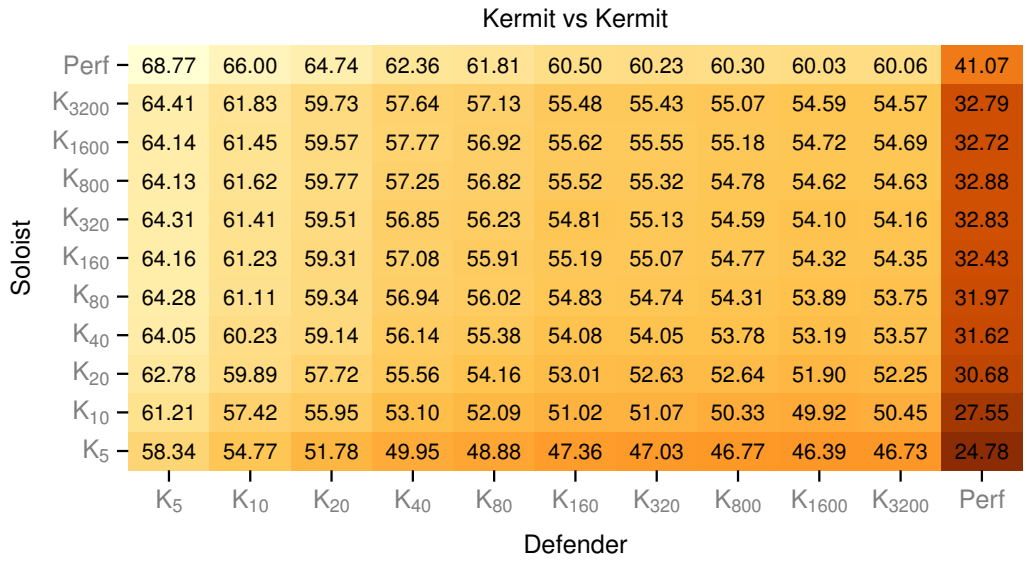

Fig. 3: Kermit vs. Kermit games showing the diminishing effects of sampling more worlds as declarer and as defender. Perf denotes the "perfect information player" who knows the state of the game. Table entries are soloist scores. 95% confidence intervals for each entry are in the range of ± 1.2 to ± 1.6.

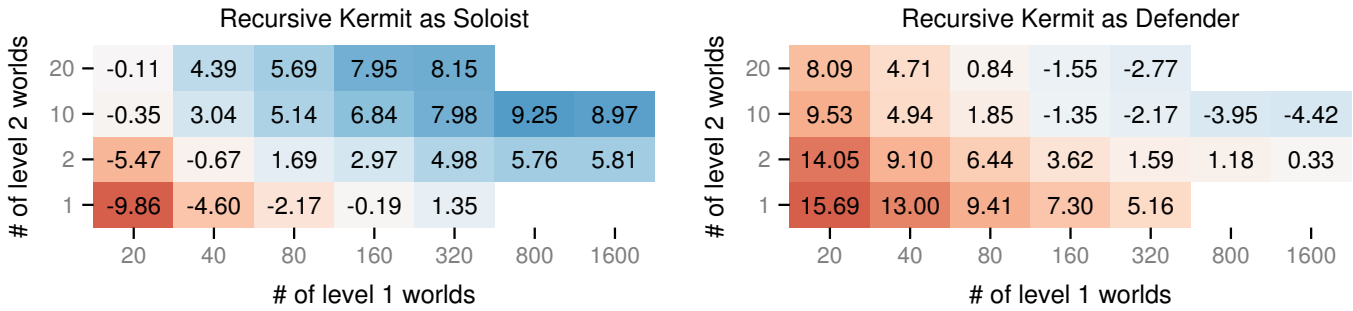

Fig. 4: Recursive Kermit with varying numbers of first- and second-level worlds vs. Kermit using 160 worlds. Soloist scores are shown, relative to a baseline Kermit vs. Kermit score of 55.19. One standard deviation is about 0.5 points for all entries. The same 18,962 games were used for each data point. The (1600,10) entries indicate that Recursive Kermit is considerably stronger than PIMC-based Kermit at its saturation point. Lower scores are better when acting as a defender.

E. RecPIMC vs. PIMC

As seen in Fig. 3, increasing the number of worlds sampled by PIMC player Kermit beyond 160 does not significantly improve playing strength with respect to either other Kermits nor a perfect-information player (which cheats by observing the true world).

To overcome this plateau, we can observe in Fig. 4 the effects of applying RecPIMC (a.k.a. `R1`) with Kermit playout modules. Playing against Kermit, both as soloist and defender, recursive Kermit achieves significantly stronger performance. For instance, $\mathrm{RK}_{160,10}$ with 160 level-1 worlds, and 10 level-2 worlds gets about 7 points more per game than Kermit, when playing as soloist, which roughly corresponds to an 80 point gain in a list — a substantial performance increase, given that Kermit at the PIMC saturation point is currently the strongest Skat program playing at human expert level [2].

We estimate that this playing strength increase makes $\mathrm{RK}_{160,10}$ competitive with the strongest human players, but further experiments have to be conducted with other opponents to see exactly how much of the performance gain can be attributed to having a good opponent model. Before arranging human-machine matches, which need to consist of hundreds of games to generate statistically significant results, we have to create a distributed version of $\mathrm{RK}_{160,10}$ to offset its roughly 8-fold slowdown compared to Kermit — a PIMC-based player considering 160 worlds. Running $\mathrm{RK}_{160,10}$ on 32 Intel x86 cores will suffice to let it play under tournament conditions ($\approx$ 3 seconds per move on average). As indicated by the (1600,10) entries in Fig. 4, increasing the hardware speed by another factor of 10 leads to an additional playing strength gain of roughly 2 points per game versus Kermit. This is exciting, because unlike PIMC-based Kermit, which doesn't benefit from further hardware acceleration, we anticipate being able to run $\mathrm{RK}_{1600,10}$ under tournament conditions in the near future. As $\mathrm{RK}_{160,10}$ is almost suitable for real-time usage, we will use it when discussing recursive Kermit unless otherwise stated.

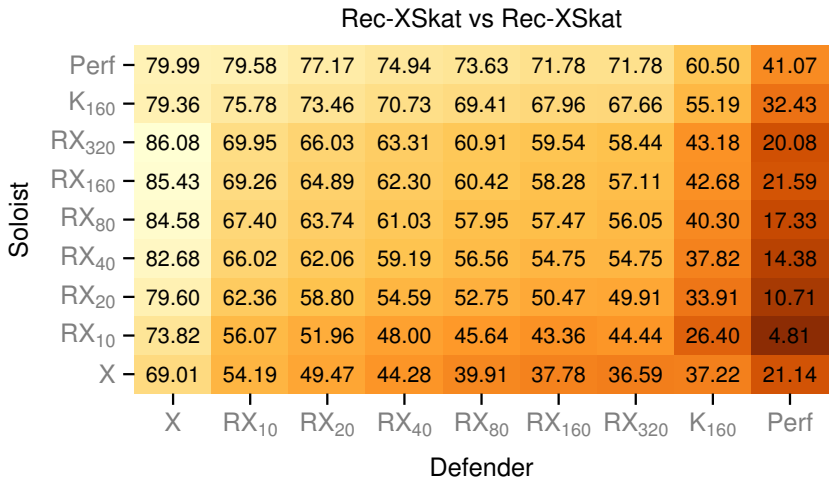

Fig. 5: Declarer scores of pairwise matches between Recursive XSkat using Kermit inference with varying numbers of top-level worlds, XSkat, and Kermit with 160 worlds. 95% confidence intervals for each entry are in the range of ± 0.9 to ± 1.6.

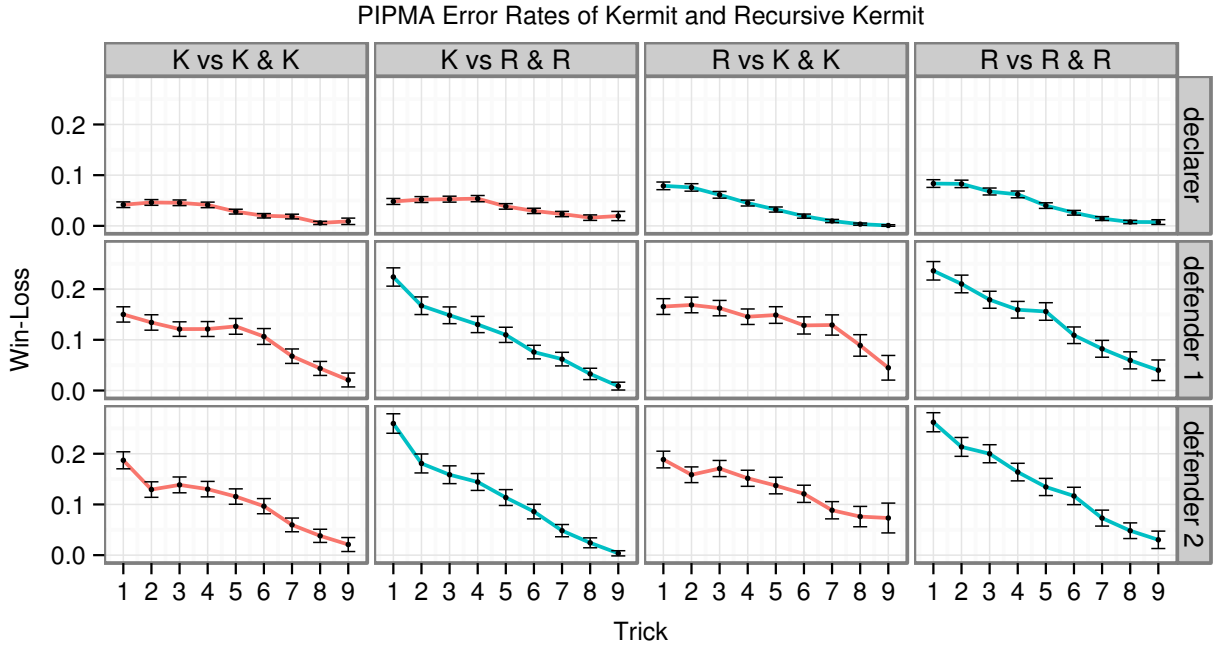

Fig. 6: PIPMA error rates of Kermit and Recursive Kermit in all roles, measuring the frequency of perfect-information game-losing mistakes. The confidence intervals indicate two standard deviations. The declarer sits in front of defender 1.

F. Implicit Opponent Modelling Effects

We can see in Fig. 5 evidence that a non-trivial portion of RXSkat's strength derives from accurately modelling its opponents. Observe that RX_{320} as soloist performs substantially better than Kermit, but only when XSkat is the defender. In the presence of stronger defense, the situation reverses, with Kermit outperforming RXSkat. However, there is a general trend that RXSkat is stronger than XSkat, for sufficiently many top-level worlds. As evidence that the gains of RK are not all due to modelling we present data (not plotted) for $RK_{160,10}$ playing against two XSkat defenders: whereas Kermit and Perfect had declarer scores of 80.7 and 80.9 respectively, RK had a score of 83.8. Since Kermit is a poor model for XSkat, we believe that the gains against XSkat were due to increased imperfect information performance, rather than exploiting a good opponent model. This effect is also visible in Fig. 5: against two K160 defenders, RXSkat with a small number of worlds is weaker than XSkat, but then catches up and eventually outperforms XSkat when using 80 worlds or more. When defending against K160, all considered RXSkat versions are stronger than XSkat.

G. Perfect Information Post Mortem Analysis

Fig. 6 shows the perfect information post mortem analysis (PIPMA, [20]) error rates for Kermit vs. Recursive Kermit tournaments. Errors are defined as moves that go from a winning open-handed position to a losing one. Positions where no errors are possible do not affect the error rates. A general

trend of making fewer perfect information mistakes towards the end of the game is clearly visible. What is interesting is that defenders' error rates are higher when recursive Kermit is the soloist. This indicates that RecPIMC based players may be able to increase confusion on the defender side. Another interesting observation is that although recursive Kermit seems to have higher error rates during the first half of the game, it is still winning against Kermit overall. Whether this is an artifact of PIPMA or indicating that recursive Kermit is trading "optimality" for creating confusion will be the subject of future work.

VII. Conclusion and Future Work

In this paper we have introduced imperfect information Monte Carlo search which can combine the tactical strength of perfect information move evaluation with superior modelling of imperfect information aspects. As a result IIMC search players are less prone to strategy fusion effects, compared to PIMC search, and they are also capable of increasing opponents' confusion by accurately anticipating their reaction to played moves. IIMC also appears to significantly outperform information set UCT on the domains we examined.

The results we obtained for RecPIMC, a IIMC variant using perfect information playouts, on synthetic games suggest that its performance is correlated with the game's disambiguation factor: the more information is revealed in course of the game, the better RecPIMC performs compared to PIMC. This corresponds to a broad class of games, notably trick-based card games, where we expect RecPIMC to perform well.

The results of the extensive tests we conducted suggest that in our domain — computer Skat — the playing strength gain achieved by using RecPIMC search can be substantial and that it in fact surpasses the strength of regular PIMC search at its saturation point. This is good news for future developments because with faster multi-core processors on the horizon, RecPIMC in Skat will likely become fast enough to defeat PIMC based players under tournament conditions within a few years time.

RecPIMC search has the potential to push the state-of-the-art in trick-based card game AI even further. For example, one can imagine that playing better in Skat's cardplay phase can be compounded with more aggressive bidding to obtain even higher scores, because stronger card players can get away with weaker hands more often. Also, RecPIMC search opens the door to actively deceiving opponents and cooperating with partners beyond what simple state inference can accomplish, simply by gauging the merit of every move in an imperfect information setting. Lastly, according to Fig. 6, non-recursive Kermit has a lower PIPMA error rate than recursive Kermit in the early game. This suggests the creation of a hybrid player that uses recursive evaluations only in the late game. However, the PIPMA rate is only a proxy for measuring how exploitable an agent is by a true best-response player — one that is constrained by imperfect information. It is possible that by playing "losing" moves, recursive Kermit is gaining important information about the hidden distribution, and/or "confusing" the defenders by directing play to positions where they are likely to make mistakes.

We have shown that recursive Kermit is significantly stronger than Kermit against a range of opponents, but it remains to be seen how well this technique generalizes to other domains where PIMC search performs well, such as Hearts or Bridge. The promising results we obtained for synthetic games with high disambiguation factors make us confident that this will indeed be the case.

Acknowledgments

The authors would like to acknowledge GRAND NCE and NSERC for their financial support.

References

[1] M. Johanson, "Robust strategies and counter-strategies: building a champion level computer Poker player," Master's thesis, University of Alberta, Canada, 2007.

[2] M. Buro, J. Long, T. Furtak, and N. R. Sturtevant, "Improving state evaluation, inference, and search in trick-based card games," in *Proceedings of IJCAI*, 2009, pp. 1407–1413.

[3] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," *Proceedings of NIPS*, vol. 20, pp. 1729–1736, 2008.

[4] I. Frank and D. Basin, "Search in games with incomplete information: a case study using Bridge card play," *Artificial Intelligence*, vol. 100, pp. 87–123, 1998.

[5] D. Levy, "The million pound Bridge program," in *Heuristic programming in Artificial Intelligence*. Asilomar, CA: Ellis Horwood, 1989.

[6] M. Ginsberg, "GIB: imperfect information in a computationally challenging game," *Journal of Artificial Intelligence Research*, vol. 14, pp. 303–358, 2001.

[7] S. Kupferschmid and M. Helmert, "A Skat player based on Monte-Carlo simulation," in *Proceedings of the 5th International Conference on Computers and Games*, 2006, pp. 135–147.

[8] N. Sturtevant and A. White, "Feature construction for reinforcement learning in Hearts," in *Proceedings of the 5th International Conference on Computers and Games*, 2006, pp. 122–134.

[9] J. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the success of perfect information Monte Carlo sampling in game tree search," in *Proceedings of AAAI*, 2010, pp. 134–140.

[10] L. Kocsis and C. Szepesvari, "Bandit based Monte-Carlo planning," in *Proceedings of the European Conference on Machine Learning*, 2006, pp. 282–293.

[11] J. Schäfer, "The UCT algorithm applied to games with imperfect information," Master's thesis, University of Magdeburg, Germany, October 2007.

[12] N. R. Sturtevant, "An analysis of UCT in multi-player games," *ICGA Journal*, vol. 31, no. 4, pp. 195–208, 2008.

[13] P. Cowling, E. Powley, and D. Whitehouse, "Information set Monte Carlo tree search," *IEEE Transactions on Computational Intelligence and AI for Games*, vol. 4, pp. 120–143, 2012.

[14] P. Cowling, C. Ward, and E. Powley, "Ensemble determinization in Monte Carlo tree search for the imperfect information card game magic: The gathering," *IEEE Transactions on Computational Intelligence and AI for Games*, vol. 4, pp. 241–257, 2012.

[15] T. Cazenave, "Nested Monte-Carlo search," in *IJCAI*, C. Boutilier, Ed., 2009, pp. 456–461.

[16] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[17] C. Chen, *Stochastic Simulation Optimization: An Optimal Computing Budget Allocation*, ser. Stochastic Simulation Optimization. World Scientific Publishing Company, Incorporated, 2010. [Online]. Available: http://books.google.ca/books?id=Hgx6g2YdMH0C

[18] T. Furtak and M. Buro, "Minimum proof graphs and fastest-cut-first search heuristics," in *Proceedings of IJCAI*, 2009, pp. 492–498.

[19] ——, "Using payoff-similarity to speed up search," in *Proceedings of IJCAI*, 2011, pp. 534–539.

[20] J. R. Long and M. Buro, "Real-time opponent modelling in trick-taking card games," in *Proceedings of IJCAI*, 2011, pp. 617–622.