Emergent bluffing and inference with Monte Carlo Tree Search

Peter I. Cowling
Department of Computer Science
York Centre for Complex Systems Analysis
University of York, UK
Email: peter.cowling@york.ac.uk

Daniel Whitehouse
Hebe Works
Leeds, UK
Email: dan@hebeworks.com

Edward J. Powley
The MetaMakers Institute
Academy for Innovation and Research
Falmouth University, UK
Email: edward.powley@falmouth.ac.uk

***Abstract*—In many card and board games, players cannot see the whole game state, with different players seeing different parts of the state. In such games, gathering of information (*inference*) is a key strategic aspect, to which information hiding (*bluffing*, among other techniques) is an important countermeasure. *Monte Carlo Tree Search (MCTS)* is a powerful general-purpose technique for decision making in games. MCTS rose to prominence through successes in combinatorial board games such as Go, but more recently has demonstrated promise in card, board and video games of incomplete information. MCTS can construct robust plans in stochastic environments (making it strong in some games), but in its vanilla form is unable to infer or bluff (making it weak in games where this is a central feature).**

In this paper, we augment MCTS with mechanisms for performing inference and bluffing. Like all algorithms based on game tree search, MCTS implicitly constructs a model of the opponents' decision processes. We show that this model can be repurposed to perform an approximation of Bayesian inference. We also obtain bluffing behaviour by *self-determinization* (introducing "impossible" worlds into the agent's pool of sampled states). We test our algorithms on *The Resistance*, a popular card game based around hidden roles.

I. Introduction

Much of the classical study of game AI focused on games of *perfect information* such as Checkers, Chess and Go: the state of the game and the actions of the opponent are fully observable. More recent studies have focused on games of *imperfect information* such as Bridge and Poker: players can make only partial observations of the game state and opponent actions. In particular, the player can narrow down the current state of the game to a set of possibilities, called an *information set*, but generally not to a single state.

An important strategic aspect of imperfect information games is the gathering and hiding of information. In many games players can perform *inference* to determine which states in the information set are more likely than others, based on opponent actions. For example in Poker, if Alice makes a large bet then her opponents might infer that she has a strong hand. However if players are making reasonable inferences, this can be exploited by *bluffing*: deliberately choosing misleading actions. For example in Poker, Alice might make a large bet in an attempt to convince her opponents that her hand is stronger than it is. Inference can also be countered by *information hiding*: choosing actions intended not to give away any information. For example in Poker, if Alice bets conservatively then Bob may be left unsure as to the strength of her hand.

Information Set Monte Carlo Tree Search (ISMCTS) [1] is an AI technique for games of imperfect information. It is based on two dimensions of Monte Carlo sampling: sampling of states from the current information set, and sampling of action sequences in the game tree. ISMCTS has shown itself effective across a range of board and card games [1], [2], [3], [4], however it lacks the ability to perform inference or to bluff. This paper investigates how these capabilities can be added to ISMCTS. ISMCTS works by sampling *determinizations* (states from the current information set) and playing them out through the MCTS tree. We show how inference can be achieved by recording the frequency with which each determinization visits each part of the tree, i.e. by tracking information which in standard ISMCTS would be discarded. We achieve bluffing by introducing *self-determinization*: allowing states to be sampled that are impossible from the search agent's point of view, but possible from the opponent's point of view. This is sufficient for bluffing to emerge given enough computational time; however we also introduce a mechanism to encourage bluffing behaviour within a smaller budget.

The structure of this paper is as follows. Section II presents relevant background material on ISMCTS and existing approaches to inference and bluffing. Section III introduces the game used for experiments. Section IV details the variant of ISMCTS used in this paper, which uses a slightly different tree structure than in previous work on ISMCTS. In Section V we show how existing information in the ISMCTS trees can be reused for inference, and Section VI introduces several techniques that can induce bluffing behaviour. Section VII presents experimental results. Finally Section VIII summarises the results and discusses directions for future work.

II. Background

A. Information Set MCTS

Perfect Information Monte Carlo (PIMC) [5] is an AI technique for games of imperfect information. PIMC works by sampling *determinizations* (states from the current information set), treating each as an instance of a perfect information game, and aggregating the results to make a decision in the original game. PIMC was introduced by Ginsberg [6], and has been combined with MCTS by several authors [7], [8], [9], [10], [11].

Frank and Basin [12] identify two shortcomings of PIMC. The first is *strategy fusion*. A determinization is treated as a perfect information game, so all information is fully observable. This means the AI can choose actions contingent on information that is not available. For example a PIMC player for Poker can decide whether to bet or fold based on the cards in its opponents' hands. The second problem is *non-locality*. PIMC samples states from the current information set with uniform probability. However some states may be more likely than others, given that opponents are rational. For example in Poker, a player betting a large amount suggests the player's hand is more likely strong than weak (assuming they are not bluffing), but a PIMC player fails to account for this.

Information Set Monte Carlo Tree Search (ISMCTS) [1] is a variant of MCTS for games of imperfect information (for an overview of MCTS refer to [13]). ISMCTS is based on the PIMC technique, but does not treat the determinizations separately. Instead, each MCTS iteration begins by sampling a determinization, which is used to guide the current playout and then discarded. Thus instead of analysing each determinization separately and aggregating the results afterwards, ISMCTS aggregates results in the same tree throughout the search. ISMCTS solves the problem of strategy fusion when taking a PIMC approach to MCTS.

B. Inference

In a game of imperfect information, a player knows the current information set but not the current state. A *belief distribution* assigns a probability to each state in the information set, specifying how likely that state is to be the true state of the game. *Inference* is the process of updating this belief distribution in response to opponent actions, with respect to an *opponent model* which predicts how the opponent would act upon different observations of hidden information. There are deductions which human players often refer to as "inference" but which do not depend on opponent policy, for example those arising from card counting or other probabilistic reasoning. In this paper we limit *inference* to deductions which stem from opponent actions with respect to an opponent model, because probabilistic deductions are already within the capabilities of ISMCTS [14].

The most common theoretical framework for inference in games is *Bayesian inference*. Consider a state s in the current information set I. There is a *prior belief* $P(s)$, the probability that s is the actual state of the game. Upon observing an opponent action a, the *posterior belief* $P(s|a)$ is the probability that s was the actual state of the game given that a was played from that state. If it is assumed that the opponent's (mixed) policy is known, then $P(a|s)$ is the probability that the opponent plays action a from state s; this assumed policy is the opponent model. Given the prior belief and the opponent model, Bayes' theorem can be applied to obtain the posterior belief:

$$P(s|a) = \frac{P(a|s)P(s)}{\sum_{u \in I} P(a|u)P(u)} \,. \tag{1}$$

There exist several approaches to integrating inference (Bayesian or otherwise) with MCTS for particular games. Most existing approaches for applying inference with MCTS use an opponent model which is computed offline, but used to determine a distribution over states at runtime. For example Ponsen et al [15] learn an opponent model for Poker which is used to influence the cards dealt and select actions for the opponent. Buro et al [16] apply a similar method to the card game Skat, where a learned opponent model is used to estimate the likely cards held by each player. Whitehouse et al [2] use a knowledge base of hand-coded rules with automatically tuned weights to influence the distribution of cards dealt to players depending on factors such as their bid.

There has been less work on methods which compute an opponent model online, though there is a natural synergy between this approach and MCTS since MCTS already models the decisions of opponents. If it is assumed that MCTS is a good opponent model, an inference model can be built by using tree statistics. For example if every state in an information set is sampled sufficient times, it is possible to measure the proportion of times each action was selected for an opponent from each state and update the belief state using Bayes' theorem. This approach has been successful in applications such as Scrabble [17]. Silver and Veness [14] propose an approximate method using a particle filter, which approaches a correct belief state as more particles are used. One drawback of this method is that particle deprivation may occur as the particles disperse in a large state space and a particle reinvigoration method is needed.

III. The Resistance

The Resistance is a social deduction game designed by Don Eskridge and first published in 2009 [18]. This section describes the rules for the 5-player game; variants for up to 10 players also exist. Each player is assigned a role at random, such that there are two *spies* and three *non-spies*. Each player knows her own role; additionally, each spy knows the identity of the other spy. The game is played over five rounds, each of which can be won by the spy team or the non-spy team. The aim of the game is to win three out of five rounds for your team. Each round has the following structure:

Team choice. One player is designated the *leader*; this passes around the players in turn. The leader chooses two (in rounds 1 and 3) or three (in rounds 2, 4 and 5) players, possibly including himself, to form a mission team.

Voting. All five players vote simultaneously to approve or reject the team. If a majority approve the team, the mission proceeds. Otherwise, the next player becomes leader and chooses a team. If five team choices are rejected in a single round, the spies automatically win the game.

Mission. Each player on the mission team chooses a card in secret: *success* or *failure*. A non-spy player must choose the success card; a spy player has a free choice. The cards are shuffled and revealed, so that only the number of successes and failures are known, and not which team member played which card. If all cards are successes, the non-spies win the mission; if any card is a failure, the spies win the mission.

If the non-spies knew the identities of the spies, they could force a win: only choose mission teams with no spies, and use their majority status to vote down any deviation from this. Thus a non-spy player must determine who her fellow non-spies are, and signal her own non-spy status; conversely, a spy player must obfuscate his identity to win the trust of the non-spies. It is always disadvantageous for a non-spy to appear spy-like, so non-spies do not need to bluff. Spies have perfect information, so do not need to perform inference. Thus we can study inference and bluffing in isolation to each other. This is in contrast to most games of hidden information, where success generally requires a mixture of inference and bluffing.

The number of states per information set is small, especially compared to the combinatorially large information sets commonly seen in other games. The only piece of hidden information is the identity of the spies, for which there are $\binom{5}{2} = 10$ possibilities. A player knows her own role and spies know all roles, so the size of an information set is $\binom{4}{2} = 6$ for a non-spy player or 1 for a spy player. Furthermore the roles do not change during the game.

IV. Many Tree ISMCTS

The original motivation for ISMCTS was to search trees of information sets. The multi-observer variant of ISMCTS [1] comes close to this goal, but nodes in opponent trees represent unions of information sets rather than single information sets. In particular, opponent decisions cannot depend on information that is known to the opponent at the root but unknown to the searching player.

To address this, we introduce *Many Tree ISMCTS (MT-ISMCTS)*. MT-ISMCTS constructs several trees per player, one for each information set that the player can possibly observe at the current time. Each iteration samples a determinization, and selects the corresponding tree for each player. For example in The Resistance, if I am player 1 and I am not a spy then there are four possible information sets that player 2 may observe: (i) player 2 is not a spy; (ii) players 2 and 3 are spies; (iii) players 2 and 4 are spies; (iv) players 2 and 5 are spies. MT-ISMCTS constructs a separate tree for each of these possibilities, and a determinization dictates which combination of trees is used for each iteration. Pseudocode for MT-ISMCTS is given in Algorithm 1. Assuming that the game satisfies properties set out in [1] (which ensure that the revelation of information during the game always occurs by observation of actions by players or by a virtual "environment" player, not by direct observation of the current information set), each node in MT-ISMCTS corresponds to a single information set. If the game does not contain transpositions, the correspondence between nodes and information sets is one-one.

In Algorithm 1, the mapping U stores root nodes only. It would be possible to store all nodes in this mapping and do away with the tree structure entirely: at lines 16 and 22, instead of looking up the child of u_i for action a, look up the node in U corresponding to information set $[d]^i$. Then U would also function as a transposition table, ensuring one node per information set. However this requires multiple hash table lookups per MCTS iteration, which is likely to be more computationally expensive than maintaining a tree structure.

Algorithm 1 The MT-ISMCTS algorithm.

```
for player $i$, state $s$, action $a$ and node $u_i$
    $[s]^i$ is $i$'s information set containing $s$
    $\rho(s)$ is the player about to act in $s$
    $\langle a\rangle^i$ is $i$'s observation of $a$
    $s.a$ is the state resulting from applying $a$ in $s$
    $u_i.\langle a\rangle^i$ is the child of $u_i$ through edge $\langle a\rangle^i$

1:  // Search from information set $I$, and return an action
2:  function MT-ISMCTS($I$)
3:      $U$ := empty mapping from information sets to nodes
4:      while time remains do
5:          sample a determinization $d$ of $I$
6:          for each player $i$ do
7:              if $U$ contains key $[d]^i$ then
8:                  $u_i := U\left([d]^i\right)$
9:              else
10:                 $U\left([d]^i\right) := u_i :=$ new node

11:         // Selection
12:         while $u_{\rho(d)}$ fully expanded and $d$ nonterminal do
13:             $a := \textsc{BanditAlgorithm}(u_{\rho(d)})$
14:             $d := d.a$
15:             for each player $i$ do
16:                 $u_i := u_i.\langle a\rangle^i$

17:         // Expansion
18:         if $u_{\rho(d)}$ is not fully expanded then
19:             $a :=$ random unexpanded action from $u_{\rho(d)}$
20:             $d := d.a$
21:             for each player $i$ do
22:                 if $u_i.\langle a\rangle^i$ does not exist then
23:                     $u_i.\langle a\rangle^i :=$ new node

24:         // Simulation
25:         apply random actions until $d$ is a terminal state

26:         // Backpropagation
27:         for each player $i$ do
28:             for each node $v_i$ visited this iteration do
29:                 update $v_i$'s win and visit statistics

30:     return most visited action from $U(I)$
```

The drawback of using additional trees is that the ISMCTS learns more slowly, since not every tree is updated on each iteration. The advantage is that the additional trees improve the opponent model of ISMCTS, which is particularly important if the opponent model is to be exploited for inference. Therefore the additional trees can be seen as a trade-off between a fast learning rate and a better opponent model. It is likely that the learning rate can be increased by using enhancements that share knowledge between trees [3]. As presented here, MT-ISMCTS is only applicable to games in which the number of states per information set is small.

This rules out many games of imperfect information, where information sets are combinatorially large. Some mechanism for "bucketing" information from similar determinizations would be required to apply MT-ISMCTS to such games; this is a subject for future work.

The pseudocode in Algorithm 1 does not specify the bandit algorithm to be used (line 13). The UCB1 algorithm [19] is by far the most commonly used bandit algorithm in the MCTS literature [20], [13]. However in this paper we use UCB-Tuned [19], which has been shown to outperform UCB1 in several domains [21], [22] and has no parameters to tune. Preliminary experiments showed a marginal benefit (around a 3–4$\%$ increase in win rate) from using UCB-Tuned instead of UCB1 in The Resistance.

V. Particle filter inference

The idea of *particle filtering* in an ISMCTS-like algorithm was suggested for single-agent POMDPs by Silver and Veness [14]. The idea is to record the frequency with which each determinization (particle) reaches each node in the tree, and use this to update the belief distribution in response to observed moves.

Consider a tree with root node u and a child v corresponding to action a. Consider also a determinization d for the root information set where d is chosen with probability $P(d)$, the prior belief that the current state is d. Then the number of visits to child v with determinization d as a fraction of the total number of visits to v is an estimate of $P(d|a)$: the probability that d was the current determinization given that action a was selected from the root. Hence this method allows the posterior belief to be sampled empirically without need for Bayes' rule (Equation 1).

In many games, there are too many states per information set to enumerate. Silver and Veness [14] sample a small number of determinizations to use as particles, and introduce a *reinvigoration* mechanism to counteract the depletion of the particle set as the tree is descended. In The Resistance the number of states per information set is small, so no such mechanism is needed: our pool of "particles" is in fact a complete enumeration with associated probabilities. We introduce a *particle filter inference* mechanism for MT-ISMCTS. We define a mapping c from (node, determinization) pairs to integers: $c(u_i, d)$ is the number of iterations that visited u_i and used determinization d. These counts begin at zero and are incremented during backpropagation.

The player also maintains a belief distribution ϕ over determinizations, separate from the tree, which is used to sample determinizations (Algorithm 1 line 5). At the beginning of the game, this distribution is chosen to match the distribution of initial game states (which often means a uniform distribution). The belief distribution is updated every time the player observes an action, according to the tree constructed the last time the player executed the MT-ISMCTS algorithm. Suppose that u is the node in player i's MT-ISMCTS tree which corresponds to the current situation. An opponent performs an action a, which player i observes as $\langle a \rangle^i$. Let $v = u.\langle a \rangle^i$ be the corresponding child of u, using the same notation as Algorithm 1. Let $n(v)$ be the visit count for v, and let N be the number of MT-ISMCTS iterations that player i executed for the last decision. For each determinization d, we update the belief distribution ϕ by

$$\phi(d) := \left(1 - \frac{n(v)}{N}\right)\phi(d) + \frac{n(v)}{N}\frac{c(v,d)}{n(v)}\,. \tag{2}$$

The update is weighted by a factor $\frac{n(v)}{N}$: the number of visits to the new node v divided by the total number of iterations. If relatively few iterations visited v then the first term dominates, and $\phi(d)$ remains close to its current value. If v was visited many times, $\phi(d)$ moves towards the second term: the number of visits to v which used determinization d, as a proportion of the total number of visits to v. As noted above, this is an approximation of $P(d|a)$.

The idea of weighting the update by $\frac{n(v)}{N}$ is that infrequently visited branches should not bias the distribution too much, as otherwise the inference may become brittle. Also since $\frac{n(v)}{N} < 1$, the frequency distribution never completely replaces the belief distribution, so a determinization probability that starts as nonzero will stay nonzero. In some games, an observed action can signal that certain determinizations are impossible. An example of this in The Resistance is mentioned in Section III: if a mission contains a failure card, then there must be a spy on the team. Our inference system detects this, and sets $\phi(d) = 0$ for those determinizations.

VI. Self-determinization and bluffing

ISMCTS traditionally only considers determinizations in the searching player's current information set, i.e. only states that are compatible with the searching player's observations. This makes bluffing and information hiding impossible: the agent assumes that whatever it can see, its opponents can also see. To solve this we need *self-determinization*: we need to sample determinizations that we know to be false, but which opponents may be considering. This models the uncertainty an opponent has about the player's information and allows the player to exploit that uncertainty.

When using MT-ISMCTS, we can safely introduce self-determinization without polluting the decision tree with false information: iterations using determinizations outside the player's information set do not descend or update the decision tree, but the fact that they descend and update the opponents' trees means that they do factor into the opponent model. However a significant amount of time is potentially "wasted" considering lines of play that are known to be impossible, for the sake of more accurate opponent modelling. Thus a balance must be struck between searching "true" determinizations (in the root information set) and self-determinizations (not in the root information set). This gives rise to a new problem: any bias towards true determinizations will also influence the opponent trees, causing the hidden information to "leak" into the modelled opponent policy and thus lessening the effectiveness of self-determinization.

A self-determinizing player needs two belief distributions: the usual belief distribution ϕ used to select true determinizations, and a distribution ψ used to select self-determinizations. The latter is the player's model of what the opponent's have inferred so far. Both are updated as described in Section V, but ψ is initialised and updated without knowledge of the player's hidden information. In The Resistance, ϕ is initialised by setting the probabilities of spy configurations containing the player to 0 (if the player is not a spy) or the probability of the actual spy configuration to 1 (if the player is a spy), whereas ψ is initialised by setting all spy configurations to equal probability, regardless of whether they contain the player and whether the player is a spy.

The distribution ϕ is updated according to the player's decision tree. If ψ were updated using the decision tree, information about the player's hidden information would leak into ψ and the benefit of self-determinization would be defeated. Instead ψ is updated by merging all the trees from the player's perspective, including the decision tree and all trees for self-determinizations. The visit counts in the merged tree are obtained by summing the visit counts across all trees. This can be thought of as a tree "averaged" across all self-determinizations, or the tree that would be built by an external observer who could not see any of the players' hidden information.

Bluffing is in general a strategic decision, i.e. the benefits of bluffing at one point in the game are often not realised until a much later point, and occur indirectly as a result of influencing the opponent policy. In The Resistance it is obvious to a human player that a spy should conceal her identity (even without the context of the game's theme), but to discover this purely by tree search is difficult. This suggests that game-specific knowledge is required to encourage bluffing. Such knowledge has been used in MCTS players for the hide-and-seek game Scotland Yard [23], where the hiding player benefits from favouring moves which increase the seekers' uncertainty about his location.

We test a number of approaches to self-determinization:

TRUEONLY. Do not perform self-determinization; only sample determinizations from ϕ. This method is included as a benchmark.

PURE. Simply run ISMCTS with determinizations sampled from ψ, i.e. all self-determinizations. Some of these determinizations will happen to be true determinizations and thus update the decision tree, but otherwise no special effort is made to search the decision tree.

SPLIT. Split the time budget (expressed as a number of iterations) in half. Sample determinizations from ψ for the first half, and from ϕ for the second half. The first phase seeds the trees with statistics from self-determinizations, and the second searches only true determinizations but benefits from pre-seeded opponent trees.

TWOSTEP. Similar to SPLIT, proceed in two phases and sample from ψ in the first phase. At the end of the first phase, the opponent trees are fixed and used to define a mixed policy where the probability of selecting a child node is proportional to its number of visits. The second phase samples determinizations from ϕ, but instead of performing bandit selection at opponent decision nodes, the mixed policy from the first phase is used and backpropagation does not update opponent trees. This preserves the integrity of the opponent model, preventing true information from polluting it. The mixed opponent policy is required as continuing to use UCB-Tuned without updating the opponent trees would lead to deterministic decisions at opponent nodes.

BLUFF. As SPLIT, but with a different mechanism for choosing the action to play at the end of the search. Denote the mean reward for an action a from the root of the decision tree by μ_a and the standard deviation by σ_a. Choose a^* with maximal number of visits from the root of the decision tree, as usual. Let

$$A^* = \{a \ : \ \mu_{a^*} - \mu_a < \min(\sigma_{a^*}, \sigma_a)\} \ , \tag{3}$$

i.e. A^* is the set of actions whose average reward is within one standard deviation of the reward for the best action. Now for each action in A^*, sum the number of visits from the root across all the current player's trees (i.e. the decision tree and all trees corresponding to self-determinizations), and play the action for which this sum is maximal. In other words, the chosen action is the most visited across all self-determinizations that is not significantly worse than the most visited action in true determinizations; the bluff that is not significantly worse than the best non-bluff.

PURE is the most theoretically sound method (and similar to Smooth UCT, which can converge to a Nash-equilibrium [24]) but "wastes" most of its iterations by not updating the actual decision tree. SPLIT provides a compromise by spending at least half its time searching true determinizations, but risks leaking information into the opponent model. TWOSTEP prevents this leakage of information by not updating the opponent model at the end of the first step, at the cost of the opponent model being weakened due to having fewer iterations. Any bluffing behaviour observed in these three methods is an emergent property of the search. BLUFF explicitly integrates bluffing behaviour into the move selection, by choosing a move which is plausible when the opponent model "cheats" (searching true determinizations) but was preferred over other moves when the opponent model does not "cheat". In other words, amongst the moves which are the most robust against opponent's inference, choose the move which best exploits their lack of knowledge.

VII. EXPERIMENTS

A. Inference and self-determinization methods

In this section we test various MT-ISMCTS players for The Resistance, namely non-spy players using the inference methods described in Section V and spy players using the self-determinization methods described in Section VI. Two non-spy players are tested: one sampling determinizations uniformly at random, and one using the particle filter inference mechanism described in Section V. Both of these players use $20\,000$ iterations per decision, and neither player uses self-determinizations. For the spies, we test players using each of the variants of self-determinization in Section VI. The TRUEONLY player uses $20\,000$ iterations per decision, whereas the others use $40\,000$ iterations. SPLIT, TWOSTEP and BLUFF perform $20\,000$ iterations per phase. For each tested configuration 1000 games were played, each with 5 players: each of the 10 possible spy configurations was tested 100 times, each time with all the spy players using one algorithm and all the non-spy players using another.

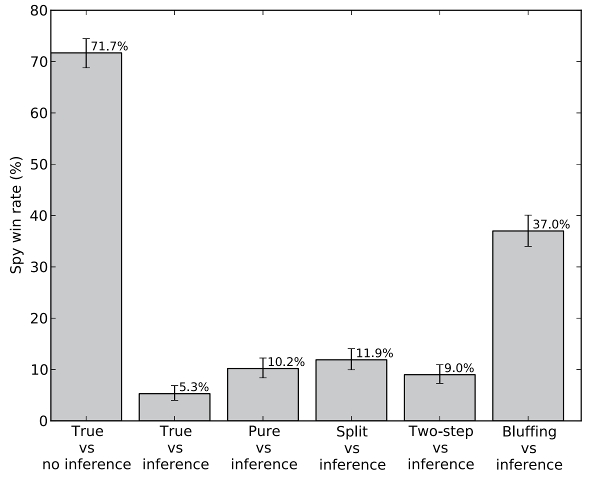

Fig. 1. Comparison of inference and self-determinization methods for The Resistance. Each bar shows the win rate for algorithm playing as the Spy team against non-spies using ISMCTS with or without particle filter inference.

Results shown in Figure 1 indicate that inference is extremely powerful against the TRUEONLY player, reducing the latter player's win rate by 66.4%. The PURE, SPLIT and TWOSTEP methods are around twice as strong as the non-self-determinizing player, with no significant difference between the three. The most effective counter to the inference player is the BLUFF player, with an improvement of 31.7% over the non-self-determinizing player. This shows that this bluffing method counteracts roughly half of the advantage gained by the non-spy players performing inference, and results in a more evenly matched game.

B. Balancing true and self-determinizations

In the previous section, the SPLIT, TWOSTEP and BLUFF players all used an equal division of their iteration budgets between self-determinizations and true determinizations. This section investigates the effect of this split on the player's performance. Once again the particle filter inference non-spy player using 20 000 iterations played against the SPLIT spy player using 40 000, but here the number of iterations T_1 used in the first (self-determinizing) search is varied between 0 and 40 000, with $T_2 = 40\,000 - T_1$ iterations for the second (true determinization) phase. Note that $T_1 = 40\,000$ is equivalent to pure self-determinization, and $T_1 = 0$ is equivalent to a player which uses only true determinizations.

Results are shown in Figure 2 and indicate an upward trend as the number of self-determinizing iterations increases, with performance reaching a plateau between 15 000 and 35 000 self-determinizing iterations and dropping off for 40 000 iterations. There is a trade-off to be made between searching the decision tree and searching other self-determinizations; these results suggest that devoting anywhere between $\frac{3}{8}$ and $\frac{7}{8}$ of the total search budget to self-determinizations yields reasonable performance. The experiment underlines that sensitivity of performance to this fraction is low in The Resistance, that 5000 true determinizaions is a sufficient amount and that any additional determinizations sampled should be self determinizations.

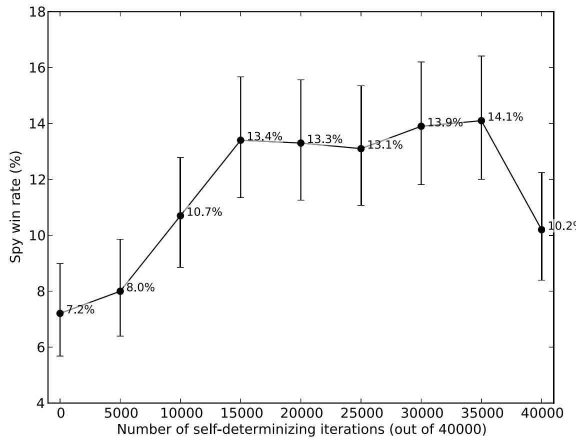

Fig. 2. Performance of the SPLIT spy player for The Resistance, devoting various numbers of iterations to self-determinizations.

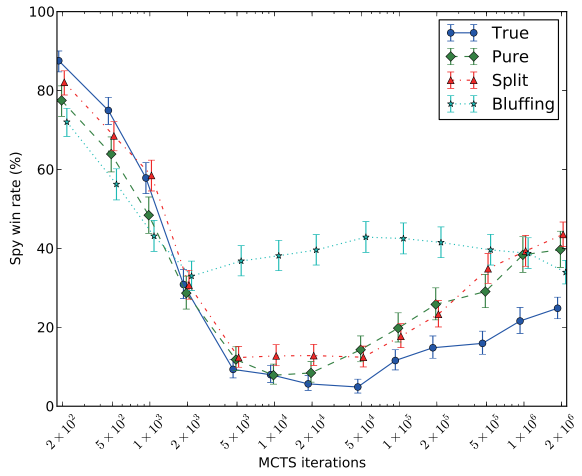

Fig. 3. Performance of ISMCTS variants for The Resistance for varying numbers of iterations. Non-spy players use ISMCTS with particle filter inference, whilst spies use ISMCTS with the specified self-determinization method (win rates are given for the Spy team). Both spies and non-spies use the specified number of iterations per decision.

C. Emergence of bluffing

Figure 3 shows the performance of the TRUEONLY, PURE, SPLIT and BLUFF spy players with varying numbers of ISMCTS iterations. In each case the non-spy players use inference but no self-determinization. All players (spies and non-spies) use the specified number of iterations, with the SPLIT and BLUFF spy players using half the iterations for self-determinizations and half for true determinizations. The four spy player types use the same number of iterations in total, unlike the experiments in Section VII-A where the self-determinizing players have twice the budget of the pure player. Note that the number of iterations varies for all players, so the spies' win rate can fluctuate up or down depending on which team is improving more quickly as the number of iterations increases.

For small numbers of iterations the performance of the different spy players is close, but the TRUEONLY and SPLIT players are stronger than the bluffing player by a statistically

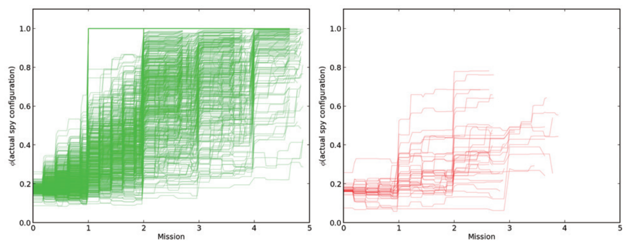

Fig. 4. Plots of the inferred probability for the correct spy configuration across 250 games of The Resistance. Non-spies use particle filter inference, whilst spies use no self-determinization (i.e. TRUE). Time moves left to right along the x-axis, from the beginning to the end of the game, scaled to align mission numbers. The left-hand plot shows games where the non-spy team won; the right-hand plot shows games won by the spies.

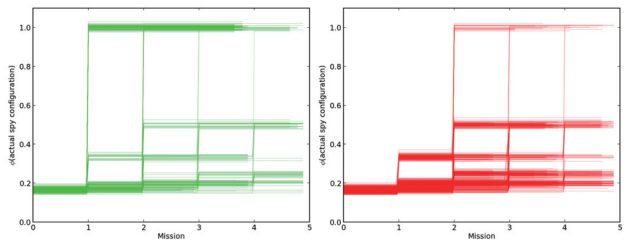

Fig. 5. As Figure 4, but without particle filter inference, i.e. only keeping track of hard constraints. To show the density of overlapping lines, a small random vertical offset has been added to each line.

significant margin (at 95% confidence). Between $5\,000$ and $200\,000$ iterations, the BLUFF player is much stronger than the other two. However for very large numbers of iterations SPLIT overtakes BLUFF whilst performance of TRUEONLY remains poor. This suggests that SPLIT with sufficiently many iterations can produce bluffing behaviour without an explicit bluffing mechanism. However it should be noted that the computational resources required for this player might be infeasible for a commercial application: a single-threaded C++ implementation of a player using $1\,000\,000$ iterations takes around 10 seconds per decision on a desktop PC with a 2.53GHz Intel Xeon processor and requires around 1GB of memory to store the MT-ISMCTS trees for a single decision.

An upward trend is visible in the win rate for the TRUEONLY player for more than $50\,000$ iterations, albeit a slower one than for the split player. One possible explanation for this is that the non-determinizing spy player may pessimistically assumes that the game is lost, as it is assuming that the non-spies know that it is a spy. MCTS tends to play randomly when its decisions do not affect the result of the game. However, playing randomly reduces the ability of the non-spies to perform inference, hence possibly leading to a situation where the spies can win. Similarly pessimistic behaviour can be seen when applying ISMCTS to a phantom connection game [1].

D. Effect of inference on the belief distribution

Figure 4 shows how a non-spy player's belief distribution evolves over time when using particle filter inference. The plots show the belief probability assigned by ϕ to the actual configuration of spies in the game, which starts as $\frac{1}{6} \approx 0.167$

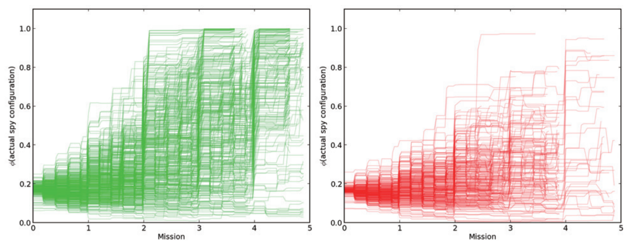

Fig. 6. As Figure 4, but with spy players using BLUFF.

and changes as the player performs inference. For comparison, Figure 5 shows how the belief distribution evolves if the player performs no inference and only keeps track of hard constraints, plotting the reciprocal of the number of spy configurations compatible with the members of failed mission teams. The inference player displays an overall upward trend much steeper than that for the player without inference and very rarely is the probability lower than $\frac{1}{6}$ (which would imply that an incorrect inference was made).

Figure 4 suggests some correlation between the success of inference and the outcome: in many of the games won by the spies, the non-spy players assign a relatively low probability to the actual spy configuration. In this experiment the spies are using the TRUE method, so are not trying to mislead the inference engine. Figure 6 shows the results when the spy players use BLUFF. Bluffing results in the probability of the actual spy configuration being typically lower and results in the spies winning more games. In particular, when the spies bluff there are no instances where the actual configuration had probability 1 after the first mission There are also many games won by the spies on the last mission whereas there are none when the spies do not bluff, suggesting that the bluffing players are more willing to sacrifice the first few rounds in order to win the game.

VIII. CONCLUSION

By including tree nodes for opponent decisions, tree search implicitly constructs an opponent model while it searches. We have shown that this opponent model can be re-used for inference. We have also shown that bluffing behaviours can be introduced simply by allowing the search to sample determinizations outside the current information set, but that certain modifications to ISMCTS can obtain this bluffing behaviour for less computational effort. The methods presented here were demonstrated to enable MCTS to perform bluffing and inference in The Resistance. Two games were played with a mix of 5 human players and 2 MCTS players, and both times there was an MCTS player on the winning side (non-spy in one game and spy in another), but the MCTS agents had mixed success at inferring the identity of the spies. More games would need to played (facilitated by development of a suitable interface) to draw any reasonable conclusions about the performance of MCTS compared with humans at The Resistance.

The new techniques presented do not use any game-specific heuristics, so it is natural to ask whether they can be applied to other games. However part of our success in The Resistance is due to the small number of states per information set, and attempting to scale the techniques to larger and more complex games does not yield the same level of performance. In order to tackle larger games within a reasonable computational budget, it would be necessary to exploit domain knowledge to reduce the number of states per information set (for the purposes of inference and bluffing). One way in which such knowledge could be injected is by performing inference not over the state space but over a much smaller feature space designed to capture the important pieces of hidden information whilst abstracting the rest away, in a manner similar to [16]. Domain knowledge could also be used improve the learning rate of MT-ISMCTS (using appropriate knowledge injection enhancements [13]), requiring fewer iterations for bluffing and inference to emerge from search. Combining both of these approaches is likely necessary for the methods developed in this work to scale to larger and more complex games.

From a game theoretic point of view, there is nothing special about inference and bluffing: they are merely labels for behaviours which must necessarily be included in a Nash equilibrium policy. As such, any AI technique proven to converge on a Nash equilibrium, such as counterfactual regret minimisation [25], is implicitly capable of inference and bluffing if required for optimal play. If application domain and computational resources permit, such techniques would normally be a more robust choice than MT-ISMCTS. However the strength of MCTS is that it can make good (but not necessarily Nash) decisions in an online fashion with a limited computational budget, and it scales well to combinatorially large game trees. This is in contrast to most techniques for approximating Nash equilibria, which require policies to be calculated offline (usually at great computational expense) and stored for later use. Some techniques are better suited to online settings, such as online MCCFR [26] and Smooth UCT [24]; comparing MT-ISMCTS to these is a subject for future work.

Acknowledgements

This work was funded by grant EP/H049061/1 of the UK Engineering and Physical Sciences Research Council (EPSRC).

References

[1] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information Set Monte Carlo Tree Search," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 2, pp. 120–143, 2012.

[2] D. Whitehouse, P. I. Cowling, E. J. Powley, and J. Rollason, "Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game," in *Proc. Artif. Intell. Interact. Digital Entert. Conf.*, Boston, Massachusetts, 2013, pp. 100–106.

[3] E. J. Powley, P. I. Cowling, and D. Whitehouse, "Information capture and reuse strategies in Monte Carlo Tree Search, with applications to games of hidden information," *Artif. Intell.*, vol. 217, pp. 92–116, 2014.

[4] N. Sephton, P. I. Cowling, E. J. Powley, and D. Whitehouse, "Parallelization of Information Set Monte Carlo tree search," in *IEEE Congress on Evolutionary Computation*, 2014.

[5] 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, Georgia, 2010, pp. 134–140.

[6] M. L. Ginsberg, "GIB: Imperfect Information in a Computationally Challenging Game," *J. Artif. Intell. Res.*, vol. 14, pp. 303–358, 2001.

[7] 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, Netherlands, 2007, pp. 57–64.

[8] J. Schäfer, "The UCT Algorithm Applied to Games with Imperfect Information," Diploma thesis, Otto-Von-Guericke Univ. Magdeburg, Germany, 2008.

[9] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower Bounding Klondike Solitaire with Monte-Carlo Planning," in *Proc. 19th Int. Conf. Automat. Plan. Sched.*, Thessaloniki, Greece, 2009, pp. 26–33.

[10] P. Ciancarini and G. P. Favini, "Monte Carlo tree search in Kriegspiel," *Artif. Intell.*, vol. 174, no. 11, pp. 670–684, 2010.

[11] 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, United Kingdom, 2011, pp. 17–24.

[12] 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.

[13] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012.

[14] D. Silver and J. Veness, "Monte-Carlo Planning in Large POMDPs," in *Proc. Neur. Inform. Process. Sys.*, Vancouver, Canada, 2010, pp. 1–9.

[15] 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. Decis. Theory Game Theory Workshop*, Atlanta, Georgia, 2010, pp. 37–42.

[16] 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, California, 2009, pp. 1407–1413.

[17] M. Richards and E. Amir, "Opponent Modeling in Scrabble," in *Proc. 20th Int. Joint Conf. Artif. Intell.*, Hyderabad, India, 2007, pp. 1482–1487.

[18] BoardGameGeek, "The Resistance," http://boardgamegeek.com/boardgame/41114/the-resistance.

[19] 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.

[20] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *Euro. Conf. Mach. Learn.*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Germany: Springer, 2006, pp. 282–293.

[21] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go," Inst. Nat. Rech. Inform. Auto. (INRIA), Paris, Tech. Rep., 2006.

[22] P. Perick, D. L. St-Pierre, F. Maes, and D. Ernst, "Comparison of Different Selection Strategies in Monte-Carlo Tree Search for the Game of Tron," in *Proc. IEEE Conf. Comput. Intell. Games*, Granada, Spain, 2012, pp. 242–249.

[23] J. A. M. Nijssen and M. H. M. Winands, "Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 4, pp. 282–294, 2012.

[24] J. Heinrich and D. Silver, "Self-Play Monte-Carlo Tree Search in Computer Poker," in *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, Québec, Canada, 2014, pp. 19–25.

[25] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret Minimization in Games with Incomplete Information," in *Proc. Adv. Neur. Inform. Process. Sys.*, Vancouver, Canada, 2008, pp. 1729–1736.

[26] M. Lanctot, V. Lisý, and M. Bowling, "Search in Imperfect Information Games using Online Monte Carlo Counterfactual Regret Minimization," in *Proceedings of the AAAI Workshop on Computer Poker and Imperfect Information*, 2014.