
Article:
Cowling, Peter I. orcid.org/0000-0003-1310-6683, Powley, Edward J. and Whitehouse, Daniel (2012) Information Set Monte Carlo Tree Search. Computational Intelligence and AI in Games, IEEE Transactions on. 6203567. pp. 120-143. ISSN 1943-068X

https://doi.org/10.1109/TCIAIG.2012.2200894

Information Set Monte Carlo Tree Search

Peter I. Cowling, *Member, IEEE*, Edward J. Powley, *Member, IEEE*, and Daniel Whitehouse, *Student Member, IEEE*

***Abstract*—Monte Carlo tree search (MCTS) is an AI technique that has been successfully applied to many deterministic games of perfect information. This paper investigates the application of MCTS methods to games with hidden information and uncertainty. In particular, three new information set MCTS (ISMCTS) algorithms are presented which handle different sources of hidden information and uncertainty in games. Instead of searching minimax trees of game states, the ISMCTS algorithms search trees of information sets, more directly analyzing the true structure of the game. These algorithms are tested in three domains with different characteristics, and it is demonstrated that our new algorithms outperform existing approaches to handling hidden information and uncertainty in games.**

***Index Terms*—Artificial intelligence (AI), game tree search, hidden information, Monte Carlo methods, Monte Carlo tree search (MCTS), uncertainty.**

I. Introduction

MONTE CARLO TREE SEARCH (MCTS) methods have gained popularity in recent years due to success in domains such as *Computer Go* [1]. In particular, the upper confidence bound for trees (UCT) algorithm [2] forms the basis of successful MCTS applications across a wide variety of domains. Many of the domains in which MCTS has proved successful, including *Go*, were considered challenging for the application of traditional AI techniques (such as minimax search with α–β pruning), particularly due to the difficulty of forecasting the winner from a nonterminal game state. MCTS has several strengths. It requires little domain knowledge, although including domain knowledge can be beneficial [1]. It is an anytime algorithm, able to produce good results with as much or as little computational time as is available. It also lends itself to parallel execution.

This paper investigates the application of MCTS to games of imperfect information. In particular, we consider games which have as elements three different types of imperfect information.

- *Information sets* are collections of states, which appear in a game when one player has knowledge about the state that another player does not. For example, in a card game each player hides his own cards from his opponents. In this example, the information set contains all states which correspond to all possible permutations of opponent cards. A player knows which information set they are in, but not which state within that information set.
- *Partially observable moves* appear in games where a player performs an action but some detail of the action is hidden from an opponent.
- *Simultaneous moves* arise when multiple players reveal decisions simultaneously without knowing what decision the other players have made. The effect of the decisions is resolved simultaneously. The well-known game of *Rock-Paper-Scissors* is an example of this.

Hidden information poses many challenges from an AI point of view. In many games, the number of states within an information set can be large: for example, there are $52! \approx 8 \times 10^{67}$ possible orderings of a standard deck of cards, each of which may have a corresponding state in the initial information set of a card game. If states are represented in a game tree, this leads to a combinatorial explosion in branching factor. Furthermore, players may be able to infer information about an opponent's hidden information from the actions they make, and in turn may be able to mislead (bluff) their opponents into making incorrect inferences. This leads to an increased complexity in decision making and opponent modeling compared to games of perfect information.

The majority of existing work on MCTS in particular, and game AI in general, focuses on games of perfect information. However, there are several existing methods of applying MCTS to games with hidden information. One popular approach is *determinization* (of hidden information) which has been successful in games such as *Bridge* [3] and *Klondike Solitaire* [4]. The determinization technique makes decisions by sampling states from an information set and analyzing the corresponding games of perfect information. However, determinization has several known weaknesses. One weakness is that the computational budget must be shared between the sampled perfect information games. The trees for these games will often have many nodes in common, but the determinization approach does not exploit this and the effort of searching those nodes is duplicated. Another weakness is known as *strategy fusion* [5]: since different states in the same information set correspond to distinct tree nodes, the search effectively makes the erroneous assumption that different decisions can be made from these states. These are not the only weaknesses with determinization, but they are the main ones addressed in this paper.

We introduce a family of algorithms, information set Monte Carlo tree search (ISMCTS), that overcome some of the weaknesses of the determinization approach. Instead of searching the minimax tree produced by a determinization, we construct a tree where the nodes represent information sets rather than states. This offers the advantage that statistics about moves are collected together in one tree, thus using the computational budget more efficiently, whereas determinization constructs

Manuscript received November 09, 2011; revised February 10, 2012; accepted May 15, 2012. Date of publication May 22, 2012; date of current version June 12, 2012. This work was supported by the U.K. Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/H049061/1.

The authors are with the Artificial Intelligence Research Centre, School of Computing, Informatics and Media, University of Bradford, Bradford BD7 1DP, U.K. and also with the Department of Computer Science, University of York, York YO10 5GH, U.K. (e-mail: peter.cowling@york.ac.uk; e.powley@bradford.ac.uk; d.whitehouse1@student.bradford.ac.uk).

Digital Object Identifier 10.1109/TCIAIG.2012.2200894

several trees and does not share any information between them. Furthermore, this approach offers an improved model of the decision-making process compared to determinization, since the search is able to exploit moves that are good in many states in the information set and the effects of strategy fusion can be lessened or eliminated entirely.

The benefit of ISMCTS is investigated in three domains.

- *Lord of the Rings: The Confrontation* [6] is a board game with elements similar to *Stratego* [7] and has several features which make it even more challenging from an AI perspective. It has hidden information, partially observable moves, and simultaneous moves, all of which make the decision making process highly complex. The game also has an asymmetry between the two players since they have different win conditions and different resources available to them, which necessitates different tactics and strategies.
- m, n, k-*games* [8] are a generalization of games such as *Noughts and Crosses* and *Renju* where players try to place k pieces in a row on an $m \times n$ grid. We will investigate the *phantom* 4, 4, 4-*game* where players cannot see each other's pieces. This leads to a game with hidden information and partially observable moves.
- We extend our previous work [9], [10] on the popular Chinese card game *Dou Di Zhu* [11], a game with hidden information.

The relative performance of ISMCTS versus determinized UCT varies across these three domains. In *Lord of the Rings: The Confrontation* a deep search is required for strong play, and so ISMCTS has the advantage due to its more efficient use of the computational budget. In the *Phantom (4, 4, 4)* game the effects of strategy fusion are particularly detrimental, so again ISMCTS outperforms determinization. However, in *Dou Di Zhu*, neither of these effects has a particularly large impact on playing strength and the information set tree has an orders-of-magnitude larger branching factor than a determinization tree, so the two algorithms are on a par. One advantage of the ISMCTS approach over determinized UCT is that it finds better quality strategies, which are less susceptible to the effects of strategy fusion and more accurately model the decision making process of the players.

In games such as *Poker*, computation of accurate prior belief distributions through inference and opponent modeling is one of the keys to strong play [12]. However, in many games, accurate belief distributions are less important as it is possible to find strategies that are robust to most or all possibilities for the hidden information. We have observed this previously for a simplified version of *Dou Di Zhu* [10], where a perfect opponent model leads to only a small increase in performance over uniform belief distributions. This suggests that situations in which useful hidden information can be inferred from our opponent's actions are rare. Our intuition is that the other games studied in this paper are also of this type: when we as humans play the games, situations where we infer hidden information based on opponent actions are much rarer than situations where we aim for robustness in the face of uncertainty. Thus, we do not consider belief distributions in this paper and instead assume uniform distributions over the states in information sets.

In many games of imperfect information, all pure policies are dominated and thus a strong player must find a mixed policy. *Rock-Paper-Scissors* and *Poker* are two examples where this is clearly the case, where mixing strategies is important to achieving a strong level of play. The algorithms described in this paper are not designed explicitly to seek mixed policies but they often do so anyway, in the sense of choosing different actions when presented with the same state. This arises from the random nature of Monte Carlo simulation: the MCTS algorithm is not deterministic. Shafiei *et al.* [13] demonstrate that the UCT algorithm finds a mixed policy for *Rock-Paper-Scissors*, and our preliminary investigations suggest that ISMCTS finds mixed policies for the small, solved game of *Kuhn Poker* [14]. However, MCTS often fails to find optimal (Nash) policies for games of imperfect information. Ponsen *et al.* [15] suggest that algorithms such as Monte Carlo counterfactual regret (MCCFR) [16] are a better fit for approximating Nash equilibria in games whose trees contain millions of nodes, whereas the strength of an MCTS approach lies in finding a strong suboptimal policy but finding it in reasonable computation time for complex games with combinatorially large trees.

The paper is structured as follows. Section II defines the notation used to describe a game and formally defines the three different types of imperfect information. Section III presents relevant background material, including an introduction to MCTS (Section III-A), a review of existing approaches for games of imperfect information (Section III-B), and remarks on handling simultaneous moves and chance nodes in tree search (Section III-C). Section IV describes in detail the algorithms presented in this paper, including the novel ISMCTS algorithms. Section V describes *Lord of the Rings: The Confrontation* and presents experimental results. Section VI discusses results for the *Phantom (4, 4, 4)* game. Section VII describes the game *Dou Di Zhu* and discusses some of the challenges this domain presents. Section IX ties together the results from the three preceding sections and offers concluding remarks. Finally, ideas for future work are presented in Section X.

II. Definitions and Notation

This section introduces the terminology and associated notation that we use throughout this paper. The notation is our own, but more detail on the concepts can be found in a standard textbook on game theory, e.g., [17].

A *game* is defined on a directed graph (S, Λ). The nodes in S are called *states* of the game; the leaf nodes are called *terminal states*, and the other nodes *nonterminal states*. A game has a positive number κ of players, numbered $1, 2, \ldots, \kappa$. There is also an *environment player* numbered 0. Each state s has associated with it a number $\rho(s) \in \{0, \ldots, \kappa\}$, the *player about to act*. Each terminal state s_T has associated with it a vector $\mu(s_T) \in \mathbb{R}^\kappa$, the *reward vector*.

The game is played as follows. At time $t = 0$ the game begins in the *initial state* s_0. At time $t = 0, 1, 2, \ldots$, if state s_t is nonterminal, player $\rho(s_t)$ chooses an edge $(s_t, s_{t+1}) \in \Lambda$ and the game transitions through that edge to state s_{t+1}. This continues until a time $t = T$ when s_T is terminal. At this point, each player receives a reward equal to the relevant entry in the vector $\mu(s_T)$, and the game ends.

Players typically do not choose edges directly, but choose *actions*. Actions are equivalence classes of edges, with the restriction that two edges starting from the same node cannot be in the same action. We also require that all edges in an action have the same player about to act in their start nodes. Actions capture the notion that different edges from different states can be in some sense equivalent: for example, an action may consist of all edges leading from a state where player 1 holds an ace (and some other cards) to the state where player 1 has just played the ace; such an action would be labeled "player 1 plays an ace." The set of actions from a state s is denoted $A(s)$; this is simply the set of action classes restricted to edges outgoing from s.

The *transition function* f maps a (state, action) pair (s_t, a) to a resulting state s_{t+1}, by choosing an edge $(s_t, s_{t+1}) \in a$. Note that the domain of f does not include all (state, action) pairs, as not all actions are available in each state.

A *policy* for player i maps each state s with $\rho(s) = i$ to a probability distribution over $A(s)$. This distribution specifies how likely the player is to choose each action from that state. One way of stating the fundamental problem of game AI (and indeed of game theory) is as finding the policy that leads to the highest expected reward, given that all other players are trying to do the same. The exception here is the environment player, whose policy is fixed as part of the game definition and specifies the probabilities of outcomes for chance events.

This paper studies games of *imperfect information*. In these games, each player partitions the state set S into *information sets*. Note that, in general, each player's partitioning is different. See Fig. 3 for example. The players do not observe the actual state of the game, but rather the information set containing the actual state. Essentially, the states in a player's information set are indistinguishable from that player's point of view. In particular, this means that the player's choices of actions must be predicated on information sets, not on states.

This paper also studies games with *partially observable moves*. Here each player further partitions the actions into *moves*. We require that the partitions for a player's own actions are singletons, i.e., that players can fully observe their own moves. When a player plays an action, the other players do not observe that action directly but observe the move to which the action belongs. The set of moves from player i's point of view from a state s is denoted $M_i(s)$, and is the set of move classes restricted to edges outgoing from s. In the case where $\rho(s) = i$, we have $M_{\rho(s)}(s) = A(s)$.

We make the restriction that two edges leading from states in a player i information set, and contained within the same move from player i's point of view, must lead to states in the same player i information set. In other words, all available information about the game can be gathered by observing moves, without the need to directly observe information sets. This also allows a transition function on (information set, move) pairs to be well defined. This property can be achieved if necessary by addition of *environment states*, in which the environment player has exactly one action available; this action may depend upon (and thus provide an observation of) the actual state, but different states in the same information set from the point of view of some other player may have different actions available. This is exemplified in Section V-A4.

In summary, we have the following definitions.

Definition 1: A *game of imperfect information* is a 9-tuple

$$\Gamma = (S, \Lambda, s_0, \kappa, \mu, \rho, \pi_0, (\sim_1, \ldots, \sim_\kappa), (\smile_1, \ldots, \smile_\kappa)) \quad (1)$$

where (S, Λ) is a finite nonempty directed graph, with S the set of *states* and Λ the set of *state transitions*; $s_0 \in S$ is the *initial state*; $\kappa \in \mathbb{N}$ is the *number of players*; $\mu : S_T \to \mathbb{R}^\kappa$ is the *utility function*, where $S_T \subseteq S$ is the set of leaf nodes (*terminal states*); $\rho : S \to \{0, 1, \ldots, \kappa\}$ is the *player about to act* in each state; $\pi_0 : \Lambda_0 \to [0, 1]$, where for all $r \in S$ with $\rho(r) = 0$ we have $\sum_{s:(r,s)\in\Lambda} \pi_0(r, s) = 1$, is the *environment policy*; $\sim_i$ is an equivalence relation on S, whose classes are player i's *information sets*; $\smile_i$ is an equivalence relation on Λ, whose classes are *moves* as observed by player i. Classes consisting of edges (s, u) with $\rho(s) = i$ are also known as player i's *actions*.

Definition 2: Consider a game Γ, a state s, and a player i. The set of *legal moves* from s from player i's point of view is

$$M_i(s) = \{[(s, u)]^{\smile_i} : (s, u) \in \Lambda\}. \quad (2)$$

The set of *legal actions* from s is

$$A(s) = M_{\rho(s)}(s) \quad (3)$$

i.e., the set of legal moves from the point of view of the player about to act.

Definition 3: Consider a game Γ. Let $B = \{(s, a) : s \in S, a \in A(s)\}$, the set of all pairs of states and their legal actions. The *transition function* for Γ is the function $f : B \to S$ such that $(s, s') \in a$ implies $f(s, a) = s'$. In other words, $f(s, a) = s'$ means that the single edge in a that starts from s ends at s'. (There is exactly one edge in a starting from s, so this is well defined.)

The transition function is extended to a function from (information set, move) pairs to information sets, where all observations are from the point of view of the same player, in the natural way.

III. Background

A. Monte Carlo Tree Search

MCTS is a class of game tree search algorithms that make use of simulated games to evaluate nonterminal states. Simulated games select random actions until a terminal state is reached and the reward is averaged over multiple simulations to estimate the strength of each action. MCTS algorithms have gained in popularity in recent years due to their success in the field of *Computer Go* [1]. In particular, the UCT algorithm [2] proposed in 2006 has led to the recent upsurge of interest in MCTS algorithms.

MCTS algorithms build a subtree of the entire decision tree where usually one new node is added after every simulation. Each node stores estimates of the rewards obtained by selecting each action and an improved estimate is available after every simulation step. Each decision in the tree is treated as a multi-armed bandit problem where the arms are actions, and the rewards are the results of performing a Monte Carlo simulation after selecting that action. MCTS is an anytime algorithm, requiring little domain knowledge.

B. AI for Games of Imperfect Information

This section briefly surveys research on AI for games with stochasticity and/or imperfect information.

1) Determinization: One approach to designing AI for games with stochasticity and/or imperfect information is determinization, also known as perfect information Monte Carlo (PIMC) [18]. For an instance of a stochastic game with imperfect information, a determinization is an instance of the equivalent deterministic game of perfect information, in which the current state is chosen from the AI agent's current information set, and the outcomes of all future chance events are fixed and known. For example, a determinization of a card game is an instance of the game where all players' cards, and the shuffled deck, are visible to all players. We then create several determinizations from the current game state, analyze each one using AI techniques for deterministic games of perfect information, and combine these decisions to yield a decision for the original game. The term determinization refers to the process of converting a game of imperfect information to an instance of a game of perfect information. The AI technique of analyzing multiple determinizations to make a decision is often called Monte Carlo sampling (of determinizations). We refer to Monte Carlo sampling of determinizations simply as determinization to avoid confusion with the Monte Carlo sampling of game simulations used by MCTS algorithms.

Ginsberg's Intelligent Bridge Player (GIB) system [3] applies determinization to create an AI player for the card game *Bridge* which plays at the level of human experts. GIB begins by sampling a set D of card deals consistent with the current state of the game. For each of these deals $d \in D$ and for each available action a, the perfect information ("double dummy") game is searched to find the score $\mu(a, d)$ resulting from playing action a in determinization d. The search uses a highly optimized exhaustive search of the double dummy *Bridge* game tree. Finally, GIB chooses the action a for which the sum $\sum_{d \in D} \mu(a, d)$ is maximal.

Bjarnason *et al.* [4] present a variant of UCT for stochastic games, called sparse UCT, and apply it to the single-player card game of *Klondike Solitaire*. Bjarnason *et al.* [4] also study an ensemble version of sparse UCT, in which several search trees are constructed independently and their results (the expected rewards of actions at the root) are averaged. They find that ensemble variants of UCT often produce better results in less time than their single-tree counterparts. A special case of ensemble sparse UCT, which Bjarnason *et al.* call HOP-UCT, is equivalent to a straightforward application of determinization (more specifically, hindsight optimization [19]) with UCT as deterministic solver, in which the determinization is constructed lazily as UCT encounters each chance event.

Bjarnason *et al.* [4] treat *Klondike Solitaire* as a stochastic game of perfect information: rather than being fixed from the start of the game, the values of face down cards are determined as chance events at the moment they are revealed. This works for single-player games where the hidden information does not influence the game until it is revealed, but generally does not work for multiplayer games where the hidden information influences the other players' available and chosen actions from the beginning of the game. Hence, the specific methods of sparse UCT and lazy determinization are not immediately applicable to multiplayer games, but the general ideas may be transferable. Bjarnason *et al.* [4] show that sparse UCT is able to win around 35% of *Klondike Solitaire* games, which more than doubles the estimated win rate for human players. Determinization is also the state-of-the-art approach for card games such as *Bridge* [3] and *Skat* [20], [21]. Determinized MCTS also shows promise in games such as *Phantom Go* [22] and *Phantom Chess (Kriegspiel)* [23], among others.

Despite these successes, determinization is not without its critics. Russell and Norvig [24] describe it (somewhat dismissively) as "averaging over clairvoyance." They point out that determinization will never choose to make an information gathering play (i.e., a play that causes an opponent to reveal some hidden information) nor will it make an information hiding play (i.e., a play that avoids revealing some of the agent's hidden information to an opponent). Ginsberg [3] adds weight to this claim by making the same observations about GIB specifically.

Russell and Norvig's criticisms of determinization are valid but equally valid are the experimental successes of determinization. Frank and Basin [5] identify two key problems with determinization.

- *Strategy fusion:* An AI agent can obviously not make different decisions from different states in the same information set (since, by definition, the agent cannot distinguish such states); however, different decisions can be made in different determinizations.
- *Nonlocality:* Some determinizations may be vanishingly unlikely (rendering their solutions irrelevant to the overall decision process) due to the other players' abilities to direct play away from the corresponding states.

Building on the work of Frank and Basin, Long *et al.* [18] identify three parameters of game trees and show that the effectiveness of determinization is related to a game's position in this parameter space. The parameters measure the ability of a player to influence the outcome of a game in its late stages (leaf correlation), the bias in the game toward a particular player (bias) and the rate at which hidden information is revealed (disambiguation). Long *et al.* [18] demonstrate how these parameters can be used to predict whether determinization is an appropriate method for a given game.

The effects of strategy fusion can manifest in different ways. First, strategy fusion may arise since a deterministic solver may make different decisions in each of the states within an information set. In this situation, the issue is that the agent assumes a different decision can be made depending on the state and this information is not known. The SO-ISMCTS algorithm described in Section IV-E addresses this issue by searching a single tree of information sets. Second, strategy fusion may arise from partially observable moves. When an opponent makes a partially observable move, a deterministic solver will assume that the move can be observed and that it can make a different decision depending on the actual move made by the opponent. The MO-ISMCTS algorithm described in Section IV-G addresses this by searching a tree for each player, which builds a tree based on information sets which cannot be distinguished by a player after observing a move made by an opponent.

2) Other Approaches: One alternative approach to tree search for stochastic games is expectimax search [24]. This is a modification of the well-known minimax algorithm to game trees containing chance nodes. The value of a chance node is the expected value of a randomly chosen child (i.e., the sum of the values of its children weighted by the probabilities of the corresponding chance outcomes).

Stochastic games of imperfect information have been well studied in the field of game theory [17]. Thus, a popular approach to AI for such games is to compute (or approximate) a Nash equilibrium strategy; examples of this approach include Gala [25] and counterfactual regret [26]. The definition of Nash equilibrium requires only that the strategy be optimal against other optimal (Nash) strategies, so Nash strategies often fail to fully exploit suboptimal opponents. Also for many domains the number of states is far too large to compute or even approximate a Nash equilibrium.

3) Belief Distributions: In games of imperfect information, it is often possible to infer hidden information by observing the moves of the other players, according to some model of the other players' decision processes. One way of capturing this notion is via *belief distributions*, probability distributions over states in the current information set where the probabilities are inferred from the history of observed moves. This type of inference has frequently been applied to the game of *Poker* [12], [27], but also to other games such as *Scrabble* [28] and the card game *Skat* [20], [21]. We do not consider belief distributions in this paper.

C. Handling Uncertainty in Tree Search

1) Simultaneous Moves: *Simultaneous moves* are a special case of imperfect information, in which each player independently chooses an action and these actions are applied at the same time.

Simultaneous moves can be modeled by having players choose their actions sequentially, while hiding their choices from the other players, until finally an environment action reveals the chosen actions and resolves their effects. With this in mind, any algorithm that can handle imperfect information in general can handle simultaneous moves in particular. However, some of our algorithms (particularly those not designed to handle partially observable moves) perform poorly using this model. Under a simple determinization approach, the first player is overly pessimistic (assuming the opponent can observe the chosen move and select the best response to it) while the second player is overly optimistic (assuming the first player's move is fixed at the point of the second player's decision, and thus determinizing it randomly).

For this reason, we add a mechanism to the algorithms studied in this paper specifically to handle simultaneous moves. The UCT algorithm has been applied to the simultaneous move game *Rock-Paper-Scissors* by Shafiei *et al.* [13], using an approach where each player's choice of action is treated as a separate independent multiarmed bandit problem. In other words, instead of selecting player 1's move, descending the corresponding tree branch, and selecting player 2's move from the resulting child node, both moves are selected independently from the same node and the tree branch corresponding to the resulting pair of moves is descended. Shafiei *et al.* [13] show that this approach finds mixed policies, though not necessarily Nash policies.

Teytaud and Flory [29] suggest a modification of this approach, in which the UCB bandit algorithm is replaced by the EXP3 algorithm [30] at nodes with simultaneous moves only (i.e., UCB is still used elsewhere in the tree). The justification for using EXP3 rather than UCB is that the optimal policy at a simultaneous move node is often mixed; UCB is designed to converge to a pure policy, whereas EXP3 explicitly seeks a mixed policy. Teytaud and Flory [29] further strengthen this justification by comparing the playing strength of UCB versus EXP3 for the card game *Urban Rivals*, showing that EXP3 performs better and requires less tuning. In this paper, our algorithms use the EXP3 approach to handle simultaneous moves.

In EXP3, the probability of selecting an arm $a \in A$ is

$$p(a) = \frac{\gamma}{|A|} + \frac{1-\gamma}{\sum_{b \in A} e^{(s(b)-s(a))\eta}} \tag{4}$$

where $s(a)$ is the sum of rewards from previously selecting arm a, each divided by the probability of selecting a on that trial, and η and γ are constant parameters. This equation is of a different form to that given by Auer *et al.* [30], but is equivalent and more numerically stable.

Naturally the performance of EXP3 depends on the choice of coefficients. After [30, Corollary 4.2], we take

$$\gamma = \min\left\{1, \sqrt{\frac{K \log K}{(e-1)n}}\right\} \tag{5}$$

and

$$\eta = \frac{\gamma}{K} \tag{6}$$

where $K = |A|$ is the number of arms, n is the total number of trials, and e is the base of the natural logarithm.

2) Chance Nodes: Handling of chance events is not a primary focus of this paper. However, chance nodes do occur under certain circumstances in one of our test domains (Section V-A), so they cannot be ignored completely. Note that our chance nodes have a small number of possible outcomes (at most four but rarely more than two), all with equal probability. Technically, another test domain (Section VII-A) includes a chance event with combinatorially many outcomes corresponding to shuffling and dealing a deck of cards at the beginning of the game, but since this occurs before any player has made a decision it never occurs as a chance node in our search tree.

Consider a chance node with k branches. To ensure that each branch is explored approximately equally, the first k visits select all outcomes in a random permutation, the second k visits select all outcomes in another random permutation, and so on. This is almost trivial to implement in UCT: since we already use UCB with random tie-breaking for action selection, it suffices to treat the environment player as a decision-making agent who has perfect information and receives a reward of zero for all terminal states. The UCB exploration term then ensures that the branches are visited in the manner described above.

IV. Description of Algorithms

This section describes the algorithms studied in this paper. Section IV-B introduces the *subset-armed bandit problem*, a variant of the multiarmed bandit problem that is used in several of the algorithms. Sections IV-C and IV-D describe how MCTS (and specifically UCT) can be adapted to games of imperfect information, by "cheating" (gaining access to the hidden information) and by determinization respectively. Section IV-E–IV-G introduce the ISMCTS family of algorithms, novel variants of MCTS for searching trees in which nodes correspond to information sets rather than states. These algorithms also make use of the techniques described in Section III-C for handling chance nodes and simultaneous moves.

A. Choice of UCB1 Exploration Constant

UCT [2] uses the UCB1 algorithm [31] for action selection. UCB1 calculates the score of a node as

$$\overline{X}_j + c\sqrt{\frac{\ln n}{n_j}} \tag{7}$$

where $\overline{X}_j$ is the average reward of the simulations passing through the node j, n is the number of times the parent of j has been visited by the algorithm, and n_j is the number of times the node j was selected from its parent.

When using UCB1 an important issue is the choice of coefficient c for the exploration term in the UCB1 formula. The choice of this parameter can affect playing strength and the optimal value can depend on both the domain and the MCTS algorithm used. We conducted an experiment where determinized UCT, SO-ISMCTS, and MO-ISMCTS players with exploration constant $\{0.25, 0.5, 0.75, \ldots, 1.75, 2\}$ played repeatedly against determinized UCT with exploration constant 0.7 for each of the three games studied. It was observed that none of the algorithms are particularly sensitive to the coefficient value for these games, although performance does decrease outside the range $[0.5, 1]$. The value of 0.7 was thus used for all algorithms in all experiments in this paper.

B. Subset-Armed Bandits

In the following algorithms, we sometimes have the situation where the set of actions available at a particular node in the tree varies between visits to that node. If the observer of an information set is not the player about to act, then different states in the same information set can have different sets of legal actions: for example, the actions available to an opponent may depend on the cards in the opponent's hand, which vary for different states in the player's information set. Consequently, a node for an information set has a branch for every action that is legal in some state, but which branches are valid depends on which state is currently under consideration.

In other words, we have a multiarmed bandit where only a subset, and generally a different subset, of the arms is available on each trial. We call this a *subset-armed bandit*. Since this problem arises from considering legal action sets for different states in an information set, it is not enough to say that each arm is available with some probability: the availability of two different arms in the same trial is often correlated. We can, however, say that the subsets available on different trials are independent.

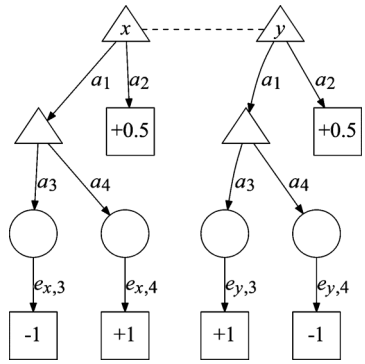


Fig. 1. A game tree for a simple single-player game. Nodes represent game states. Nodes shaped △ denote player 1 decision states, ◯ environment states, and □ terminal states labeled with reward values for player 1. Nonterminal nodes in corresponding positions in the x and y subtrees are in the same player 1 information set; this is shown by a dashed line for the root nodes. Adapted from [18, Fig. 1].

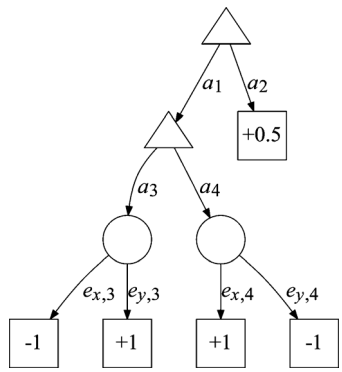


Fig. 2. An information set search tree for the game shown in Fig. 1. Here nodes shaped △ denote information sets where player 1 is both the observer and the player about to act.

We apply a simple modification to UCB1 and other standard bandit algorithms to handle subset-armed bandits. We replace n in (7) with the number of trials in which the parent was visited *and node j was available*. Without this modification, rare actions (i.e., actions available in few states in the information set) are over-explored: whenever they are available for selection their ratio of visits to parent visits is very small, resulting in a disproportionately large UCB value. If every state in the information set has a rare action, this results in the search doing almost no exploitation and almost all exploration.

One drawback of this approach is that it does not allow an action to have a different value depending on which subset of actions it belongs to (instead the value is the average across all visited subsets). An alternative approach was considered in which the statistics used in (7) are calculated independently for each subset. This allows the choice of action to depend on the set of actions available, but requires many visits to each subset to gather accurate statistics, making it impractical when the number of subsets is large. An in-depth analysis of the mathematics of subset-armed bandits is a subject for future work.

C. Cheating UCT

As a benchmark we consider UCT agents that are allowed to "cheat" and observe the actual state of the game. (Throughout

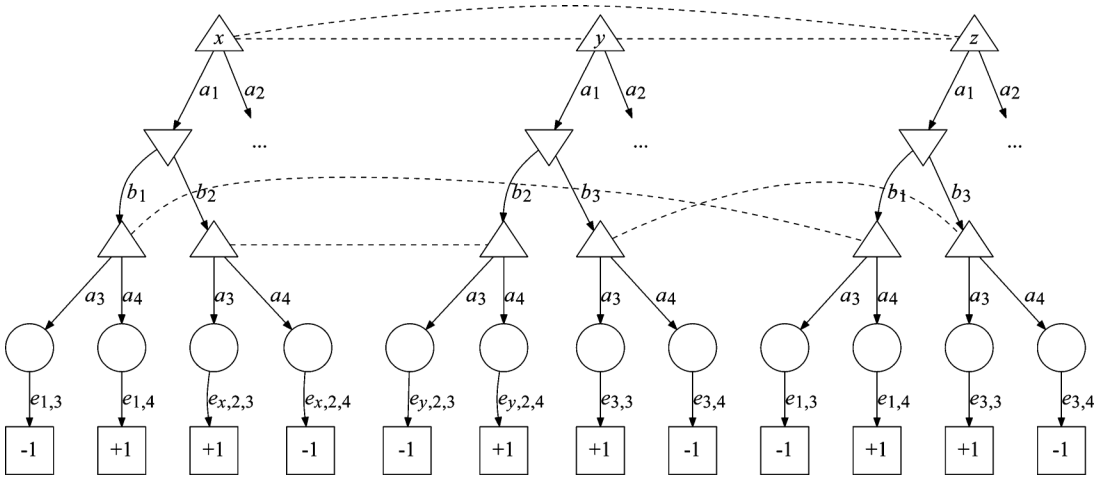


Fig. 3. A game tree for a simple two-player game. Nodes shaped △ denote player 1 decision states, ▽ player 2 decision states, ◯ environment states, and □ terminal states labeled with reward values for player 1 (the game is zero-sum, so player 2's rewards are the negation of those for player 1). Player 1's information set relation is shown by dashed lines for selected nodes. The partitioning of the remaining nodes is determined by their positions in subtrees: if two nodes occupy the same position in two subtrees, and the roots of those subtrees are in the same information set as each other, then the two nodes are in the same information set as each other. The remaining nodes are partitioned in the obvious way. Player 2 has perfect information, i.e., her information sets are singletons.

the rest of this paper, the word *cheat* refers specifically to observing information that is supposed to be hidden or uncertain, rather than any other violation of the game rules.) Cheating in this way is not a valid approach to AI for games of imperfect information, but it provides a useful benchmark for other algorithms since it is an approach which is expected to work very well compared to approaches that do not cheat.

For fair comparison with our other algorithms, we consider two cheating UCT agents: one using plain UCT with a single search tree, and one using ensemble UCT [32] with several independent search trees whose root statistics are combined at the end of the search. As we will see, these are cheating versions of information set MCTS and determinized UCT respectively.

D. Determinized UCT

Our simplest noncheating agent uses a determinization approach, as described in Section III-B1. It samples a number of (not necessarily different) states from the current information set uniformly at random, constructs independently a UCT tree rooted at each of these states, and chooses a move for which the number of visits from the root, summed across all trees, is maximal.

E. Single-Observer Information Set MCTS (SO-ISMCTS)

To overcome the problems associated with the determinization approach, we propose searching a single tree whose nodes correspond to information sets rather than states. In single-observer information set MCTS (SO-ISMCTS), nodes in the tree correspond to information sets from the root player's point of view, and edges correspond to actions (i.e., moves from the point of view of the player who plays them). The correspondence between nodes and information sets is not one–one: partially observable opponent moves that are indistinguishable to the root player have separate edges in the tree, and thus the resulting information set has several nodes in the tree. We address this in subsequent sections.

Fig. 1 shows a game tree for a simple single-player game of imperfect information. The root information set contains two states: x and y. The player first selects one of two actions: a_1 or a_2. Selecting a_2 yields an immediate reward of $+0.5$ and ends the game. If the player instead selects a_1, he must then select an action a_3 or a_4. If the game began in state x, then a_3 and a_4 lead to rewards of -1 and $+1$, respectively (this information being revealed by means of environment action $e_{x,3}$ or $e_{x,4}$); if the game began in state y, then the rewards are interchanged.

If states x and y are equally likely, action a_1 has an expectimax value of 0: upon choosing a_1, both a_3 and a_4 have an expectimax value of 0. Thus, the optimal action from the root is a_2. However, a determinizing player searches trees corresponding to each state x and y individually and assigns a_1 a minimax value of $+1$ in each (by assuming that the correct choice of a_3 or a_4 can always be made), thus believing a_1 to be optimal. This is an example of strategy fusion (Section III-B1).

Fig. 2 shows the tree searched by SO-ISMCTS for this game. In this case, each node is in one–one correspondence with an information set. After a sufficiently large number of iterations the algorithm assigns each environment node an expected value of 0 and thus assigns the same value to action a_1, thus overcoming strategy fusion and correctly identifying a_2 as the optimal move.

Fig. 3 shows a game tree for a more complex, two-player game. The game starts in one of three states: x, y, or z. These states are distinguishable to player 2 but not to player 1. Player 1 first selects an action a_1 or a_2. If he chooses a_1, player 2 then selects an action b_1, b_2, or b_3. However, only two of these actions are available, and which two depends on the initial state. Player 1 then selects a_3 or a_4, and both players receive rewards as shown. Note that if player 2 chooses b_1 or b_3, then the rewards do not depend on the initial state, but if player 2 chooses b_2, then the rewards do depend on the initial state.

Fig. 4(a) shows the tree searched by SO-ISMCTS for this game. For an information set $[s]^{\sim_i}$ where the observer is not the player about to act, i.e., $\rho([s]^{\sim_i}) \neq i$, the set $A(s)$ of available actions can differ for different states $s \in [s]^{\sim_i}$. The set of legal actions may depend on information to which another player does not have access. When searching trees of information sets, this creates a problem at opponent nodes. There must

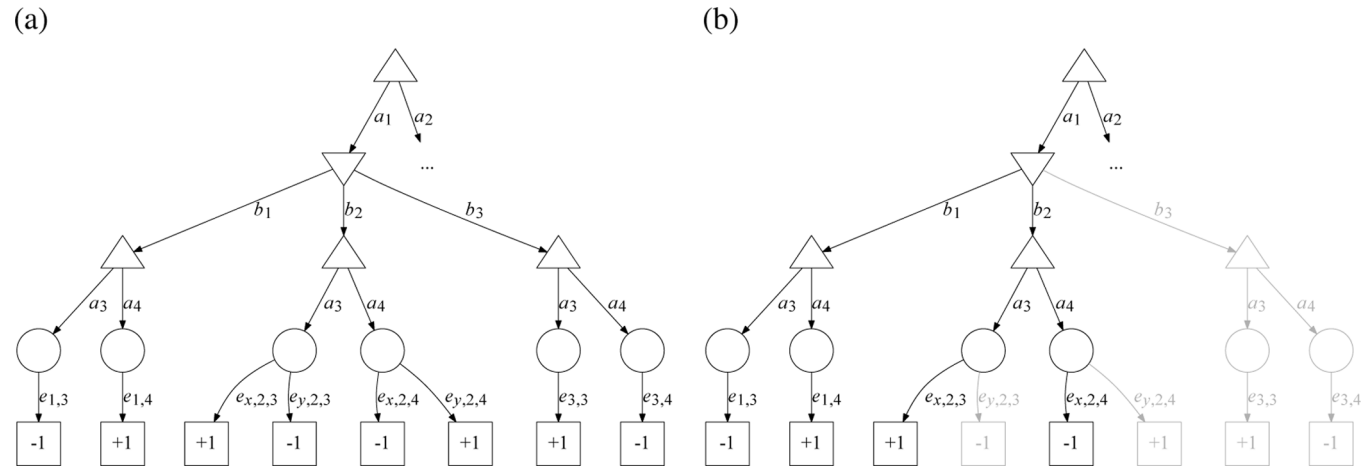


Fig. 4. An information set search tree for the game shown in Fig. 3. (a) The entire tree. (b) The restriction of the tree to determinization x.

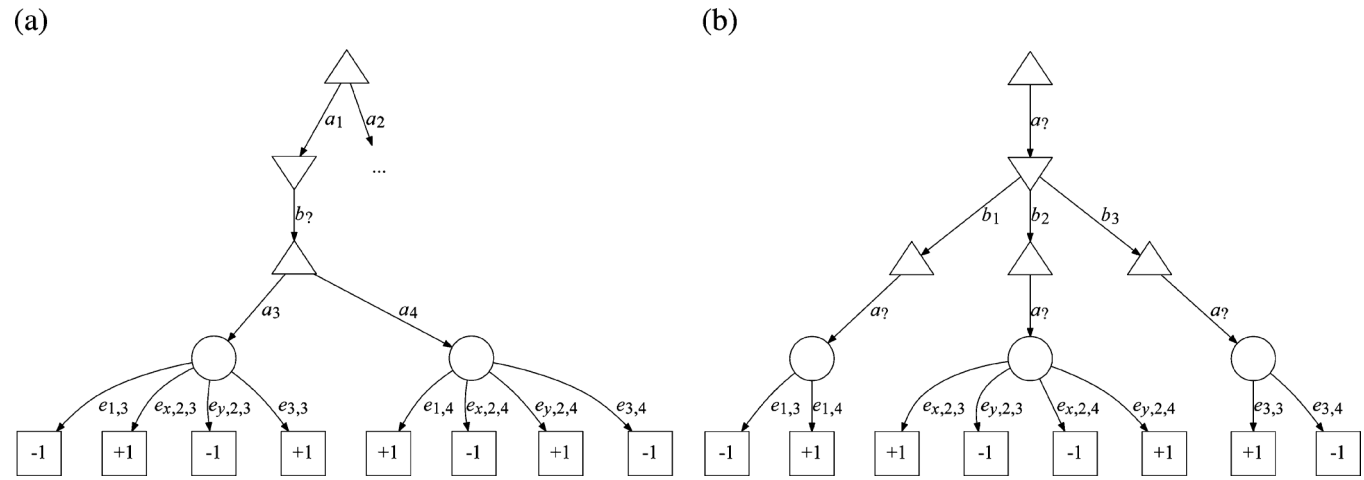


Fig. 5. Information set search trees for the game shown in Fig. 3 with partially observable moves, where player 2 cannot distinguish a_1 from a_2 or a_3 from a_4 and player 1 cannot distinguish between b_1, b_2 and b_3: (a) the tree searched by SO-ISMCTS; (a) and (b) the pair of trees searched by MO-ISMCTS, where (a) is from player 1's point of view and (b) from player 2's point of view.

be a branch for every action that can possibly be available from that information set; this is illustrated in Fig. 4(a), where the opponent decision node has branches for all three actions b_1, b_2, b_3 even though only two of those three actions are available in each state $f(x, a_1)$, $f(y, a_1)$, $f(z, a_1)$ in the corresponding player 1 information set. However, the exploitation and exploration of actions must be balanced with how likely those actions are to be available. For example, we wish to avoid overexploiting an action that is a certain win for the opponent but is only available with probability 1/100 (i.e., in only one of 100 states in the information set).

To address this, at the beginning of each iteration, we choose a determinization, and restrict that iteration to those regions of the information set tree that are consistent with that determinization. Thus, the branches at opponent nodes are available for selection precisely as often as a determinization is chosen in which the corresponding action is available. In other words, the probability of an action being available for selection on a given iteration is precisely the probability of sampling a determinization in which that action is available. The set of actions available at an opponent node can differ between visits to that node, and thus action selection is a subset-armed bandit problem (Section IV-B). Fig. 4(b) demonstrates such a restriction of the search tree shown in Fig. 4(a).

High-level pseudocode for the SO-ISMCTS algorithm is presented in Algorithm 1. More detailed pseudocode is given in part A of the Appendix. In this and other pseudocode in this paper, it is assumed that player 1 is conducting the search. The pseudocode does not specify which bandit algorithm is used during selection. The experiments in this paper all use UCB modified for subset-armed bandits as described in Section IV-B, or EXP3 as described in Section III-C-I at nodes with simultaneous moves (which only occur in LOTR:C, Section V).

Algorithm 1: High-level pseudocode for the SO-ISMCTS algorithm. More detailed pseudocode is given in part A of the Appendix. For the variant of this algorithm with partially observable moves (SO-ISMCTS+POM) simply replace the word "action" below with "move (from player 1's viewpoint)," and see the more detailed pseudocode in part B of the Appendix.

1: **function** SO-ISMCTS($[s_0]^{\sim_1}, n$)

create a single-node tree with root v_0 corresponding to the root information set $[s_0]^{\sim_1}$ (from player 1's viewpoint)
for n iterations **do**
  choose a determinization d at random from $[s_0]^{\sim_1}$, and use only nodes/actions compatible with d this iteration

  //Selection
  repeat
    descend the tree (restricted to nodes/actions compatible with d) using the chosen bandit algorithm
  until a node v is reached such that some action from v leads to a player 1 information set which is not currently in the tree **or until** v is terminal

  //Expansion
  if v is nonterminal **then**
    choose at random an action a from node v that is compatible with d and does not exist in the tree
    add a child node to v corresponding to the player 1 information set reached using action a and set it as the new current node v

  //Simulation
  run a simulation from v to the end of the game using determinization d

  //Backpropagation
  for each node u visited during this iteration **do**
    update u's visit count and total simulation reward
    for each sibling w of u that was available for selection when u was selected, including u itself **do**
      update w's availability count

return an action from the root node v_0 such that the number of visits to the corresponding child node is maximal

The idea of constructing trees of information sets and sampling determinizations to restrict the region to be searched is similar to the partially observable UCT (PO-UCT) approach of Silver and Veness [33], although PO-UCT operates on the domain of partially observable Markov decision problems (i.e., single-player games of imperfect information) rather than adversarial games. Schäfer [21] also applied an information set tree approach for the game *Skat* using the UCB1 algorithm for selection. The information sets are from the point of view of the player about to play, rather than from the point of view of one player as in SO-ISMCTS.

Consider the example tree in Fig. 4(b). Note that the restricted tree is never explicitly constructed, but the tree policy is restricted as it descends the tree by means of the determinization d. In turn, d is updated as the tree is descended by applying the selected actions. Otherwise, selection works as in plain UCT. Suppose that we used determinization x and the sequence of actions selected is a_1, b_2, a_3, $e_{x,2,3}$. Let us identify each of the visited nodes with its incoming action (i.e., the label of the incoming edge). At nodes $e_{x,2,3}$, a_3, b_2, a_1 and the root, the visit count $n(v)$ and total reward $r(v)$ is updated as usual. For these nodes and for all siblings that were also available for selection, i.e., including nodes a_4 and b_1 but *not* nodes b_3 and $e_{y,2,3}$, the availability count $n'(v)$ is incremented by 1. The availability count replaces the parent node's visit count in the UCB formula in order to adapt UCB to the subset-armed bandit problem, as discussed in Section IV-B.

F. Single-Observer Information Set MCTS With Partially Observable Moves (SO-ISMCTS + POM)

SO-ISMCTS does not completely avoid the problem of strategy fusion, as it treats all opponent moves as fully observable. Suppose that the game in Fig. 3 is modified to include partially observable moves, so that player 2 cannot distinguish a_1 from a_2 nor a_3 from a_4 and player 1 cannot distinguish between b_1, b_2 and b_3. Here the search assumes that different actions can be taken in response to opponent actions b_1 and b_2, for instance, whereas in fact these actions are indistinguishable and lead to the same player 1 information set.

In SO-ISMCTS, edges correspond to actions, i.e., moves from the point of view of the player who plays them. In single-observer information set MCTS with partially observable moves (SO-ISMCTS + POM), edges correspond to moves from the point of view of the root player. Thus, actions that are indistinguishable from the root player's point of view share a single edge in the tree. Fig. 5(a) shows the SO-ISMCTS + POM search tree for the game in Fig. 3. The branches from player 1's decision nodes are unchanged; however, player 2's decision node now has a single branch corresponding to the single move from player 1's point of view, rather than one branch for each action.

As in SO-ISMCTS, each iteration is guided by a determinization. This raises the problem of how to update the determinization d according to a selected partially observable opponent move. For a determinization d and a move $[a]^{\sim_1}$, the set of compatible actions is $\alpha = A(d) \cap [a]^{\sim_1}$. If α is a singleton, then we simply apply its single element to d to obtain the determinization for the next level in the tree. If $|\alpha| > 1$, then we choose an action from α uniformly at random, since the tree does not store any data with which to make a more informed choice.

High level pseudocode for the algorithm is as given in Algorithm 1 for SO-ISMCTS, except that "action" must be replaced by "move (from player 1's viewpoint)." Part B of the Appendix has more detailed pseudocode which specifies how the determinization should be maintained as we descend the tree for SO-ISMCTS-POM.

Consider the example in Fig. 5(a). SO-ISMCTS + POM functions in much the same way as SO-ISMCTS (recall the example at the end of Section IV-E), except that branch $b_?$ is selected in place of b_2. When updating the determinization while descending the tree, an action must be applied corresponding to the selection of $b_?$. In this case, one of b_1, b_2, or b_3 is applied depending on which are legal actions in the current determinization. For each determinization, there are two possibilities, so one is chosen uniformly at random.

G. Multiple-Observer Information Set MCTS (MO-ISMCTS)

SO-ISMCTS + POM solves the strategy fusion problem of SO-ISMCTS, at the expense of significantly weakening the opponent model: in particular, it is assumed that the opponent chooses randomly between actions that are indistinguishable to the root player. In the extreme case, when SO-ISMCTS + POM is applied to a phantom game (as in Section VI) all opponent actions are indistinguishable and so the opponent model is essentially random.

To address this, we propose multiple-observer information set MCTS (MO-ISMCTS). This algorithm maintains a separate tree for each player, whose nodes correspond to that player's information sets and whose edges correspond to moves from that player's point of view. Each iteration of the algorithm descends all of the trees simultaneously. Each selection step uses statistics in the tree belonging to the player about to act in the current determinization to select an action. Each tree is then descended by following the branch corresponding to the move obtained when the corresponding player observes the selected action, adding new branches if necessary.

The information set trees can be seen as "projections" of the underlying game tree. Each iteration induces a path through the game tree, which projects onto a path through each information set tree. Fig. 5 depicts these trees for the simple game of Fig. 3: Fig. 5(a) corresponds to information sets and moves from player 1's point of view, and Fig. 5(b) from player 2's point of view.

The MO-ISMCTS approach is similar to the MMCTS algorithm proposed by Auger [34]. However, there are several differences between MO-ISMCTS and MMCTS, the most important being that MO-ISMCTS uses determinizations to guide and restrict each search iteration whereas MMCTS does not. Also, whereas Auger [34] describes use of MMCTS in an offline manner (running the algorithm for a very large number of simulations and querying the resulting tree for decisions during play), MO-ISMCTS is designed for the more conventional online mode of play.

Pseudocode for MO-ISMCTS is given in Algorithm 2 and part C of the Appendix.

Algorithm 2: High-level pseudocode for the MO-ISMCTS algorithm. More detailed pseudocode is given in part C of the Appendix.

function MO-ISMCTS($[s_0]^{\sim_1}, n$)
  for each player $i = 0, \ldots, \kappa$, create a single-node tree with root v_0^i (representing $[s_0]^{\sim_1}$ from player i's viewpoint)
  for n iterations **do**
    choose a determinization d at random from $[s_0]^{\sim_1}$, and use only nodes/actions compatible with d this iteration

    //Selection
    repeat
      descend all trees in parallel using a bandit algorithm on player ρ's tree whenever player ρ is about to move
    until nodes $v^0, \ldots, v^\rho, \ldots, v^\kappa$ are reached in trees $0, \ldots, \kappa$ respectively, player ρ is about to move at node v^ρ, and some action from v^ρ leads to a player ρ information set which is not currently in the player ρ tree **or until** v^ρ is terminal

    //Expansion
    if v^ρ is nonterminal **then**
      choose at random an action a from node v^ρ that is compatible with d and does not exist in the player ρ tree
      for each player $i = 0, \ldots, \kappa$ **do**
        if there is no node in player i's tree corresponding to action a at node v^i, then add such a node

    //Simulation
    run a simulation from v^ρ to the end of the game using determinization d, (starting with action a if v^ρ is nonterminal)

    //Backpropagation
    for each node u^i visited during this iteration, for all players i **do**
      update u^i's visit count and total simulation reward
      for each sibling w^i of u^i that was available for selection when u^i was selected, including u^i itself **do**
        update w^i's availability count

  return an action from $A([s_0]^{\sim_1})$ such that the number of visits to the corresponding child of v_0^1 is maximal

Consider the example in Fig. 5. We begin by randomly generating a determinization. We then select an action from the root of Fig. 5(a) (i.e., player 1's tree), say a_1. We descend the trees by following the branch for the move corresponding to a_1, namely a_1 for player 1's tree and $a_?$ for player 2's tree. The determinization d is updated by applying action a_1. We now have $\rho(d) = 2$ so Fig. 5(b) (player 2's tree) is used for selection, and an action legal in d is selected, say b_2. The trees are descended through $b_?$ and b_2, respectively, and d is updated by applying b_2. The selection process continues in this way. Backpropagation works similarly to SO-ISMCTS (as in the example at the end of Section IV-E), but updates all visited nodes (and their available siblings) in each player's tree.

V. Experimental Results for *Lord of the Rings: The Confrontation*

A. The Game

Lord of the Rings: The Confrontation (LOTR:C) [6] is a two-player strategy game themed on J. R. R. Tolkien's *The Lord of the Rings* novels. The gameplay has common features with *Stratego* [7], where identities (but not locations) of a player's pieces are hidden from the opponent. Furthermore, the identity of a piece specifies certain unique characteristics. *LOTR:C* is an interesting game from an AI point of view since it features hidden information, chance events, partially observable

moves and simultaneous moves. It is also asymmetric since both players have different win conditions and thus require different tactics and strategies.

1) Game Structure: The game is played on a 4×4 grid, with the players' home squares at opposite corners. Most squares can be occupied by more than one piece simultaneously, subject to restrictions. The players are designated Light and Dark, with Dark playing first. Each player has nine character pieces, which they place on the board at the start of the game subject to certain constraints. Each character has an associated strength value between 0 and 9, and a special ability that changes the rules of the game in certain situations. Light's characters are different from Dark's. Generally characters move one space at a time toward the opponent's home square, although some characters and some squares on the board allow for different moves.

The identities of a player's characters are hidden from the opponent until revealed in combat. This leads to a source of hidden information, where the information set specifies the number of opponent pieces in each square and the states in the information set specify the identity of all the pieces. When an opponent moves one of their pieces, this move is partially observable since a player knows a piece moved (and this leads to a new information set) but only the opponent knows which piece moved. Knowledge about the locations of opposing characters can decrease as well as increase. For example, if a character whose identity is known enters a square with an unknown character and then later exits the square, the identities of both the exiting character and the remaining character are unknown. Since players must move pieces forward (aside from a few special rules), the *LOTR:C* game tree has very few cycles and random games are almost always fairly short.

2) Objectives: LOTR:C has multiple win conditions, which differ for each player. For the Light player there are three ways to win:

- moving the character Frodo into Dark's home square;
- killing all Dark characters;
- the Dark player being unable to move any characters.

For the Dark player there are also three ways to win:

- killing the character Frodo;
- moving any four characters into Light's home square;
- the Light player being unable to move any characters.

3) Combat: When a character moves into a square that contains opponent characters, combat is initiated. The moving character becomes the attacker and a randomly chosen opponent character in the square is the defender, then both players simultaneously choose one of the combat cards from their hand. This leads to simultaneous moves being a feature of the game. Each player begins with nine cards (which are removed once played) and each character has a strength value, as do some of the cards. In combat, the player whose combined character and card strength is greatest wins the combat. Some characters and some cards feature text that can alter the outcome of the combat, by either offering a player extra choices or altering the rules of combat. Typically the outcome of combat is that one or both characters are defeated and removed from play.

4) Implementation: Character movement in *LOTR:C* is partially observable. We define actions such that they identify the character and the source and destination squares (e.g., "move Frodo from Cardolan to Eregion"). The move observed by the opponent does not identify the character (e.g., "move a character from Cardolan to Eregion").

Some care is needed to ensure the structure of the game tree, particularly around combat, conforms to that described in Section II. An environment player is used to model actions taken by the game. Specifically, the environment player is responsible for deciding the outcome of chance events and for revealing information to players. In our implementation, a typical instance of combat consists of the following sequence of actions:

1) the attacking player moves a piece into a square occupied by an opponent piece;
2) the environment player reveals the identities of the attacker and defender pieces, choosing a defender at random if necessary (which leads to a source of chance events);
3) one player chooses a card;
4) the other player chooses a card (steps 3 and 4 occur simultaneously);
5) the environment player reveals the chosen cards and resolves the combat.

A skilled human player of *LOTR:C* remembers the information revealed about the identities of characters. Our implementation has perfect recall for all players: information about which characters can possibly occupy which squares based on previously revealed information is encoded in the game state. In particular, our algorithms always sample determinizations that are consistent with this information.

5) Initial Setup: Before each game, players can place their characters on the board in any configuration subject to certain constraints. The choice of initial setup has important strategic consequences, however tree search is not well suited to solving this problem: each player has a choice between $9!/4! = 15\,120$ possible initial setups for their pieces, and both players choose simultaneously. We do not tackle this problem in this paper; instead, we conduct all of our experiments on a single, hand-designed initial setup intended to be typical of those that a pair of human players might choose. This reuse of the same initial setup also has the effect of reducing the variance in our experimental results. No information persists between trials, so there is no danger of the algorithms adapting themselves to this particular setup.

B. Balancing Determinizations and Iterations in Determinized UCT

This experiment studies the effect on the playing strength of determinized UCT of varying the number of determinizations while keeping the total number of iterations fixed. We present a similar experiment for *Dou Di Zhu* in [9], and find that as long as the number of determinizations and the number of iterations per determinization are sufficiently large, this tradeoff has little effect on playing strength, although we will see in Section VII that *Dou Di Zhu* is unusual in that revealing hidden information has little effect, which may contribute to this result.

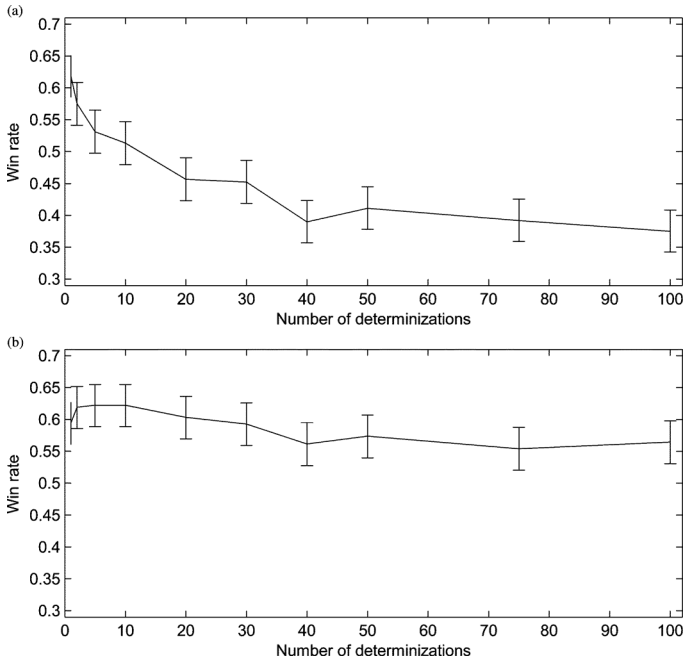


Fig. 6. Results of the determinization balancing experiment for *LOTR:C*. (a) The win rate for a Light player using determinized UCT with d determinizations and $\lfloor 10\,000/d \rfloor$ iterations per determinization, against a Dark opponent with 40 determinizations and $10\,000/40 = 250$ iterations per determinization. (b) The same with Light and Dark exchanged (in particular, win rates are for the Dark player). Error bars show 95% confidence intervals.

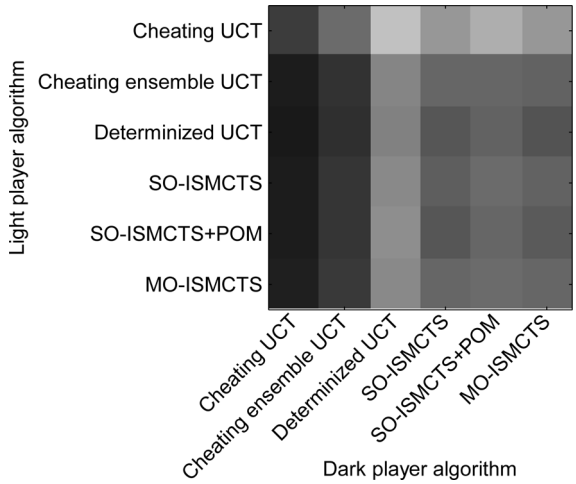


Fig. 7. Heat map showing the results of the *LOTR:C* playing strength experiment. A white square would indicate a 100% win rate for the specified Light player algorithm against the specified Dark player algorithm, while a black square would indicate a 100% win rate for Dark against Light. Shades of gray interpolate between these two extremes.

Results of this experiment for *LOTR:C* are shown in Fig. 6.[1] For brevity, let us write $d \times s$ to refer to a determinized UCT player with d determinizations and s iterations per determinization. Contrary to our results for *Dou Di Zhu*, here the playing strength appears to worsen as the number of determinizations increases. For instance, a Light $1 \times 10\,000$ player significantly outperforms a 40×250 player by 22.9%. This is a consequence of the increased number of iterations per determinization, rather than the reduced number of determinizations: against a 40×250 player a $40 \times 10\,000$ player achieves win rates of 73.2% for Light and 83.0% for Dark, which exceed significantly the corresponding win rates for $1 \times 10\,000$. Naturally, the $40 \times 10\,000$ player also takes approximately 40 times longer to make each decision than a player using a total of 10 000 iterations. Our justification for choosing 10 000 iterations as the standard benchmark is that, with an efficient implementation on modern hardware, we expect 10 000 iterations to equate to roughly one second of computation, which is an acceptable delay for play against a human.

For the $1 \times 10\,000$ player, the average depth of the tree constructed from the initial game state is 8.6, and the average depth of a node is 4.0. For 40×250, the average tree depth is 4.1 and the average node depth is 2.4. Given that a single instance of combat in *LOTR:C* can account for five or more levels in the tree, searching to a depth of 4.1 is simply insufficient to make an informed decision.

The effect of worsening playing strength as the number of determinizations is increased is more pronounced for the Light player. One possible reason for this is that Light's primary win condition (moving Frodo into Mordor) requires more long-term planning and thus deeper search than Dark's primary win condition (kill Frodo).

[1]The confidence intervals shown in this and other figures are Clopper–Pearson intervals [35], considering each game as a Bernoulli trial.

C. Playing Strength

In this experiment, the following algorithms play in a round-robin tournament: cheating UCT, cheating ensemble UCT, determinized UCT, SO-ISMCTS, SO-ISMCTS + POM, and MO-ISMCTS. Each algorithm runs for 10 000 iterations per decision. Determinized UCT uses ten determinizations with 1000 iterations for the Dark player, and applies all 10 000 iterations to a single determinization for the Light. These values were chosen based on the results in Section V-B. Cheating ensemble UCT uses ten trees with 1000 iterations each for both Light and Dark; devoting all iterations to a single tree would be equivalent to cheating single-tree UCT. The results of this experiment are shown in Figs. 7 and 8.

Cheating single-tree UCT consistently outperforms the other algorithms by a large margin. For the Dark player, cheating ensemble UCT outperforms ISMCTS. However, for the Light player, cheating ensemble UCT and ISMCTS are on a par. This is slightly surprising, and would seem to suggest that the benefit of cheating is balanced by the increased depth to which ISMCTS is able to explore the tree (due to devoting all of its iterations to a single tree). That this only holds true for one of the players may shed some light on the differences in approaches required for strong Light and Dark play in *LOTR:C*: as discussed in Section V-B, to Dark the locations of Light's characters are the most important factor, but to Light it is equally important to be able to plan further ahead.

For the Dark player, determinized UCT is outperformed by the other algorithms by a large margin. In particular, determinized UCT is outperformed by all three ISMCTS variants. The success of ISMCTS here is probably due to the reduction in the effects of strategy fusion caused by using a tree of information sets, as well as the additional tree depth that arises

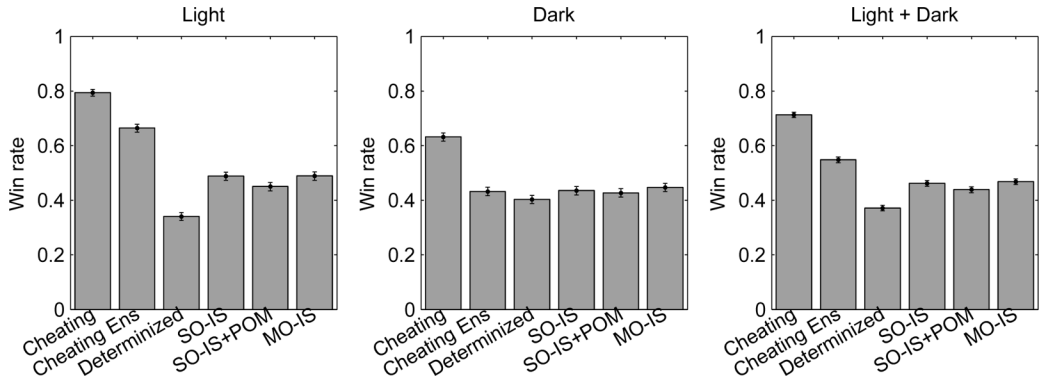


Fig. 8. Results of the playing strength experiment for *LOTR:C*. In the "Light" graph, each algorithm indicated on the x-axis plays an equal number of games as the Light player against each algorithm as the Dark player, and the proportion of wins averaged over all Dark algorithms is plotted. The "Dark" graph is similar, with Light and Dark players interchanged. The "Light + Dark" graph averages these results for each algorithm regardless of player identity. In all cases, error bars show 95% confidence intervals.

by collecting all simulations in a single tree. For the Light player determinized UCT is also worse than ISMCTS, but less dramatically so: the difference between determinized UCT and MO-ISMCTS is around 4.4%, which is significant with 95% confidence. Since the Light player devotes all its computational resources to a single determinization the tree depth argument does not hold, but evidently there is still some benefit to the ISMCTS approach over determinized UCT, most likely the reduced impact of strategy fusion.

There is no significant difference in playing strength between the variants of ISMCTS. SO-ISMCTS + POM seems to perform slightly worse than the other variants of ISMCTS, but this difference is not statistically significant. That there is no significant difference between the algorithms seems to imply that the strategy fusion effects of assuming that opponent moves are fully observable in SO-ISMCTS, and the assumption that the opponent values indistinguishable actions equally in SO-ISMCTS + POM, are not as harmful as intuition may suggest.

As noted above, each trial in this experiment starts from the same hand-designed initial setup. We repeated the experiment with each game beginning from a different randomly generated initial setup. This biases the game slightly toward the Dark player, since a random initial setup is more likely to disadvantage the Light player (e.g., by placing Frodo in a vulnerable starting position). We carried out the same number of trials (750 for each combination of players) in order to achieve similar confidence intervals. Similar results to those above were observed, which support the same conclusions.

D. Playing Strength Versus Human Opponents

The previous section assesses the relative playing strengths of several algorithms for *LOTR:C*, but gives no indication of their absolute strength. We are not aware of any existing AI, commercial or otherwise, for this game. To test the playing strength of MO-ISMCTS, we played several games between an MO-ISMCTS agent and a range of human opponents. The human opponents can be characterized as experienced game players with two having expert ability at *LOTR:C* and five having intermediate ability.

For this experiment, playing all games with the same initial setup would not be a fair test: our AI agents cannot learn the opponent's initial setup between games, but a human player certainly can. We use random initial setups, with constraints to avoid generating particularly bad placements: certain characters (Frodo, Sam, and the Balrog) are always placed in their player's home cell. Since this information is known to the human player, it is also made available to the AI agent by appropriate construction of the initial information set.

When humans play *LOTR:C*, the game is partly a test of memory: one must remember the identities of revealed enemy characters. Since this is trivial for the AI agent, our graphical interface makes this information available to the human player. This ensures that the human and AI players are compared solely on the strength of their decisions, and not on the inherent advantage of a computer player in a test of memory.

We played 32 games with a human player as Dark and the MO-ISMCTS player as Light, and 32 games with a human as Light and MO-ISMCTS as Dark. MO-ISMCTS achieved 14 wins as Light and 16 as Dark. MO-ISMCTS was evenly matched with intermediate to expert human players, so we may conclude that MO-ISMCTS achieved strong play in an absolute sense.

Our human *LOTR:C* players observed anecdotally that MO-ISMCTS plays highly plausible moves, and is particularly adept at engineering favorable endgame scenarios. Its weakest aspect is card play during combat: for example, it has a tendency to waste its more powerful cards in situations where less powerful cards would suffice. Presumably this occurs when the agent does not search deeply enough to see the value of holding onto a more powerful card until later in the game.

VI. Experimental Results for the *Phantom (4, 4, 4)* Game

A. The Game

An m, n, k-*game* [8], [36] is a two-player game played on an $m \times n$ grid. Players take alternating turns to mark a square. The winner is the first player to mark k squares in a horizontal, vertical, or diagonal row. For example, the well-known game of *Noughts and Crosses* (or *Tic-Tac-Toe*) is the 3, 3, 3-game, and *Go-Moku* [37] is the 19, 19, 5-game.

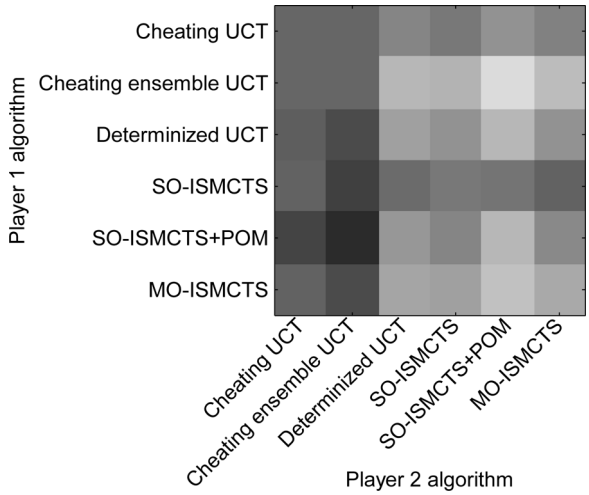


Fig. 9. Heat map showing the results of the *Phantom (4, 4, 4)* game playing strength experiment. A white square would indicate a 100% win rate for the specified player 1 algorithm against the specified player 2 algorithm, while a black square would indicate a 100% win rate for player 2 against player 1. Shades of gray interpolate between these two extremes.

A *Phantom* m, n, k*-game* is an m, n, k-game in which neither player can see the positions of the opponent's marks. If a player tries to mark a square that is already occupied by his opponent, he is told that this is an invalid action and is allowed to choose again. There is no penalty associated with playing an invalid move. Indeed, playing invalid moves is the only mechanism by which the *Phantom* m, n, k*-game* player can gain information about his opponent's previous plays, so doing so is never detrimental and often beneficial. In terms of the game tree, each player action is followed by an environment action specifying whether the move is valid or invalid.

We are not aware of any previous study of *Phantom* m, n, k*-games* in general, although phantom Tic-Tac-Toe (i.e., the phantom 3, 3, 3-game) has been studied by Auger [34] and by Teytaud and Teytaud [38], and other phantom games have been studied in the context of MCTS [22], [23].

In this paper, we study the *Phantom (4, 4, 4)* game (which has enough states that our algorithms do not exhaustively search the full perfect information tree). The perfect information 4, 4, 4-game is known to be a draw [8]. However this analysis does not carry over to the phantom version of the game: intuitively, even a perfect (but noncheating) player cannot block a line that he cannot see. We are not aware of a theoretical analysis of the *Phantom (4, 4, 4)* game; our intuition based on empirical evidence is that the game has no forced result, and while player 1 has a strategy that can lead to a fast win (create four in a row as quickly as possible, hoping that player 2 does not discover or block the line) the game is somewhat balanced overall.

B. Playing Strength

In this experiment, the six algorithms listed in Section V-C again play a round-robin tournament. Each algorithm uses a total of 10 000 iterations, with cheating ensemble UCT and determinized UCT using 40 trees with 250 iterations per tree.

Results of this experiment are shown in Figs. 9 and 10. From the "Players $1+2$" graph in Fig. 10(c) the algorithms can be ordered from best to worst as follows, with statistical significance at 95% confidence in each case:

1) cheating ensemble UCT;
2) cheating UCT;
3) MO-ISMCTS;
4) determinized UCT;
5) SO-ISMCTS;
6) SO-ISMCTS + POM.

Unsurprisingly, the cheating players perform best. The determinization approach appears to be strong for this game, although not as strong as MO-ISMCTS.

There is some asymmetry between the two players in terms of the relative strengths of the algorithms. For player 1, SO-ISMCTS and MO-ISMCTS are on a par while SO-ISMCTS + POM underperforms. For player 2, SO-ISMCTS is outperformed by SO-ISMCTS + POM which is in turn outperformed by MO-ISMCTS. The three algorithms differ mainly in the assumptions they make about future play. SO-ISMCTS assumes that all actions are fully observable, which is both optimistic (I can respond optimally to my opponent's actions) and pessimistic (my opponent can respond optimally to my actions). SO-ISMCTS hence suffers from strategy fusion, since it is assumed the agent can act differently depending on information it cannot observe. In a phantom game, SO-ISMCTS + POM optimistically assumes that the opponent plays randomly. MO-ISMCTS's opponent model is more realistic: each opponent action has its own statistics in the opponent tree and so the decision process is properly modeled, but whichever action is selected leads to the same node in the player's own tree thus preventing the player from tailoring its response to the selected action. This addresses the strategy fusion problem which affects SO-ISMCTS.

Since player 1 has the advantage of moving first, it seems likely that these optimistic and pessimistic assumptions will have varying degrees of benefit and detriment to the two players. For example, a pessimistic player 2 algorithm may conclude (incorrectly) that the game is a loss, and so make poor decisions from that point. In short, we argue that solving the problem of strategy fusion (see Section III-B-I) is the key to strong play in the *Phantom (4, 4, 4)* game. Of the three ISMCTS variants, MO-ISMCTS is the most successful in overcoming strategy fusion. Indeed, the two SO-ISMCTS variants suffer more from the effects of strategy fusion than does determinized UCT.

One weakness of a cheating player is that it is overly pessimistic regarding the strength of its opponent. In particular, it assumes the opponent also cheats. In the *Phantom (4, 4, 4)* game, it often arises that the current state is a draw in the perfect information game but the noncheating player has insufficient information reliably to force the draw. In other words, there are states where a noncheating opponent is likely to choose an action that a cheating player would consider a mistake. If the cheating player could direct the game toward these states it would often win, but it sees no incentive to aim for these states in preference to any other state that leads to a draw. A noncheating player rarely suffers from this problem, as it generally lacks the information to identify the state as a draw in the first place. It should be noted that this never causes a cheating player to lose a game, only to draw a game that it could conceivably have won.

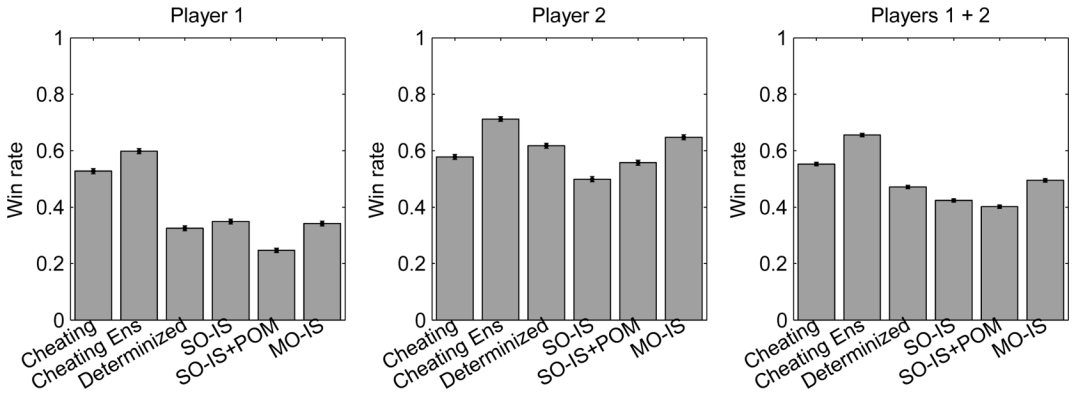


Fig. 10. Results of the playing strength experiment for the *Phantom (4, 4, 4)* game. In the "Player 1" graph, each algorithm indicated on the x-axis plays an equal number of games as player 1 against each algorithm as player 2, and the proportion of wins averaged over all player 2 algorithms is plotted. The "Player 2" graph is similar, with players 1 and 2 interchanged. The "Players $1 + 2$" graph averages these results for each algorithm regardless of player identity. In all cases, error bars show 95% confidence intervals.

TABLE I
DOU DI ZHU MOVE CATEGORIES

Name	Description
Solo	Any individual card, for example A or 2. It is also possible to play runs of sequential cards with length at least 5, for example 345678 or 89TJQKA.
Pair	Any pair of identically ranked cards for example 55 or 77. It is possible to play runs of sequential pairs with length at least 3, for example 334455 or TTJJQQKK.
Trio	Any three identically ranked cards for example AAA or 888. It is possible to play runs of sequential trios of any length, for example 444555 or TTTJJJQQQ. Each trio may also be played with a single *kicker* (single card or pair). In a sequence of trios kickers are either all pairs or all single cards, with all (or none) of the trios having kickers. For example 444555TJ (two single card kickers) or 999QQ (one pair as a kicker).
Quadplex	Any four identically ranked cards with two kickers of differing rank attached, for example 4444TJ or 999955KK.
Bomb	Any four identically ranked cards, for example 5555 or 2222.
Nuke	The red joker and the black joker together.

For this experiment the cheating algorithms played a total of 37 880 games, and did not lose a single game.

The above is a possible explanation for why cheating ensemble UCT outperforms cheating single-tree UCT. The former searches less deeply, and so its estimates for the game-theoretic values of states are less accurate. When the values of states are influenced more by random simulations than by the tree policy, there is a natural tendency to overestimate the value of states in which the opponent has more opportunities to make a mistake. In [10], we make similar observations on the propensity of noncheating players to make mistakes, and the benefit to a cheating minimax player of a tie-breaking mechanism that favors states from which the opponent has more suboptimal moves available.

VII. EXPERIMENTAL RESULTS FOR *DOU DI ZHU*

A. The Game

1) Background: Dou Di Zhu is a three-player gambling card game that originated in China, which falls into the class of ladder games. The name *Dou Di Zhu* translates into English as "Fight The Landlord" and is a reference to the class struggle during the Cultural Revolution in China where peasants were authorized to violate the human rights of their Landlords. In the original version of the game, studied in this paper, two players compete together against a third player, the *Landlord*. There are other versions of the game involving four and five players but these are less popular.

The game was only played in a few regions of China until quite recently, when versions of the game on the Internet have led to an increase in the popularity of the game throughout the whole country. Today *Dou Di Zhu* is played by millions of people online, although almost exclusively in China, with one website reporting 1 450 000 players per hour. In addition, there have been several major *Dou Di Zhu* tournaments including one in 2008 which attracted 200 000 players.

Dou Di Zhu is interesting from an AI perspective as it necessitates both competition (between the Landlord and the other two players) and cooperation (between the two non-Landlord players).

2) Rules: Dou Di Zhu uses a standard 52 card deck with the addition of a black joker and a red joker. We give a brief description of the rules here; a complete description can be found in [11]. Suit is irrelevant but the cards are ranked in ascending

order $3, 4, \ldots, T, J, Q, K, A, 2$. A bidding phase, which is not considered here, designates one of the players as the *Landlord*. The Landlord receives 20 cards dealt from a shuffled deck, while the other players receive 17 each. The goal of the game is to be the first to get rid of all cards in hand. If the Landlord wins, the other two players must each pay the stake to the Landlord. However, if either of the other two players wins, the Landlord pays the stake to both opponents. This means the two non-Landlord players must cooperate to beat the Landlord. The non-Landlord players do not see each other's cards, so the game cannot be reduced to a two-player game with perfect recall.

Card play takes place in a number of rounds until a player has no cards left. The Landlord begins the game by making a *leading play*, which can be any group of cards from their hand provided this group is a member of one of the legal move categories (see Table I). The next player can play a group of cards from their hand provided this group is in the same category and has a higher rank than the group played by the previous player, or may pass. A player who holds no compatible group has no choice but to pass. This continues until two players pass, at which point the next player may start a new round by making a new leading play of any category.

One exception to the rule that successive plays are of the same type is that a Bomb or a Nuke may be played at any point. Only a Bomb of higher rank or a Nuke can follow a Bomb, and no move can follow a Nuke. Some categories allow extra kicker cards to be played with the group which have no effect on the rank of the move being played. If a move with kickers is played, the next player must play a move in the same category with the same number of kickers.

Making a leading play is a good position to be in, allowing a player to choose a move type where he holds multiple groups, or holds a high-ranking group that opponents are unlikely to be able to follow. The two non-Landlord players also need to work together since they either both win or both lose.

3) Implementation: We do not consider the bidding phase, instead assigning an arbitrary player as the Landlord. This allows us to compare the strength of algorithms based on card play alone. Also determinization is carried out in the natural way, with all hidden cards from the point of view of a particular player being randomly reassigned amongst opponents.

The branching factor for leading plays is typically around 40, and for nonleading plays is much smaller. However, in situations where moves with kickers are available each combination of move and kicker must be considered as a separate move, leading to a combinatorial explosion in the branching factor for leading plays. It should be noted that this is a problem specific to *Dou Di Zhu* caused by the game mechanic of being able to attach kicker cards to a play. To ameliorate this, we use an approach similar to the move grouping approach of Childs *et al.* [39]: the player first chooses the base move and then the kicker, as two separate consecutive decision nodes in the tree.

B. Comparison of ISMCTS and Determinized UCT

Experiments for *Dou Dhi Zhu* we conducted previously [10] indicated that there was no significant difference in playing strength between ISMCTS and determinized UCT for *Dou Di Zhu*, when averaged across all deals. These comparisons were made across 1000 preselected deals (details in [9]) with ISMCTS, determinized UCT and cheating UCT as the Landlord against determinized UCT players.

It should be noted that the structure of the trees searched by determinized UCT and cheating UCT are the same on average (this is discussed further in Section VII-C). The most significant differences between the two are the access to hidden information and the consistency due to each UCT tree in the cheating player's ensemble corresponding to the same perfect information game. In deals where the cheating UCT player performed better than determinized UCT, and hence where hidden information and consistency in decision making had some impact, it was observed that ISMCTS performed better than determinized UCT. Since ISMCTS has no access to hidden information, this would suggest that the single tree approach is providing some of the same benefit the cheating ensemble player gains through consistency of the UCT trees searched. Indeed our previous results [10] suggest that hidden information is not often important in *Dou Di Zhu* and it is a highly unusual feature of this game, that knowledge of information often has little impact on players that use determinization.

Since the cards dealt is a significant deciding factor in the outcome of a game of *Dou Di Zhu*, the observed results may have been influenced by the small sample size of 1000 deals. This experiment was repeated with a larger sample of 5000 new randomly chosen deals, where each of the three algorithms played each deal 75 times. Note that all three variants of ISMCTS are equivalent for a game with fully observable moves. For this reason, only SO-ISMCTS was tested for *Dou Di Zhu*, and we refer to it simply as ISMCTS for the remainder of this section.

The overall win rate for determinized UCT was 43.6%, for ISMCTS it was 42.3%, and for cheating UCT it was 56.5%. The win rates are approximately the same as those we previously obtained [10]. This is unsurprising: the 1000 deals we originally selected [9] were chosen to be a good indicator of typical playing strength. Each deal was then put into one of three categories according to the difference in win rate between cheating UCT and determinized UCT. If cheating UCT outperformed determinized UCT (with 95% significance) the deal was put into the category > 0. If determinized UCT outperformed cheating UCT (also with 95% significance) the deal was put into the category < 0. All other deals were put into the category ≈ 0. There are 1421 deals in category > 0, 3562 in category ≈ 0, and the remaining 17 in category < 0.

The win rates of ISMCTS and determinized UCT for the categories > 0 and ≈ 0 are shown in Fig. 11. Since being in category < 0 is such a rare event, we do not give results for this category. In the deals in which cheating UCT is significantly better than determinized UCT (category > 0), the win rate of ISMCTS is significantly better than that of determinized UCT. These deals are arguably those where knowing hidden information has an impact on the game and also deals where determinization may suffer from strategy fusion issues. In deals where there is no significant difference between cheating and determinization, we observe that determinization is better than ISMCTS. This is also a similar result to that obtained in [10]. It is arguable that hidden information has little impact on these deals, for example, that

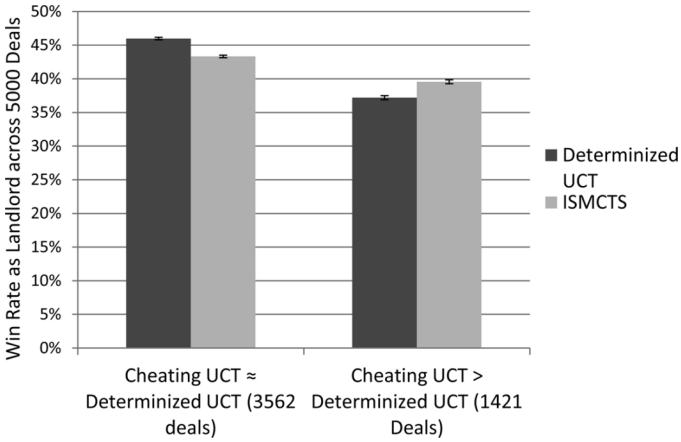


Fig. 11. Playing strength of ISMCTS and determinized UCT for *Dou Di Zhu*. The playing strength is shown for deals in which cheating UCT is better than determinized UCT (category >0) or where the two algorithms perform approximately equally (category ≈ 0).

one of the players has such a strong hand that a win is assured irrespective of what other players hold.

Despite the fact that overall the strength of ISMCTS and determinized UCT is approximately the same, there can be a great difference in the behavior of the two depending on which cards are dealt. In the 1421 deals where a cheating UCT player outperforms determinized UCT, so does ISMCTS on average. This suggests that ISMCTS may be benefiting from a lack of strategy fusion issues along with the cheating player in these deals. It is an unusual feature of *Dou Di Zhu* among hidden information games that having access to hidden information only provides a strong advantage in a minority of deals, and has little effect in the rest.

In 3562 deals there is no significant difference between cheating UCT and determinized UCT. In these deals, ISMCTS has a slightly lower win rate than determinized UCT. In these deals some factors other than hidden information and strategy fusion may be causing a detrimental effect on the performance of ISMCTS, but not on determinized UCT. The most significant difference between the two algorithms is the structure of the trees searched. The tree searched by ISMCTS offers several advantages over the determinization approach in general, but may be disadvantageous in certain deals. We investigate whether this difference is caused by branching factor in Section VIII.

C. Influence of Branching Factor on Playing Strength

The fact that there are some deals in which determinization outperforms cheating and many in which there is no difference between the two algorithms is a surprising feature of *Dou Di Zhu*, since intuitively the cheating player should have a strong advantage. One possible explanation for this is that branching factor has a large influence on the playing strength of these algorithms. In *Dou Di Zhu*, certain hands may have a large total number of moves available when making a leading play since there are many possible ways of choosing kicker cards to attach to a main group. Another feature of the game is that every move a player makes in the game is in the set of moves that player could make as a leading play from their starting hand. This set therefore forms an upper bound on the number of moves available in each state for a particular deal and if this set is large, there is likely to be many more nodes in the tree than if this set is small.

In the case that determinizations produce hands with completely different sets of moves, ISMCTS is at a disadvantage compared to determinized UCT. This is because ISMCTS will spend a lot of time adding new nodes near the root of the tree (since many determinizations will have unique moves that are not common to other determinizations) and consequently the statistics in the search tree will mostly be derived from random playouts near the root. On the other hand, determinizing players will be able to perform a deeper search for each determinization, since a large number of possible opponent moves will be ignored.

The following measurements were made for each of the 5000 deals tested in Section VII-B:

- the total number of moves the non-Landlord players would be able to make as a leading play from their starting hand (using the actual cards these players hold for this particular deal);
- the average of the above for 200 random determinizations of the deal (where the cards held by the non-Landlord players are randomized);
- the average number of unique leading plays for non-Landlord players that are discovered in 40, 250, and 1000 determinizations, i.e., after generating a certain number of determinizations how many possible unique leading plays have been seen for the non-Landlord players.

These measurements, averaged across all 5000 deals, are presented in Table II. It should be noted that these measurements are a function of the deal; the first measurement is exact for each deal, while the second depends on the sampled determinizations. These measurements were made only for the non-Landlord players since the playing strength experiments in Section VII-B were conducted from the point of view of the Landlord player. This means the algorithms tested always had the same number of branches at nodes where the Landlord makes a move, since the Landlord can see his cards in hand. The first measurement is an indicator for the number of branches that may be expected at opponent nodes for the cheating UCT player as the Landlord. Similarly, the second measurement indicates the number of branches for opponent nodes with determinized UCT as the Landlord. Both of these measurements are upper bounds, since if an opponent has played any cards at all then the number of leading plays will be smaller. The third, fourth, and fifth measurements indicate how many expansions ISMCTS will be making at opponent nodes after a certain number of visits, since a new determinization is used on each iteration. Again this measurement is an upper bound since only one move is actually added per iteration and if there were moves unique to a determinization which were never seen again, only one of them would be added to the tree.

As seen in Table II, from 1000 determinizations on average 1500 unique leading plays are seen and yet there are only approximately 88 unique leading plays for a particular determinization of a deal. What is apparent from these measurements is that there are a lot of moves available within the information

TABLE II
AVERAGES OF DIFFERENT MEASUREMENTS OF LEADING PLAYS FOR THE OPPONENTS IN *DOU DI ZHU* ACROSS 5000 DEALS

Measurement	**Result** (3 s.f.)
Total leading plays	87.7
Average leading plays per determinization	87.8
Average unique leading plays from 40 determinizations	754
Average unique leading plays from 250 determinizations	1230
Average unique leading plays from 1000 determinizations	1500

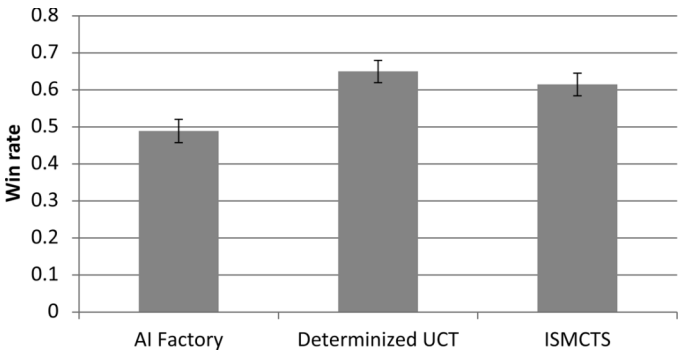


Fig. 12. Playing strengths of a commercial AI, determinized UCT, and ISMCTS for *Dou Di Zhu*. Error bars show 95% confidence intervals.

set and only a small number of them are available within each state in the information set. After just 40 determinizations, ISMCTS will on average have seen nearly ten times as many unique moves as there are per determinization. This means that at nodes in the information set tree where an opponent makes a leading play, node expansion will happen for many more simulations than if the moves were derived from one single determinization. At other nodes in the tree where the opponent must play a certain move type, any move that either player could play will appear as branches at nodes for both opponents. This suggests that nodes in the tree corresponding to an opponent making a leading play act as a bottleneck for ISMCTS; the algorithm very rarely explores beyond these nodes with only 10 000 simulations. With 250 simulations per determinization, it is likely that determinized UCT reaches a similar depth in the tree, which would explain why the overall performance of the two algorithms is broadly similar.

Another observation that can be made from these results is that the average number of leading moves for the actual state of the game and for each determinization is approximately the same. This is unsurprising since both measurements are derived from states constructed by randomly dealing unseen cards. This implies that cheating UCT and determinized UCT are searching trees of approximately the same size on average. Results from [10] suggest that the extra knowledge gained by cheating does not always provide a strong advantage.

We speculate that ISMCTS will strongly outperform determinized UCT if the former is equipped with some mechanism to handle the large number of moves accumulated near the root of the tree. Quite often the precise cards chosen as kickers are not of particular strategic importance; indeed attaching a card as a kicker is often a way of getting rid of a "useless" card that would be difficult to play otherwise. *Dou Di Zhu* also has many features which are unusual (and arguably pathological) for games of hidden information. As shown in Sections V and VI, the performance of ISMCTS is better in domains which lack this pathology.

D. Playing Strength Against a Commercial AI

To assess the absolute strength of determinized UCT and ISMCTS for *Dou Di Zhu*, we test them against a strong AI agent developed commercially by AI Factory Ltd.[2] This agent uses flat Monte Carlo evaluation coupled with hand-designed heuristics.

For implementation reasons the methodology of this experiment differs from that of other experiments in this section. For each game, the identity of the Landlord player is decided by a bidding phase. Since we concentrate on card play in this paper, all agents use the AI Factory agent's AI for bidding. We test three algorithms: the AI Factory agent, determinized UCT with 40 determinizations and 250 iterations per determinization, and ISMCTS with 10 000 iterations. Each plays 1000 games against two copies of the AI Factory agent. When all three agents are identical the expected number of wins for each is 500.

Results of this experiment are shown in Fig. 12. We see that both determinized UCT and ISMCTS significantly outperform the AI Factory agent. For reasons of computational efficiency, the AI Factory agent uses only a small number of Monte Carlo simulations. A further experiment was conducted to show that if we give the AI Factory player 100 times as many iterations as in the commercial version, the playing strength of all three agents is not significantly different. Thus, we can conclude in absolute terms that both determinized UCT and ISMCTS produce plausible, and indeed strong, play for *Dou Di Zhu*.

VIII. COMPUTATION TIME

It has been demonstrated that SO-ISMCTS and MO-ISMCTS offer advantages over determinized UCT, however both of these algorithms are more complex and computationally expensive. Experiments so far have performed a fixed number of iterations without consideration of the algorithm used. It could be that a simpler algorithm could perform more iterations in the same amount of time as a more complex algorithm and achieve a better result. This sort of comparison is dependent on the efficiency of the implementation of each algorithm and may be difficult to test in practice. Instead, it has been observed that MCTS algorithms can reach a point where additional simulations lead to increasingly diminishing returns in terms of playing strength. If two MCTS-based algorithms reach this point (independent

[2]www.aifactory.co.uk.

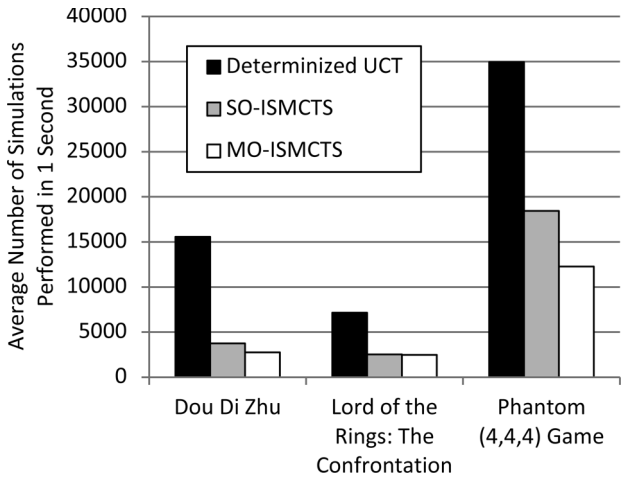


Fig. 13. Average number of simulations performed in 1 s by determinized UCT, SO-ISMCTS, and MO-ISMCTS on a desktop PC running Windows 7 with 6 GB of RAM and a 2.53-GHz Intel Xeon E5630 processor.

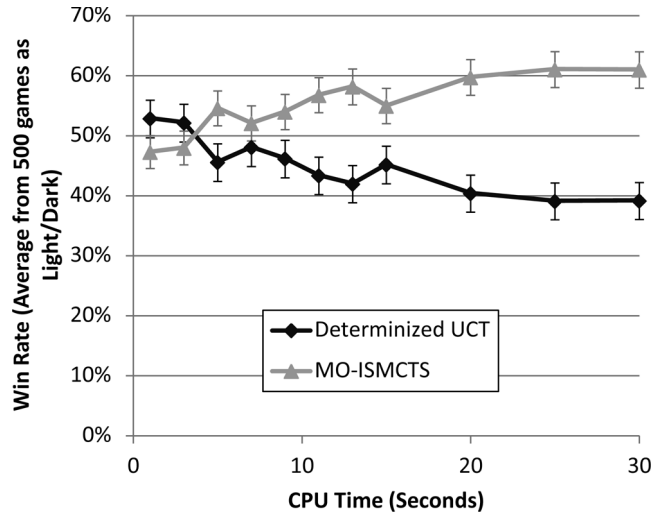


Fig. 14. Playing strength of determinized UCT and MO-ISMCTS for different amounts of decision time playing LOTR:C.

of the efficiency of implementations) and are using the same amount of time to make decisions, then their relative strengths should not change much as more time is used. In this section, it is demonstrated that with enough time per decision, the results obtained lead to the same conclusions as in previous experiments.

First, an experiment was conducted where determinized UCT, SO-ISMCTS, and MO-ISMCTS made the first decision for games of *Dou Di Zhu*, *LOTR:C*, and the *Phantom (4, 4, 4)* game. Each algorithm used 10 000 simulations (with 40 trees and 250 iterations per tree for determinized UCT) and the average time to make a decision was recorded from 25 trials for each game. This was performed on a desktop PC running Windows 7 with 6 GB of RAM and a 2.53-GHz Intel Xeon E5630 processor. These results were used to calculate the number of iterations each algorithm could perform in 1 s for each game. The results are presented in Fig. 13.

It is clear from Fig. 13 that SO-ISMCTS and MO-ISMCTS are two to four times slower than determinized UCT and also that the game being played has an impact on the amount of time it takes to perform an MCTS iteration. In order to compare algorithms based on the amount of decision time, it was important to remove factors which affect the execution time of the experiments: our experiments run on a cluster of heterogeneous machines, all of which have other processes running at the same time. The algorithms were tested with a fixed number of iterations corresponding to a certain amount of decision time, assuming the rate of iterations per second for each algorithm/game in Fig. 13. Approaches that build larger trees have increasing overheads per iteration, for example, due to MCTS selection being applied to more nodes in the tree. Assuming the rate of iterations per second from Fig. 13 is reasonable, since after a few hundred iterations the overheads increase slowly.

For the *Phantom (4, 4, 4)* game, the three algorithms already take less than a second to execute 10 000 MCTS iterations due to the simplicity of the game logic. However, for *Dou Di Zhu* and *LOTR:C*, it is clear that in 1 s of decision time SO-ISMCTS and MO-ISMCTS execute around a third the number of iterations that determinized UCT does. From Fig. 13, it is clear that the MO-ISMCTS implementation has some additional overheads, since it performed fewer iterations per second for *Dou Di Zhu* than SO-ISMCTS, although the two algorithms are equivalent for *Dou Di Zhu*, since it has no partially observable moves. The performance of these algorithms when decision time is a factor was investigated for all three games.

For *Dou Di Zhu*, determinized UCT was compared to SO-ISMCTS with 0.25–8 s of decision time. In each case, the algorithms played as the Landlord, with the non-Landlord players using determinized UCT with 40 determinizations and 250 iterations per determinization (as in Section VII). Playing strength was measured across the 1000 deals chosen previously [9]. The number of trees and iterations for determinized UCT was chosen for each total number of iterations to preserve the ratio of trees to iterations as 40/250. The relative playing strength of each algorithm was not significantly different to the results obtained in Section VII-B for any amount of decision time (although SO-ISMCTS appeared slightly weaker with less than 1 s of decision time). This supports the conclusion from Section VII-C that after reaching a certain depth, SO-ISMCTS spends many simulations expanding opponent decision nodes near the root of the tree and does not improve in playing strength.

For *LOTR:C*, MO-ISMCTS was compared to determinized UCT for 1–30 s of decision time where determinized UCT used one tree when playing as the Light player and a ratio of trees to iterations of 10/1000 when playing as the Dark player (these values were optimized in Section V). The two algorithms played each other as both the Dark player and the Light player 500 times. The results are presented in Fig. 14. It is clear that for 1 s of decision time, MO-ISMCTS is slightly inferior to determinized UCT, but when at least 3 s of decision time is used MO-ISMCTS is significantly stronger than determinized UCT. The results in Fig. 14 indicate that with a sufficient amount of decision time MO-ISMCTS offers a clear advantage over determinized UCT, which increases with increasing CPU time.

For the *Phantom (4, 4, 4)* game, determinized UCT, SO-ISMCTS, and MO-ISMCTS were compared for 0.25–5 s of decision time per move. For each pair of algorithms, 500 games were played with each algorithm playing as player 1

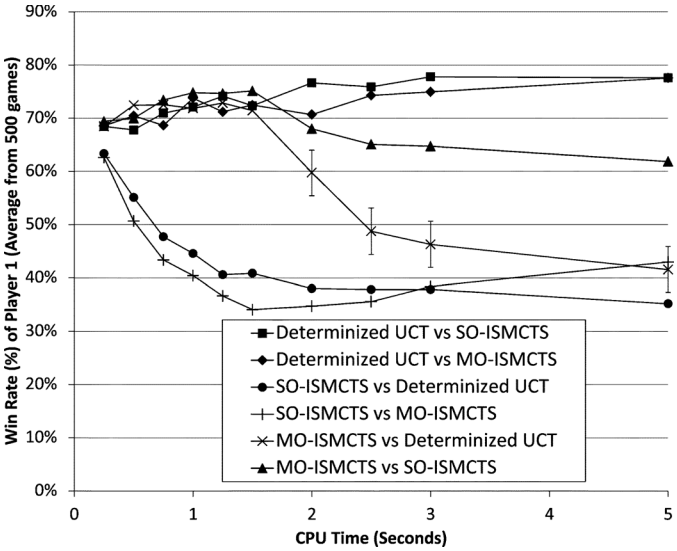


Fig. 15. Playing strength of determinized UCT and SO/MO-ISMCTS for different amounts of decision time playing the *Phantom (4, 4, 4)* game. For each pair of algorithms A versus B, win rates are presented for algorithm A where algorithm A makes the first move. Error bars show 95% confidence intervals for MO-ISMCTS versus determinized UCT for ≥ 2 s per decision, and are representative of the size of error bars for the other series.

and as player 2. The results are presented in Fig. 15. When the CPU time used is less than 1.5 s per move the results are not significantly different to those for 10 000 iterations presented in Section VI, with MO-ISMCTS slightly stronger than determinized UCT and clearly stronger than SO-ISMCTS. We also see the clear advantage of going first over going second. Above 1.5 s per move, the MO-ISMCTS algorithm continues to outperform SO-ISMCTS, in terms of results of games between these algorithms and performance against determinized UCT. However, both of these algorithms become relatively weaker than determinized UCT with increasing CPU time.

When using determinized UCT, implicitly we assume perfect information for both players. The SO-ISMCTS and MO-ISMCTS players do not assume knowledge of hidden information. However, SO-ISMCTS does make the pessimistic assumption that the opponent has perfect information. MO-ISMCTS improves on this, supposing that the opponent knows the root state but not the moves made by the MO-ISMCTS player. Two properties of the *Phantom (4, 4, 4)* game are important here: the game is a loss if the opponent observes the game state at a crucial moment, even if he does not cheat subsequently; and the game is simple enough that MO-ISMCTS with more than 1.5 s can search a significant proportion of the entire game tree. The pessimism of the assumption that the opponent knows some or all of the hidden information often leads SO-ISMCTS and MO-ISMCTS to conclude, incorrectly, that the game is a loss, and thus play randomly since all lines of play have the same reward value. Determinized UCT has the more balanced, although highly inaccurate, view that both players can see all hidden information.

Dou Di Zhu and *LOTR:C* are more complex than *Phantom (4, 4, 4)* so that it is not practical to search a substantial fraction of the whole tree within a reasonable time. Furthermore, both *Dou Di Zhu* and *LOTR:C* remain difficult to win even when hidden information is known. Hence, we do not see the reduction in playing strength for SO-ISMCTS and MO-ISMCTS with increasing CPU time. We conjecture that if the determinization approach of MO-ISMCTS was modified to also take the opponent's uncertainty into account, this effect would no longer occur in the *Phantom (4, 4, 4)* game. This is an interesting possible direction for future research.

IX. Conclusion

In this paper, we studied several variants of MCTS for games of imperfect information. Determinization is a popular approach to such games, but one with several shortcomings. Frank and Basin [5] identified two such shortcomings: strategy fusion (assuming the ability to make different decisions from different future states in the same information set) and nonlocality (ignoring the ability of other players to direct play towards some states in an information set and away from others). For MCTS, a third shortcoming is that the computational budget must be shared between searching several independent trees rather than devoting all iterations to exploring deeply a single tree.

To solve the problem of nonlocality requires techniques such as opponent modeling and calculation of belief distributions, which are beyond the scope of this paper. The solution we propose to the other two problems is the information set MCTS family of algorithms. These do not abandon determinization entirely, but use multiple determinizations to construct a single search tree (MO-ISMCTS searches multiple trees, but each tree is associated with a player rather than a determinization). Searching a single tree allows the entire computational budget to be devoted to searching it deeply. The sharing of information between determinizations also addresses the problem of strategy fusion: the algorithm cannot wrongly assume the ability to distinguish two future states if those states share a node in the tree. The SO-ISMCTS algorithm addresses the issue of wrongly assuming the player can distinguish between two states in an information set. The MO-ISMCTS algorithm additionally addresses the issue of wrongly assuming the player can distinguish between different partially observable moves made by an opponent.

We have considered three experimental domains: a complex board game (*Lord of the Rings: The Confrontation*), a simple phantom game [the *Phantom (4, 4, 4)* game], and a card game (*Dou Di Zhu*). In *Lord of the Rings: The Confrontation*, we have shown that ISMCTS significantly outperforms determinized UCT. We conjecture that one of the key benefits of ISMCTS over determinized UCT in *LOTR:C* is that the former is able to search more deeply in the game tree given the same computational budget. It seems to be a feature of *LOTR:C* for the Light player, and also for the Dark player to a lesser extent, that the ability to search the game tree more deeply and thus plan further ahead is more important than careful consideration of the hidden information. It is important to note that this increased search depth is *not* due to a lower branching factor, indeed the ISMCTS tree has a higher branching factor than the determinized UCT trees. Instead it is because the entire computational budget is devoted to a single tree in ISMCTS, as opposed to being shared between trees in determinized UCT.

There is no significant difference in playing strength between the three variants of ISMCTS for *LOTR:C*. This seems to suggest that strategy fusion is not a major factor in this game: the assumption that the identities of opponent characters are revealed when they move (i.e., that actions are fully observable) appears not to be exploited, or if it is exploited then this is not ultimately detrimental. However, in the *Phantom (4, 4, 4)* game MO-ISMCTS significantly outperforms the other ISMCTS variants. This is unsurprising: in a phantom game, SO-ISMCTS suffers from strategy fusion, and SO-ISMCTS + POM assumes random opponent play. The gap between determinized UCT and MO-ISMCTS is smaller than for *LOTR:C*, and indeed the SO-ISMCTS variants fail to outperform determinized UCT. Compared to *LOTR:C* and *Dou Di Zhu*, the *Phantom (4, 4, 4)* game tree is relatively small, and the game is more tactical than strategic, so issues such as search depth do not have such a dramatic effect on playing strength.

In *Dou Di Zhu*, the performance of ISMCTS is on a par with that of determinized UCT. The benefit to ISMCTS of devoting the entire computational budget to a single tree is negated by the fact that this tree's branching factor is often an order of magnitude larger than that of a determinized UCT tree. The union of legal action sets for all states in an information set is much larger than the action set for a single state in that information set

$$\left| \bigcup_{s' \in [s]^{\sim_i}} A(s') \right| \gg |A(s)| \tag{8}$$

which is not the case for *LOTR:C* or the *Phantom (4, 4, 4)* game. Indeed, it is sometimes the case that ISMCTS is encountering unseen actions from new determinizations on almost every iteration of the search, meaning that UCB never has the chance to exploit any action. With some mechanism for handling the large branching factor and controlling the expansion of the tree, we conjecture the playing strength of ISMCTS for *Dou Di Zhu* would be greatly increased.

ISMCTS allows a single tree to be searched for games of imperfect information. As a result, branching factor permitting, ISMCTS searches more deeply than determinized UCT with the same computational budget. This is true whether the computational budget is expressed in number of iterations or in CPU seconds, even taking into account that determinized UCT can execute more iterations per second. Furthermore, SO-ISMCTS addresses one source of strategy fusion issues and MO-ISMCTS additionally addresses another, providing an improved model of the decision making process compared to existing determinization methods. In domains where deep search is possible and beneficial or strategy fusion is detrimental, ISMCTS shows great promise. However, in domains such as *Dou Di Zhu*, where information sets have large numbers of legal moves and the effect of strategy fusion is not so clear, ISMCTS offers no immediate benefit over existing approaches.

Our ultimate goal is to develop ISMCTS into a general purpose algorithm for arbitrary games (and other decision problems) with stochasticity, imperfect information, and/or incomplete information and large state spaces. The next step toward this goal is to apply ISMCTS to wider classes of games and assess its strengths and weaknesses on those domains. Section X identifies some other directions for future work.

X. Future Work

We have empirically demonstrated the strength of the ISMCTS family of algorithms for several domains. It is clear that an enhanced version of ISMCTS should yield better playing strength, especially for domains such as *Dou Di Zhu* where there is a need for some mechanism to handle the large branching factor at opponent nodes. It remains to establish the theoretical properties of these algorithms and their potential for converging to game-theoretic solutions. MO-ISMCTS is arguably the most theoretically defensible of the three ISMCTS algorithms as it most accurately models the differences in information available to each player. A subject for future work is to conduct a full theoretical analysis of MO-ISMCTS, and investigate the situations under which it converges to an optimal (Nash equilibrium) policy. The SO-ISMCTS + POM algorithm currently assumes the opponent chooses indistinguishable moves at random, which is clearly incorrect as a decision model for the opponent. There is room for improvement in this aspect of the algorithm.

We observed in Section IX that ISMCTS suffers less from the effects of strategy fusion than determinization-based approaches. The first example in Section IV-E backs this up by showing that ISMCTS behaves correctly in a simple situation where strategy fusion occurs. It is possible to measure the presence of strategy fusion, albeit indirectly, for example, using the approach of Long *et al.* [18]. However, identifying situations in which strategy fusion occurs is difficult when games are large and nontrivial. Furthermore, strategy fusion is not the only factor in the performance of ISMCTS; for example, we have argued that it benefits from deeper search in *LOTR:C* but suffers from increased branching factor in *Dou Di Zhu*. Understanding situations in which strategy fusion is an important predictor of the success of ISMCTS, and developing methods to take advantage of this, is a subject for future work.

One limitation of the algorithms presented here is that they assume the opponents have access to the player's hidden information: when the player chooses a determinization to use during the search, it does not determinize its own cards or the locations of its own pieces. Essentially the searching player assumes a cheating opponent, which is a worst case assumption but does mean that the opponent's lack of information can never be exploited. Furthermore, the assumption will be particularly harmful in games where there are no strategies that offer a chance of winning against a cheating opponent. This problem was observed in Section VIII when larger amounts of CPU time were used for MO-ISMCTS playing the *Phantom (4, 4, 4)* game. However, the solution is not as simple as merely randomizing one's own information during determinization, as this destroys the player's ability to plan ahead beyond its current move (the searching player then assumes that it will forget its own information after the current turn). Addressing this issue, and particularly striking a balance between considering the actual situation and considering the other situations that the opponent

thinks are possible, is important in games where information hiding is a significant part of successful play. It is worth noting that there are games where one player has imperfect information but the other has perfect information; *Scotland Yard* is one example [40]. ISMCTS is well suited in such domains when playing as the player with imperfect information, since its assumption about the opponents' knowledge of the state of the game is correct.

None of our algorithms model belief distributions, instead sampling determinizations uniformly at random. It is well known in game theory that maintaining an accurate belief distribution is essential for optimal play in games of imperfect information, and this is also essential to solving the problem of nonlocality. In a phantom game, for instance, ISMCTS algorithms assume that the opponent has played randomly up to the current point, regardless of how correct or incorrect their assumptions about subsequent play may be. Integrating belief distributions into ISMCTS is an important direction for future work and should address the issue of nonlocality which arises through determinization.

APPENDIX
PSEUDOCODE FOR THE ISMCTS ALGORITHMS

This Appendix gives complete pseudocode for the ISMCTS algorithms, to complement the higher level pseudocode given in Section IV. In order to give a concrete implementation, the pseudocode given here illustrates UCB for selection, but other bandit algorithms may be used instead.

A. The SO-ISMCTS Algorithm

The following notation is used in this pseudocode:

- $c(v)$ = children of node v;
- $a(v)$ = incoming action at node v;
- $n(v)$ = visit count for node v;
- $n'(v)$ = availability count for node v;
- $r(v)$ = total reward for node v;
- $c(v,d) = \{u \in c(v) : a(u) \in A(d)\}$, the children of v compatible with determinization d;
- $u(v,d) = \{a \in A(d) : \nexists c \in c(v,d) \text{ with } a(c) = a\}$, the actions from d for which v does not have children in the current tree. Note that $c(v,d)$ and $u(v,d)$ are defined only for v and d such that d is a determinization of (i.e., a state contained in) the information set to which v corresponds.

---

function SO-ISMCTS($[s_0]^{\sim_1}, n$)
  create a single-node tree with root v_0 corresponding to $[s_0]^{\sim_1}$
  for n iterations **do**
    choose $d_0 \in [s_0]^{\sim_1}$ uniformly at random
    $(v,d) \leftarrow$ SELECT(v_0, d_0)
    if $u(v,d) \neq \emptyset$ **then**
      $(v,d) \leftarrow$ EXPAND(v,d)
    $r \leftarrow$ SIMULATE(d)
    BACKPROPAGATE(r, v)
  return $a(c)$ where $c \in \arg\max_{c \in c(v_0)} n(c)$

function SELECT(v,d)
  while d is nonterminal and $u(v,d) = \emptyset$ **do**
    select[3] $c \in \arg\max_{c \in c(v,d)} \left(\frac{r(c)_{\rho(d)}}{n(c)} + k\sqrt{\frac{\log n'(c)}{n(c)}} \right)$
    $v \leftarrow c$; $d \leftarrow f(d, a(c))$
  return (v,d)

function EXPAND(v,d)
  choose a from $u(v,d)$ uniformly at random
  add a child w to v with $a(w) = a$
  $v \leftarrow w$;$d \leftarrow f(d,a)$
  return (v,d)

function SIMULATE(d)
  while d is nonterminal **do**
    choose a from $A(d)$ uniformly at random
    $d \leftarrow f(d,a)$
  return $\mu(d)$

function BACKPROPAGATE(r, v_l)
  for each node v from v_l to v_0 **do**
    increment $n(v)$ by 1
    $r(v) \leftarrow r(v) + r$
    let d_v be the determinization when v was visited
    for each sibling w of v compatible with d_v, including v itself **do**
      increment $n'(w)$ by 1

B. The SO-ISMCTS + POM Algorithm

This pseudocode uses the notation from part A of the Appendix, with the following modifications:

- $a(v)$ = incoming move from player 1's point of view at node v;
- $c(v,d) = \{u \in c(v) : m(u) \in M_1(d)\}$, the children of v compatible with determinization d;
- $u(v,d) = \{m \in M_1(d) : \nexists c \in c(v,d) \text{ with } m(c) = m\}$, the moves from d for which v does not have children.

The following functions differ from the pseudocode given in part A of the Appendix.

---

function SELECT(v,d)
  while d is nonterminal and $u(v,d) = \emptyset$ **do**
    select[3] $c \in \arg\max_{c \in c(v,d)} \left(\frac{r(c)_{\rho(d)}}{n(c)} + k\sqrt{\frac{\log n'(c)}{n(c)}} \right)$
    choose $a \in A(d) \cap a(c)$ uniformly at random
    $v \leftarrow c$; $d \leftarrow f(d,a)$
  return (v,d)

function EXPAND(v,d)
  choose m from $u(v,d)$ uniformly at random
  add a child w to v with $a(w) = m$
  choose $a \in A(d) \cap m$ uniformly at random
  $v \leftarrow w$; $d \leftarrow f(d,a)$
  return (v,d)

---

[3]While the selection shown here is based on UCB, other bandit algorithms could be used instead.

C. The MO-ISMCTS Algorithm

This pseudocode uses the notation from part A of the Appendix, with the following modifications:

- v^i = a node in player i's tree;
- $a(v^i)$ = incoming move from player i's point of view at node v^i.

The following functions differ from the pseudocode given in part A of the Appendix.

---

function MO-ISMCTS($[s_0]^{\sim_1}, n$)
  for each player i **do**
    create a single-node tree with root v_0^i
  for n iterations **do**
    choose $d_0 \in [s_0]^{\sim_1}$ uniformly at random
    $(v^0, \ldots, v^\kappa, d) \leftarrow$ SELECT$(v_0^0, \ldots, v_0^\kappa, d_0)$
    if $u(v^{\rho(d)}, d) \neq \emptyset$ **then**
      $(v^0, \ldots, v^\kappa, d) \leftarrow$ EXPAND$(v^0, \ldots, v^\kappa, d)$
    $r \leftarrow$ SIMULATE(d)
    for each player i **do**
      BACKPROPAGATE(r, v^i)
  return $a(c)$ where $c \in \arg\max_{c \in c(v_0^1)} n(c)$

function SELECT$(v^0, \ldots, v^\kappa, d)$
  while d is nonterminal and $u(v^{\rho(d)}, d) = \emptyset$ **do**
    select[3] $c \in \arg\max_{c \in c(v^{\rho(d)}, d)} \left(\frac{r(c)_{\rho(d)}}{n(c)} + k\sqrt{\frac{\log n'(c)}{n(c)}} \right)$
    for each player i **do**
      $v^i \leftarrow$ FINDORCREATECHILD$(v^i, a(w))$
    $d \leftarrow f(d, a(c))$
  return $(v^0, \ldots, v^\kappa, d)$

function EXPAND$(v^0, \ldots, v^\kappa, d)$
  choose a from $u(v^{\rho(d)}, d)$ uniformly at random
  for each player i **do**
    $v^i \leftarrow$ FINDORCREATECHILD(v^i, a)
  $d \leftarrow f(d, a)$
  return $(v^0, \ldots, v^\kappa, d)$

function FINDORCREATECHILD(v^i, a)
  if $\exists c \in c(v^i)$ with $a(c) = a / \backsim_i$ **then**
    return such a c
  else
    create and **return** such a c

ACKNOWLEDGMENT

The authors would like to thank J. Rollason of AI Factory (www.aifactory.co.uk) for introducing them to *Dou Di Zhu*, for several useful and interesting conversations, and for providing the commercial AI opponent and test framework used in Section VII-D. They would also like to thank the volunteers who assisted with the playing strength experiment in Section V-D. Finally, the authors would like to thank the anonymous reviewers for their helpful and insightful comments.

REFERENCES

[1] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artif. Intell.*, vol. 175, no. 11, pp. 1856–1875, Jul. 2011.
[2] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. Eur. Conf. Mach. Learn.*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., Berlin, Germany, 2006, pp. 282–293.
[3] M. L. Ginsberg, "GIB: Imperfect information in a computationally challenging game," *J. Artif. Intell. Res.*, vol. 14, pp. 303–358, 2001.
[4] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower bounding Klondike Solitaire with Monte-Carlo planning," in *Proc. 19th Int. Conf. Autom. Plan. Sched.*, Thessaloniki, Greece, 2009, pp. 26–33.
[5] I. Frank and D. Basin, "Search in games with incomplete information: A case study using Bridge card play," *Artif. Intell.*, vol. 100, no. 1–2, pp. 87–123, 1998.
[6] BoardGameGeek, *Lord of the Rings: The Confrontation,* 2011 [Online]. Available: http://boardgamegeek.com/boardgame/3201/lord-of-the-rings-the-confrontation
[7] BoardGameGeek, *Stratego,* 2011 [Online]. Available: http://boardgamegeek.com/boardgame/1917/stratego
[8] J. W. H. M. Uiterwijk and H. J. van den Herik, "The advantage of the initiative," *Inf. Sci.*, vol. 122, no. 1, pp. 43–58, Jan. 2000.
[9] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Determinization in Monte-Carlo tree search for the card game Dou Di Zhu," in *Proc. Artif. Intell. Simul. Behav.*, York, U.K., 2011, pp. 17–24.
[10] D. Whitehouse, E. J. Powley, and P. I. Cowling, "Determinization and information set Monte Carlo tree search for the card game Dou Di Zhu," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, Korea, 2011, pp. 87–94.
[11] J. McLeod, *Dou Dizhu,* 2010 [Online]. Available: http://www.pagat.com/climbing/doudizhu.html
[12] J. Rubin and I. Watson, "Computer poker: A review," *Artif. Intell.*, vol. 175, no. 5–6, pp. 958–987, Apr. 2011.
[13] M. Shafiei, N. R. Sturtevant, and J. Schaeffer, "Comparing UCT versus CFR in simultaneous games," in *Proc. Int. Joint Conf. Artif. Intell. Workshop Gen. Game Playing*, Pasadena, CA, 2009 [Online]. Available: http://webdocs.cs.ualberta.ca/~nathanst/papers/uctcfr.pdf
[14] H. Kuhn, "A simplified two-person poker," in *Contributions to the Theory of Games*, H. Kuhn and A. Tucker, Eds. Princeton, NJ: Princeton Univ. Press, 1950, pp. 97–103.
[15] M. Ponsen, S. de Jong, and M. Lanctot, "Computing approximate Nash equilibria and robust best-responses using sampling," *J. Artif. Intell. Res.*, vol. 42, pp. 575–605, 2011.
[16] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling, "Monte Carlo sampling for regret minimization in extensive games," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2009, pp. 1078–1086.
[17] R. B. Myerson, *Game Theory: Analysis of Conflict*. Cambridge, MA: Harvard Univ. Press, 1997.
[18] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the success of perfect information Monte Carlo sampling in game tree search," in *Proc. Assoc. Adv. Artif. Intell.*, Atlanta, GA, 2010, pp. 134–140.
[19] E. K. P. Chong, R. L. Givan, and H. S. Chang, "A framework for simulation-based network control via hindsight optimization," in *Proc. IEEE Conf. Decision Control*, Sydney, Australia, 2000, pp. 1433–1438.
[20] M. Buro, J. R. Long, T. Furtak, and N. R. Sturtevant, "Improving state evaluation, inference, and search in trick-based card games," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, Pasadena, CA, 2009, pp. 1407–1413.
[21] J. Schäfer, "The UCT algorithm applied to games with imperfect information," Diploma, Otto-Von-Guericke Univ. Magdeburg, Magdeburg, Germany, 2008.
[22] J. Borsboom, J.-T. Saito, G. M. J.-B. Chaslot, and J. W. H. M. Uiterwijk, "A comparison of Monte-Carlo methods for phantom Go," in *Proc. BeNeLux Conf. Artif. Intell.*, Utrecht, The Netherlands, 2007, pp. 57–64.
[23] P. Ciancarini and G. P. Favini, "Monte Carlo tree search in Kriegspiel," *Artif. Intell.*, vol. 174, no. 11, pp. 670–684, Jul. 2010.
[24] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2009.
[25] D. Koller and A. Pfeffer, "Representations and solutions for game-theoretic problems," *Artif. Intell.*, vol. 94, no. 1–2, pp. 167–215, 1997.
[26] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2008, pp. 1729–1736.

[27] M. Ponsen, G. Gerritsen, and G. M. J.-B. Chaslot, "Integrating opponent models with Monte-Carlo tree search in poker," in *Proc. Conf. Assoc. Adv. Artif. Intell.: Inter. Decision Theory Game Theory Workshop*, Atlanta, GA, 2010, pp. 37–42.
[28] M. Richards and E. Amir, "Opponent modeling in Scrabble," in *Proc. 20th Int. Joint Conf. Artif. Intell.*, Hyderabad, India, 2007, pp. 1482–1487.
[29] O. Teytaud and S. Flory, "Upper confidence trees with short term partial information," in *Proc. Appl. Evol. Comput.*, C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, J. J. M. Guervós, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. N. Yannakakis, Eds., Torino, Italy, 2011, pp. 153–162.
[30] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Proc. Annu. Symp. Found. Comput. Sci.*, Milwaukee, WI, 1995, pp. 322–331.
[31] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.
[32] A. Fern and P. Lewis, "Ensemble Monte-Carlo planning: An empirical study," in *Proc. 21st Int. Conf. Autom. Plan. Scheduling*, Freiburg, Germany, 2011, pp. 58–65.
[33] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Proc. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2010, pp. 1–9.
[34] D. Auger, "Multiple tree for partially observable Monte-Carlo tree search," in *Proc. Evol. Games.*, Torino, Italy, 2011, pp. 53–62.
[35] C. J. Clopper and E. S. Pearson, "The use of confidence or fiducial limits illustrated in the case of the binomial," *Biometrika*, vol. 26, no. 4, pp. 404–413, 1934.
[36] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. van Rijswijck, "Games solved: Now and in the future," *Artif. Intell.*, vol. 134, no. 1–2, pp. 277–311, Jan. 2002.
[37] L. V. Allis, H. J. van den Herik, and M. P. H. Huntjens, "Go-Moku solved by new search techniques," *IEEE Comput. Intell. Mag.*, vol. 12, no. 1, pp. 7–23, Feb. 1996.
[38] F. Teytaud and O. Teytaud, "Lemmas on partial observation, with application to phantom games," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, Korea, 2011, pp. 243–249.
[39] B. E. Childs, J. H. Brodeur, and L. Kocsis, "Transpositions and move groups in Monte Carlo tree search," in *Proc. IEEE Symp. Comput. Intell. Games*, Perth, Australia, 2008, pp. 389–395.
[40] J. A. M. Nijssen and M. H. M. Winands, "Monte-Carlo tree search for the game of Scotland Yard," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, Korea, 2011, pp. 158–165.

Peter I. Cowling (M'05) received the M.A. and D.Phil. degrees from Corpus Christi College, University of Oxford, Oxford, U.K., in 1989 and 1997, respectively.

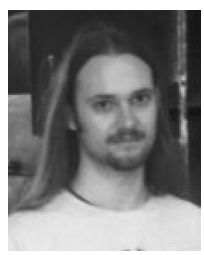
He is a Professor of Computer Science and Associate Dean (Research and Knowledge Transfer) at the University of Bradford, Bradford, U.K., where he leads the Artificial Intelligence Research Centre. In September 2012, he will take up an Anniversary Chair at the University of York, York, U.K., joined between the Department of Computer Science and the York Management School. His work centers on computerized decision making in games, scheduling and resource-constrained optimization, where real-world situations can be modeled as constrained search problems in large directed graphs. He has a particular interest in general-purpose approaches such as hyperheuristics (where he is a pioneer) and Monte Carlo tree search (especially the application to games with stochastic outcomes and incomplete information). He has worked with a wide range of industrial partners, developing commercially successful systems for steel scheduling, mobile workforce planning, and staff timetabling. He is a director of two research spinout companies. He has published over 80 scientific papers in high-quality journals and conferences.

Prof. Cowling is a founding Associate Editor of the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI FOR GAMES. He has won a range of academic prizes and "best paper" awards, and given invited talks at a wide range of universities and conference meetings.

Edward J. Powley (M'10) received the M.Math. degree in mathematics and computer science and the Ph.D. degree in computer science from the University of York, York, U.K., in 2006 and 2010, respectively.

He is currently a Research Fellow at the University of Bradford, Bradford, U.K., where he is a member of the Artificial Intelligence Research Centre in the School of Computing, Informatics and Media. His current work involves investigating Monte Carlo tree search (MCTS) for games with hidden information and stochastic outcomes. His other research interests include cellular automata, and game theory for security.

Dr. Powley was awarded the P B Kennedy Prize and the BAE Systems ATC Prize.



Daniel Whitehouse (S'11) received the M.Math. degree in mathematics from the University of Manchester, Manchester, U.K., in 2010. He is currently working toward the Ph.D. degree in artificial intelligence in the School of Computing, Informatics and Media, University of Bradford, Bradford, U.K.

He is a member of the Artificial Intelligence Research Centre, University of Bradford. His Ph.D. work is funded as part of the EPSRC project "UCT for games and Beyond" and is investigating the application of Monte Carlo Tree Search methods to games with chance and hidden information.