Bandits all the way down: UCB1 as a simulation policy in Monte Carlo Tree Search

Edward J. Powley, Daniel Whitehouse, and Peter I. Cowling
Department of Computer Science
York Centre for Complex Systems Analysis
University of York, UK
Email: edward.powley@york.ac.uk, dw830@york.ac.uk, peter.cowling@york.ac.uk

***Abstract*—Monte Carlo Tree Search (MCTS)** **is a family of asymmetric anytime aheuristic game tree search algorithms which have advanced the state-of-the-art in several challenging domains. MCTS learns a playout policy, iteratively building a partial tree to store and further refine the learned portion of the policy. When the playout leaves the existing tree, it falls back to a default *simulation policy*, which for many variants of MCTS chooses actions uniformly at random.**

This paper investigates how a simulation policy can be learned during the search, helping the playout policy remain plausible from root to terminal state without the injection of prior knowledge. Since the simulation policy visits states that are previously unseen, its decisions cannot be as context sensitive as those in the tree policy. We consider the well-known *Move-Average Sampling Technique (MAST)*, which learns a value for each move which is independent of context. We also introduce a generalisation of MAST, called *N-gram-Average-Sampling-Technique (NAST)*, which uses as context a fixed-length sequence (or N-tuple) of recent moves.

We compare several policies for selecting moves during simulation, including the UCB1 policy for multi-armed bandits (as used in the tree policy for the popular UCT variant of MCTS). In addition to the elegance of treating the entire playout as a series of multi-armed bandit problems, we find that UCB1 gives consistently strong performance. We present empirical results for three games of imperfect information, namely the card games Dou Di Zhu and Hearts and the board game Lord Of The Rings: The Confrontation, each of which has its own unique challenges for search-based AI.

I. Introduction

Monte Carlo Tree Search (MCTS) methods have attracted considerable interest in recent years [1], due to their success in challenging domains such as computer Go [2] and their applicability to a wide variety of sequential decision problems. MCTS is *asymmetric*: it concentrates its resources on the most promising regions of the search space. MCTS is *anytime*: it can make a decision within any time budget, however large or small. MCTS is *aheuristic*: in its basic form it requires no domain knowledge other than a forward simulation model.

MCTS uses reinforcement learning on policies. It performs repeated playouts of the game from current state to completion, using these trajectories to build a partial game tree and using their terminal rewards to reinforce promising regions of the tree. Each playout consists of a tree descent phase followed by a simulation phase. Tree descent typically uses a multi-armed bandit policy to balance exploitation of known good regions of the tree with exploration of unknown regions. Simulation refers to the portion of the playout for which the tree does not yet exist, and a typical simulation policy is to choose moves uniformly at random (this lends Monte Carlo Tree Search the first half of its name). This simulation policy is simple but often effective. Nevertheless, the performance of MCTS can often be improved if a more informed simulation policy is used. This can be achieved by injecting domain knowledge into the simulation policy, or by a wide variety of online learning methods [1].

This paper investigates two modifications to MCTS which learn a simulation policy in parallel with the learning of the tree policy. The *Move-Average Sampling Technique (MAST)* [3] works by learning a value estimate for every possible move in the game independent of the state from which it is played, the aim being to learn the "goodness" or "badness" of certain moves. *N-gram-Average Sampling Technique (NAST)* generalises this to sequences of N moves, learning the value of the Nth move in the context of the $N-1$ moves that preceded it. These ideas have been studied before by other authors [4], [5]; our contribution is to investigate the mechanism by which the value estimates are used to influence the simulation policy. We show that treating the simulation policy as a multi-armed bandit problem, and using UCB1 [6] as a simulation policy, yields consistently strong results. This is also rather elegant: both the tree and simulation phases of the playout use the same policy (UCB1), differing only in the contextual information they use. Another key contribution of this paper is to show that MAST and N-gram techniques, previously applied to games of perfect information, can also be effective for games of imperfect information in the setting of Information Set MCTS [7].

The structure of this paper is as follows. Section II reviews background material on MCTS. Section III defines the MAST and NAST enhancements and the different policy functions they can use. Section IV introduces the three games on which we test the enhancements. Section V compares the performance of UCB1 and other simulation policies for MAST, and also compares different N-gram lengths for NAST. Finally Section VI gives some concluding remarks.

II. Background

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a family of decision tree search algorithms which has been successfully applied to a wide variety of domains [1], mostly notably Go [2]. MCTS was discovered independently by several authors in 2006 [8], [9], [10] and has become notable for performing well in many domains without the need for extensive domain knowledge (leading to success in General Game Playing [11]) and domains where other search algorithms have failed to produce strong performance (for example Go, where today's most competitive computer players use MCTS [12], [13]).

MCTS is an anytime algorithm, which means it can be interrupted at any point and queried for a decision. In general allowing more time yields better decisions. The UCT algorithm [9] is the most common basis for MCTS implementations. The algorithm works by iteratively building a subtree of the decision tree where on each iteration four steps are performed:

- *Selection*: The algorithm descends through the tree built so far until a leaf node is reached. The *Tree Policy* is used to guide the descent through the search tree. In the case of the UCT algorithm, the UCB1 multi-armed bandit algorithm [6] is used to guide tree descent.
- *Expansion*: When a leaf node is reached, a new node is added to the tree (if possible) corresponding to an unvisited state.
- *Simulation*: After adding a new node to the tree, the game is played out to completion from the state corresponding to the new node according to the *Simulation Policy*. In the case of the UCT algorithm, the simulation policy selects an action uniformly at random from the set of all possible actions in each state. Improving the simulation policy is the focus of this work.
- *Backpropagation*: After the simulation policy is used to play a game out to completion, the outcome of this game is used to update statistics in each node visited during this iteration. In the case of the UCT algorithm, nodes record the number of times they are selected in the Selection phase, and the number of simulated games passing through the node which led to a win (or a total reward value if the game result is nonbinary).

Thus each MCTS iteration plays out a game to completion from the current state to a terminal state, with some decisions made by the Tree Policy and the rest made by the Simulation Policy. The entire sequence of actions from the current state to a terminal state is referred to as a *playout*.

B. Information Set MCTS

Much work on MCTS has focused on domains with *perfect information* [1]. *Information Set MCTS (ISMCTS)* [7], [14] is a family of MCTS variants designed to handle games with *imperfect information* (hidden information, simultaneous actions or chance events). ISMCTS makes use of *determinization*, i.e. random sampling of states consistent with the player's observations. ISMCTS has proven to be an effective approach for handling imperfect information, producing strong play and outperforming existing determinization approaches [7]. Additionally ISMCTS is designed to overcome the issue of *strategy fusion* [15] which affects other determinization based approaches. ISMCTS builds a tree of *information sets*, each information set being a collection of states which are indistinguishable from the point of view of a particular player. For example in a card game, a player's information set may contain all possible permutations of hidden cards held by opponents.

In addition to searching a tree of information sets, ISMCTS differs from MCTS by restricting each playout to a random *determinization* of the root information set. A determinization of an information set is one of the states contained within the information set and corresponds to randomly assigning (or guessing) hidden information. For example, a determinization in a card game may correspond to guessing the identity of all hidden cards. Sampling determinizations with the correct distribution automatically gives us the correct probability distributions over availability of opponent moves and outcomes of chance events.

Two of the games tested in this paper feature *partially observable moves*: a player can observe another player taking an action, but some aspect of that action is hidden. In LOTR:C (Section IV-C) players move pieces on a board, but the identity of the moving piece is hidden from the opponent. LOTR:C also features simultaneous actions, which we model as each player in turn playing a hidden action, with both actions revealed subsequently. In Hearts (Section IV-B) each round begins with the players passing cards to each other face-down and simultaneously, so the cards are hidden and subsequently revealed to the recipient only. In this paper we use the *Multi-Observer ISMCTS (MO-ISMCTS)* algorithm [7] to handle partially observable moves. MO-ISMCTS creates an information set tree from the point of view of each player, with the tree policy selecting each move using the tree from the point of view of the player making the move. The separate trees model the players' differing points of view more accurately than a single tree can. When simultaneous actions are modeled as partially observable moves as described above, the resulting MO-ISMCTS tree structure is algorithmically equivalent to the method of handling simultaneity in MCTS described by Shafiei et al [16] and Teytaud and Flory [17], where each player's move is selected by an independent UCB1 instance with a branch in the tree for each possible combination of moves.

III. MAST and NAST

This section describes the two simulation enhancements studied in this paper. In Section III-A we define Move-Average Sampling Technique (MAST) [18], which works by learning a value for each move independent of the context in which it is played. Additionally we define four different policies by which these learned evaluations can be used to sample actions during simulation. In Section III-B we introduce N-gram-Average Sampling Technique (NAST), which generalises MAST to learning an evaluation for fixed-length sequences of moves.

A. Move-Average Sampling Technique (MAST)

The *Move-Average Sampling Technique (MAST)* is a simulation enhancement for MCTS, first introduced by Finnsson

and Björnsson [3] in their CADIAPLAYER General Game Playing system [11]. MAST works by keeping track of an average reward for every possible action in the game, independent of the state from which the action was played. These statistics are updated whenever the action appears in a playout, and used to inform the simulation policy on future playouts.

More concretely, let A be the set of all actions that are legal in at least one game state. For each $a \in A$ we maintain a total reward $Q(a)$ and a trial count $n(a)$, both initialised to zero at the beginning of the search. Now consider a playout consisting of actions $\langle a_1, \ldots, a_m \rangle$ with reward vector $\boldsymbol{\mu}$. For each a_i $(i = 1, \ldots, m)$ we update

$$Q(a_i) \leftarrow Q(a_i) + \boldsymbol{\mu}_\rho \tag{1}$$
$$n(a_i) \leftarrow n(a_i) + 1\,, \tag{2}$$

where player ρ played action a_i and $\boldsymbol{\mu}_\rho$ is the component of $\boldsymbol{\mu}$ for player ρ. If the same action occurs more than once in the playout (i.e. if $a_i = a_j$ for $i \neq j$) then it is updated more than once.

For each state s with legal action set $A(s)$, the simulation policy gives a probability distribution π over $A(s)$. The simulation actions are selected according to these distributions. There are several ways in which the MAST statistics can be used to dictate the policy:

1) Gibbs distribution sampling: The original formulation of MAST [3] defines the simulation policy by means of the Gibbs distribution:

$$\pi(a) = \frac{\exp\left(\frac{Q(a)}{n(a)}/\tau\right)}{\sum_{b \in A(s)} \exp\left(\frac{Q(b)}{n(b)}/\tau\right)}\,. \tag{3}$$

Here $\tau > 0$ is a parameter used to "stretch" or "flatten" the distribution: as $\tau \to 0$ the policy greedily selects actions with maximal average reward $\frac{Q(a)}{n(a)}$; as $\tau \to \infty$ the policy selects actions uniformly at random. Intermediate values of τ can be used to tune between these two extremes.

2) ε-greedy: Tak et al [5] propose the use of ε-greedy selection as an alternative simulation policy for MAST. Let

$$A_{\max} = \arg\max_{a \in A(s)} \frac{Q(a)}{n(a)} \tag{4}$$

be the subset of $A(s)$ for which the average reward is maximal. Then the policy is defined by

$$\pi(a) = \begin{cases} \frac{1-\varepsilon}{|A_{\max}|} & \text{if } a \in A_{\max} \\ \frac{\varepsilon}{|A(s)|-|A_{\max}|} & \text{if } a \notin A_{\max}\,. \end{cases} \tag{5}$$

Here ε $(0 \leq \varepsilon \leq 1)$ is a tunable parameter. The policy greedily chooses an action with maximal reward with probability $1 - \varepsilon$, or any other action with probability ε. As with Gibbs distribution sampling, it is possible to tune ε-greedy between pure greed $(\varepsilon = 0)$ and uniform random $(\varepsilon = 1)$.

3) Roulette wheel: In roulette wheel sampling, we select moves with probability proportional to their average reward:

$$\pi(a) = \frac{\frac{Q(a)}{n(a)}}{\sum_{b \in A(s)} \frac{Q(b)}{n(b)}}\,. \tag{6}$$

Unlike the other simulation policies, there is no parameter to tune here.

4) UCB1: A good MCTS simulation policy must strike a balance between exploiting good moves (to ensure that the simulation reward accurately reflects the game theoretic value) and exploring other moves (to ensure sufficient diversity in the search's exploration of the space). The three policies described above achieve this, but we may expect a multi-armed bandit policy (specifically UCB1 [6]) to address this tradeoff more effectively since the balance of exploration and exploitation is critical to the design of bandit algorithms. Let

$$A_{\text{ucb}} = \arg\max_{a \in A(s)} \left(\frac{Q(a)}{n(a)} + c\sqrt{\frac{\log \sum_{b \in A(s)} n(b)}{n(a)}} \right)\,. \tag{7}$$

be the subset of $A(s)$ on which the UCB1 value is maximal. The exploration constant $c > 0$ is a tunable parameter, which may or may not have the same value as the exploration constant for the MCTS tree policy. The simulation policy is given by

$$\pi(a) = \begin{cases} \frac{1}{|A_{\text{ucb}}|} & \text{if } a \in A_{\text{ucb}} \\ 0 & \text{if } a \notin A_{\text{ucb}}\,, \end{cases} \tag{8}$$

i.e. it chooses uniformly amongst the actions with maximal UCB1 value.

Gibbs distribution and ε-greedy sampling policies achieve exploration by ensuring $\pi(a) > 0$ for all actions, as does the roulette wheel policy provided that $Q(a) \neq 0$. The UCB1 policy controls exploration more explicitly, ensuring that every action is selected at least logarithmically often [6] (i.e. the amount of time between successive selections of the same action increases as the logarithm of the number of trials).

B. N-gram-Average Sampling Technique (NAST)

In the context of games, an *N-gram* is a sequence of N consecutive actions. In the same way that MAST maintains average reward statistics for actions, we introduce *N-gram-Average Sampling Technique (NAST)* to maintain average reward statistics for N-grams. A 1-gram is simply an action, so MAST is a special case of NAST with $N = 1$. NAST can also be thought of as a generalisation of the last good reply principle [19]: the average reward for a 2-gram $\langle a_1, a_2 \rangle$ indicates whether action a_2 is a good reply to action a_1, and more generally the average reward for an N-gram $\langle a_1, \ldots, a_{N-1}, a_N \rangle$ indicates whether action a_N is a good reply to the sequence of consecutive actions $\langle a_1, \ldots, a_{N-1} \rangle$. This sequence can contain actions played by several players, including the one playing action a_N.

Stankiewicz et al [4] apply N-grams of length 2 and 3 with an ε-greedy simulation policy to the game of Havannah,

achieving a significant increase in playing strength. Tak et al [5] suggest an enhancement similar to NAST which uses a combination of 1-, 2- and 3-grams, and demonstrate its effectiveness in the domain of General Game Playing. There are two key differences between these approaches and NAST: first, NAST uses only a single N-gram length, allowing the effectiveness of different lengths to be measured in isolation; second, NAST allows for simulation policies other than ε-greedy to be used, including UCB1.

The operation of NAST is similar to that of MAST. Let A^N be the set of all N-grams. For each $\langle a_1, \ldots, a_N \rangle \in A^N$ we store a total reward $Q(a_1, \ldots, a_N)$ and a number of trials $n(a_1, \ldots, a_N)$. Now consider a playout $\langle a_1, \ldots, a_m \rangle$ with reward $\boldsymbol{\mu}$. For each N-gram occurring in the playout, $\langle a_i, \ldots, a_{i+N-1} \rangle$ for $i = 1, \ldots, m - N + 1$, we update

$$Q(a_i, \ldots, a_{i+N-1}) \leftarrow Q(a_i, \ldots, a_{i+N-1}) + \boldsymbol{\mu}_\rho \quad (9)$$

$$n(a_i, \ldots, a_{i+N-1}) \leftarrow n(a_i, \ldots, a_{i+N-1}) + 1\,, \quad (10)$$

where ρ is the player who played action a_{i+N-1} and $\boldsymbol{\mu}_\rho$ is the reward vector component for that player. If the same N-gram occurs more than once then it receives multiple updates.

Suppose that the simulation reaches state s, and the sequence of playout actions so far is $\langle a_1, \ldots, a_t \rangle$. If $t < N-1$ then the simulation policy is uniform random. Otherwise an action $a \in A(s)$ is selected according to the statistics gathered for the N-gram $\langle a_{t-N+2}, \ldots, a_t, a \rangle$. The MAST simulation policies defined in Section III-A are adapted for NAST by making the following substitutions:

$$Q(a) \rightarrow Q(a_{t-N+2}, \ldots, a_t, a) \quad (11)$$

$$n(a) \rightarrow n(a_{t-N+2}, \ldots, a_t, a) \quad (12)$$

$$A(s) \rightarrow \{\langle a_{t-N+2}, \ldots, a_t, a \rangle \ : \ a \in A(s)\}\,. \quad (13)$$

In other words, in place of statistics for an action a, we consider statistics for the N-gram consisting of the $N-1$ most recent actions in the playout followed by a. Instead of the set of legal actions $A(s)$, we use the set of all such N-grams for all $a \in A(s)$.

The idea of MAST is to measure the "goodness" of each available move independent of context. NAST instead measures the "goodness" of moves in the context of the $N-1$ moves that preceded them. We expect NAST to offer an improvement over MAST in domains where contextual information is important to success and useful statistics about chains of moves can be learnt. We also expect NAST to be a *robust* enhancement: in games when there is no useful information for NAST to learn, the N-gram values will be approximately equal and the simulation policy will resemble random play. NAST is unlikely to bias the playouts by reinforcing incorrect values.

IV. Experimental domains

A. Dou Di Zhu

Dou Di Zhu is a 3-player climbing card game which is hugely popular in China [20]. The game is played with a standard 52 card deck plus a red joker and a black joker. In each game two of the players form a coalition against the third player, who is designated as the *Landlord*, and has double the stake of the non-landlord players. The game consists of several rounds, where one player leads by playing a group of cards from one of several possible categories (singletons, pairs, straights and others). Then players take turns to play out a group of cards from the same category, but of higher rank. Players may also choose to pass, and must do so if they cannot beat the last group played. If two consecutive players pass, the third (who was last to play a group) then leads a new round with a group of cards from any category. The winning team is the first one to have a player discard all of their cards. A complete description of the rules is available at [20].

Dou Di Zhu poses several challenges for MCTS, in particular the high branching factor at nodes where players are making a leading play. Since there are several categories, many permutations of valid groups of cards within each category and a large number of possible cards the opponents may hold the branching factor is often in excess of 1000 [7]. Desipte this, our previous work on Dou Di Zhu [21], [14], [7] has demonstrated that without domain knowledge, ISMCTS plays at a level competitive with a commercially-developed heuristic AI.

B. Hearts

Hearts is a 4-player trick taking card game, widely known through a version distributed with Microsoft Windows. The game is played with a standard 52 card deck and the goal of the game is to minimise the number of points accumulated across several rounds. In our experiments, the game ends when one or more of the players finish a round with a total of 50 or more points. Each round begins with each player being dealt a hand of 13 cards and then players simultaneously pass three cards to an opponent (the direction cards are passed rotates after each round, and every fourth round no cards are passed). The game is then played like a regular trick taking game: each player plays one card at once, following suit if they are able, and the highest suited card takes the trick. At the end of the round each ♡ card a player has taken is worth 1 point and the $Q\spadesuit$ is worth 13 points. The only exception is if a player takes all 13 ♡s and $Q\spadesuit$: this is called *shooting the moon*, and gives that player no points and the other players 26 points each. A complete description of the rules is available at [22].

MCTS has previously been applied to Hearts [23] where the game was played with perfect information (all cards visible to all players). We retain the imperfect information and apply ISMCTS. Simultaneous card passing leads to partially observable moves in the game tree, for this reason we use the Multi-Observer ISMCTS algorithm (MO-ISMCTS). Since each round in Hearts begins with a new hand dealt to each player, there is little benefit to searching moves beyond the current hand. Instead we perform simulations of future rounds using a database of 10 000 simulated round scores generated offline by playing games of Hearts randomly. This fixes the horizon for MCTS at the end of the hand, significantly reducing the time taken to perform each MCTS iteration. We return a value of 1 for a win, 0 for a loss (fourth place) and 0.5 for placing second or third. This ensures that MCTS tries to avoid losing even in situations where winning is not possible.

C. Lord of the Rings: The Confrontation (LOTR:C)

Lord of the Rings: The Confrontation (LOTR:C) [24] is a 2-player board game by prolific game designer Reiner Knizia. It draws its theme from the novels of J. R. R. Tolkien. Each player has 9 character pieces each with different strengths and abilities (and there is a different set of characters for each player) which are arranged on a 4×4 grid rotated $45°$ such that each player sits at a corner of the board. Players know the location and identity of their own pieces but only know the location (and not identities) of the opponent pieces. When two characters from opposing sides occupy the same square they are revealed and fight each other. Combat is resolved through the use of combat cards (chosen simultaneously by each player) in addition to the strengths and abilities of the characters. The game is highly asymmetric since in addition to information asymmetry, different character sets and different combat cards, the two players have different win conditions. The *Light* player aims to move the character *Frodo* into the *Mordor* square whereas the *Dark* player aims to kill Frodo in combat.

LOTR:C has several sources of imperfect information (hidden information, partially observable actions and chance events) and we use the MO-ISMCTS algorithm to play this game. Although players alternate turns in LOTR:C, each turn contains many steps and choices for both players. We use a tree structure that creates nodes and branches for each choice within a turn. A turn involving combat can consist of five or more plies in the tree, including actions for both players as well as chance events. Our previous work has demonstrated ISMCTS to play LOTR:C at a level equal to experienced human players [7] although it is slightly erratic at some aspects of the game (such as simultaneous card choices). Additionally we ignore one aspect of the game: players arrange their characters on the board at the start of the game in a configuration (subject to some constraints) which is hidden to the opponent. We opted to use the same configuration for each game (designed by an experienced player) to reduce noise introduced by some board configurations being strategically weak. Note that this common starting configuration is not revealed to the AI players.

V. Experiments

A. Simulation policies for MAST

This experiment compares the four simulation policies for MAST described in Section III-A: Gibbs distribution, ε-greedy, roulette wheel and UCB1.

To tune the parameters for the policies, we played between 500 and 1000 games for each of our domains and each of several parameter settings. For Gibbs sampling we tested values of $\tau \in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 1, 1.5, 2, 4\}$, and found $\tau = 1$ to give the strongest play across all three games. For ε-greedy we tried $\varepsilon \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ and found $\varepsilon = 0.2$ to be best. For UCB1 we tested $c \in \{0.2, 0.5, 0.7, 1.0\}$ and found $c = 0.7$ to give the best performance overall. Roulette wheel sampling has no parameters to tune.

For a κ-player game, we play one instance of the algorithm in question against $\kappa - 1$ instances of the baseline player (unenhanced MO-ISMCTS). Each algorithm uses 5 000 iterations per decision. For Dou Di Zhu the MAST player plays as the Landlord against two non-Landlord baseline players; for Hearts the MAST player plays against three baseline players; for LOTR:C the MAST player plays as both Light and Dark (in separate experiments) against a baseline player. For each experiment we played between 500 and 2500 games across a cluster of seven PCs with Intel Xeon E31230 CPUs and 16GB RAM running Windows Server 2008 R2. The experiments presented in this paper represent approximately 8 single-core CPU-months of computation in total.

The results of this experiment are shown in Figure 1. The relative strengths of the policies varies between games, but we see that UCB1 and ε-greedy perform consistently well across all domains. UCB1 is significantly better than all other policies for Dou Di Zhu and is within experimental error of the best policies for the other games, while ε-greedy is amongst the best policies for all domains except Dou Di Zhu. Gibbs distribution sampling is significantly worse than the other policies for LOTR:C as Dark, and indeed shows no benefit over the player not using MAST in this domain. Gibbs distribution sampling does outperform the baseline in Dou Di Zhu, but does not reach the performance level of MAST with UCB1. Roulette wheel sampling performs well in LOTR:C, but fails to outperform the baseline for Dou Di Zhu and Hearts.

For Hearts, MAST has no effect on the simulation policy used for the root player's own moves: all of the cards in the player's hand must be played at some point, so every playout updates the same set of actions. Thus the average reward for an action is the average reward for all playouts, which is the same for all actions. This is not true for opponent decisions, as the opponent's cards vary between determinizations; here the average reward for an action is the average reward for all playouts on determinized hands that contain that card. This has the effect that if dealing a particular card to an opponent gives them an advantage, the opponent will be more likely to play that card earlier in subsequent playouts. The weakness of this strategy coupled with the lack of influence on the root player's strategy may explain why MAST, independent of simulation policy, is less beneficial for Hearts than for the other games. Indeed we see no statistically significant benefit to MAST for Hearts, although the fact that three of the four policies give a win rate higher than 25% suggests that significance is likely to be achieved with more trials. This argument does not apply to Dou Di Zhu and LOTR:C, where the set of all available moves is far larger than the set of moves in a particular playout.

Several authors (e.g. [25], [26], [27]) have observed that the simulation policy used by MCTS must preserve *diversity*: that is, it must strike a balance of playing plausible moves but not doing so too deterministically. A strong but deterministic simulation policy can often lead to weaker play than a less strong policy that incorporates randomness. Although it is not always phrased as such, this is an exploitation-exploration tradeoff. All of the policies considered here achieve this to some extent, but the best performing policies are those explicitly designed to mix determinism and randomness (ε-greedy) or to handle the exploitation-exploration tradeoff in multi-armed bandits (UCB1). Tak et al [5] have previously

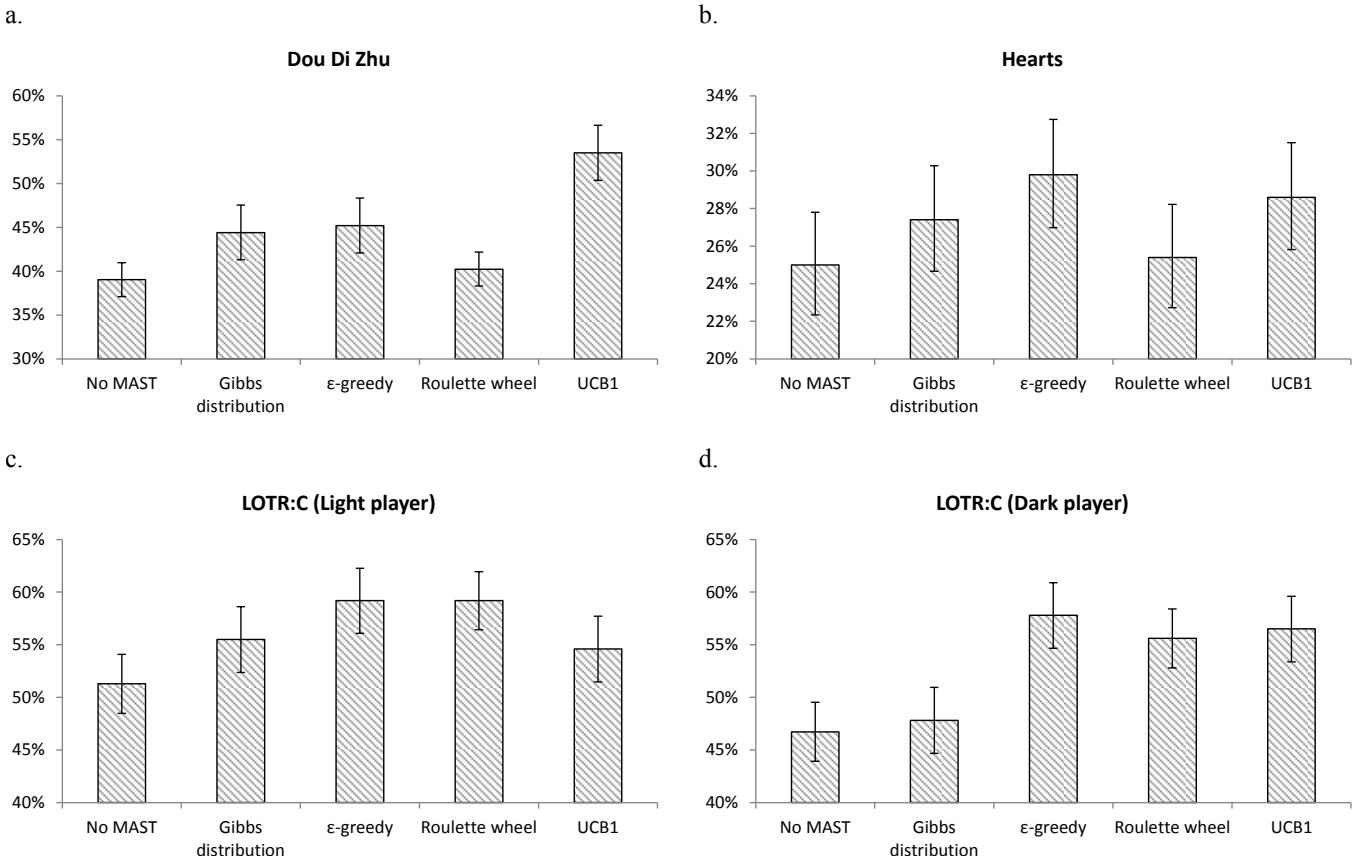

Fig. 1. Comparison of simulation policies used by the MAST enhancement, showing percentage of games won with 95% confidence intervals. The win rate for a player not using MAST is included for reference.

noted the strong performance of an ε-greedy policy for MAST; we have shown that UCB1 is not significantly worse than ε-greedy, and in Dou Di Zhu is significantly better.

B. N-gram lengths

This experiment tests the strength of the NAST enhancement (Section III-B) for different N-gram lengths. Having established UCB1 as a consistently strong simulation policy for MAST in the previous section, we use it exclusively in this section. We test the NAST player under the same conditions as Section V-A. The results are shown in Figure 2.

For LOTR:C as Dark, there is a sharp drop-off in performance between $N = 2$ and $N = 3$. For $N = 2$, NAST is maintaining average rewards for each pair consisting of an opponent move and a root player action in reply. This is a similar principle to the Last-Good-Reply Policy [19]. These results suggest that this principle is useful for LOTR:C as Dark, but longer N-grams dilute the statistics beyond the point of usefulness. This effect is not seen for LOTR:C as Light. This is in keeping with our observation in [7] that Dark must use a more reactive play style whilst Light must plan further ahead. For Hearts, $N = 2$ and $N = 3$ give significantly better performance than the baseline, whereas other values do not. The limited usefulness of $N = 1$ (MAST) for Hearts was discussed in Section V-A. These results suggest that there is value in applying a Last Good Reply like principle, or more generally in learning the values of moves in the context of the last move or pair of moves (which in many cases means the card or pair of cards most recently played in the current trick). For Dou Di Zhu and LOTR:C as Light the performance of NAST decreases as N increases, with $N = 1$ performing best or equal best.

The main disadvantage of longer N-grams is that more iterations are required to accumulate reliable average rewards. This is illustrated in Figure 3: the average number of visits to each N-gram decreases exponentially as N increases. Not only does this mean that the value estimates for N-grams are less accurate for larger N, it also indicates a lower chance of encountering an N-gram that has previously been seen during simulation, thus limiting the influence of NAST.

VI. Conclusion

In this paper we investigate the use of online learned simulation policies in MCTS. The resulting algorithm uses UCB1 (or other multi-armed bandit approaches) for the entire playout, with the only difference between tree and simulation policies being one of context. In the tree policy, the move statistics are specific to the current node (i.e. to a single game state or information set). In the simulation policy, the move statistics are universal to all states (for MAST) or specific only to the most recent $N - 1$ moves in the playout (for NAST). This allows a strong simulation policy to be learned long before the corresponding states are added to the tree, improving the quality of the playout policy overall.

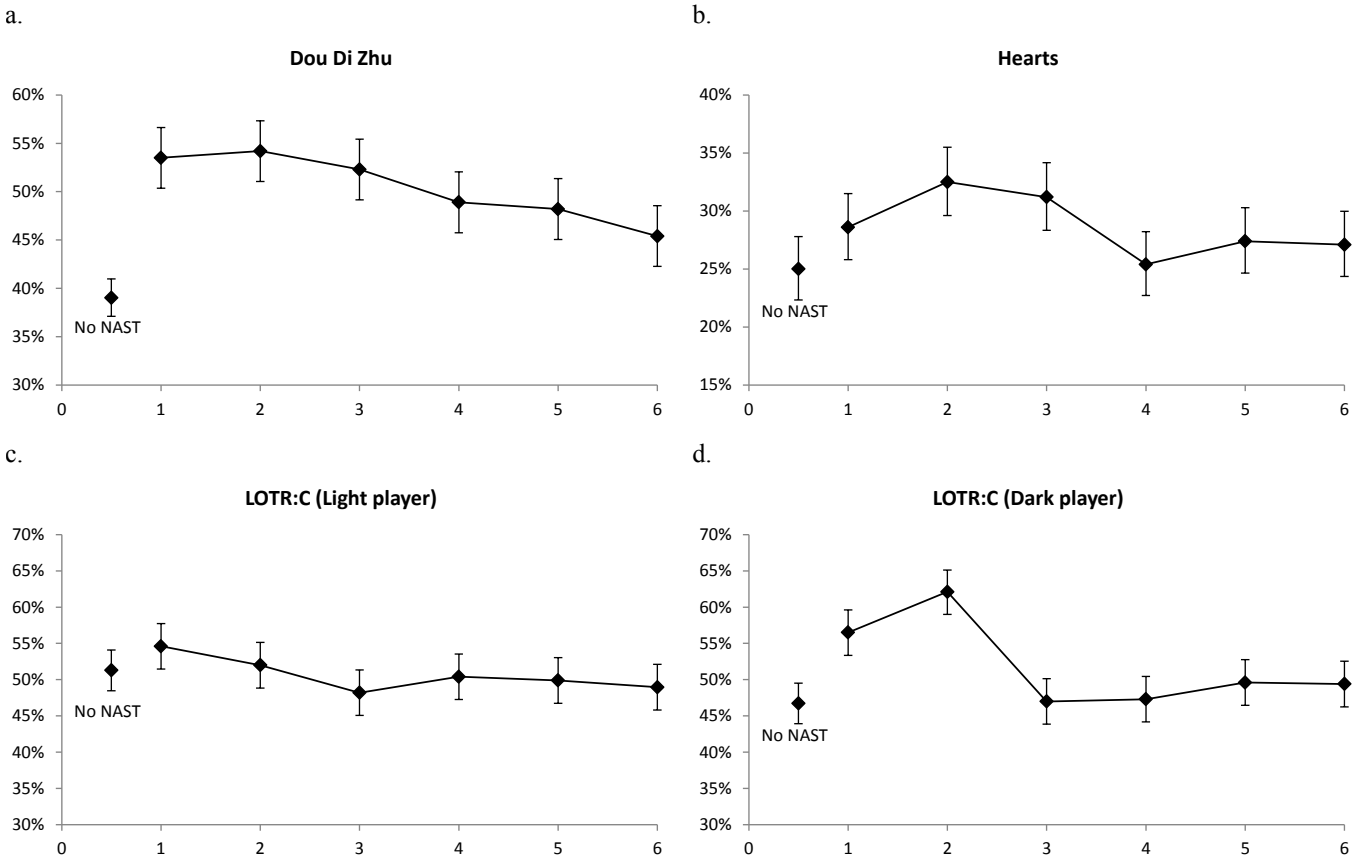

Fig. 2. Comparison of N-gram lengths for the NAST enhancement, showing percentage of games won with 95% confidence intervals. The win rate for a player not using NAST is included for reference.

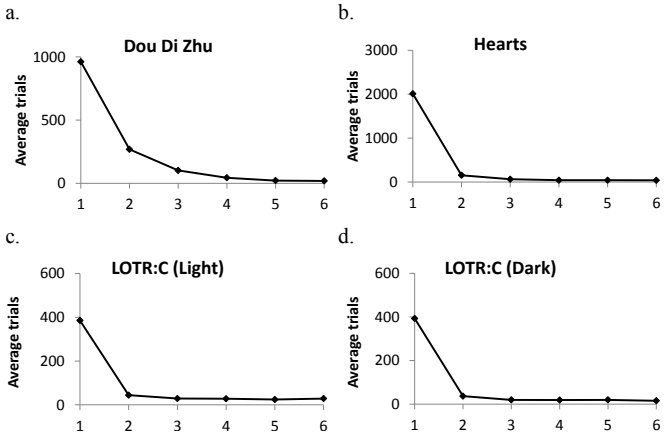

Fig. 3. For NAST with varying N-gram lengths, these graphs plot the average number of trials accumulated for each N-gram (including only those N-grams that receive at least one trial), averaged over all decisions by one player in 500 games.

UCB1 as simulation policy performs equal to or better than the weighted randomised sampling policies used by previous authors, and fits elegantly into the framework of MCTS.

We observed that using NAST with $N = 2$ offered an improvement over MAST for the Dark player in LOTR:C and that $N = 2, 3$ offered a small improvement in Hearts. In all other cases, longer N-gram lengths (where $N > 2$) failed to outperform $N = 1$ which is equivalent to MAST. However we have preliminary results to suggest that players using larger values of N show greater improvement for a given increase in computational budget. This suggests that longer N-grams are likely to be most beneficial when the computational budget is large, i.e. large enough to allow sufficient learning to take place. Longer N-grams increase the contextual information available to the simulation policy and thus potentially lead to stronger policies, but at the expense of slower learning (simply because the chance of a given N-gram appearing in a given playout is lower for larger N). The best solution may be to collect statistics for both short and long N-grams, and transition from the former to the latter as the search progresses (as the more context-sensitive statistics for the longer N-grams attain an appropriate level of statistical significance).

This work suggests a family of MCTS enhancements which use UCB1 as simulation policy, differing in the context used during simulation. One way of thinking about context is as a partitioning of the state space. For example MAST "partitions" the space into a single set, whereas NAST groups together all states for which the $N - 1$ preceding moves are equal. Finnsson and Björnsson [28] suggest a variant of MAST along these lines, called *Predicate-Average Sampling Technique (PAST)*, which defines a number of predicates on states (i.e. boolean properties that each state either does or does not have) and learns value estimates for (predicate,

action) pairs. One could imagine similarly taking features or predicates, either defined for a specific domain or learned in a game-independent way, and using them as contextual information for a UCB1-based simulation policy.

Once a state is added to the tree, information learned for the simulation policy no longer influences the playout policy through that state. In the long-term this is sensible as the information learned by the tree policy will be tailored to that specific state, but initially it will result in a degradation in quality of playouts through that state. A possible future enhancement is to allow the simulation policy information to continue influencing tree nodes until they achieve a certain number of visits. This is similar to the progressive history [29] and RAVE [30] enhancements, in which MAST-like values influence the tree policy with a weight which diminishes the more the node is visited.

The cutoff between tree policy and simulation policy is usually abrupt in MCTS. This work suggests variants of MCTS with a fuzzier tree boundary, gradually narrowing the context at a given step in the playout policy from an all-inclusive MAST-like context through NAST-like or feature-based contexts to the state-specific context of the tree. This progressive bootstrapping of policy learning extracts more value from each playout, thus potentially leading to much improved performance within a given computational budget.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful comments. This work is funded by grant EP/H049061/1 of the UK Engineering and Physical Sciences Research Council (EPSRC).

REFERENCES

[1] 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.

[2] C.-S. Lee, M. Müller, and O. Teytaud, "Guest Editorial: Special Issue on Monte Carlo Techniques and Computer Go," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 225–228, 2010.

[3] H. Finnsson and Y. Björnsson, "Simulation-Based Approach to General Game Playing," in *Proc. Assoc. Adv. Artif. Intell.*, Chicago, Illinois, 2008, pp. 259–264.

[4] J. A. Stankiewicz, M. H. M. Winands, and J. W. H. M. Uiterwijk, "Monte-Carlo Tree Search Enhancements for Havannah," in *Proc. 13th Int. Conf. Adv. Comput. Games, LNCS 7168*, Tilburg, The Netherlands, 2012, pp. 60–71.

[5] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "N-Grams and the Last-Good-Reply Policy Applied in General Game Playing," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 2, pp. 73–83, 2012.

[6] 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.

[7] 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.

[8] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *Proc. 5th Int. Conf. Comput. and Games, LNCS 4630*, Turin, Italy, 2007, pp. 72–83.

[9] 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.

[10] G. M. J.-B. Chaslot, J.-T. Saito, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik, "Monte-Carlo Strategies for Computer Go," in *Proc. BeNeLux Conf. Artif. Intell.*, Namur, Belgium, 2006, pp. 83–91.

[11] Y. Björnsson and H. Finnsson, "CadiaPlayer: A Simulation-Based General Game Player," *IEEE Trans. Comp. Intell. AI Games*, vol. 1, no. 1, pp. 4–15, 2009.

[12] C.-S. Lee, M.-H. Wang, G. M. J.-B. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, "The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments," *IEEE Trans. Comp. Intell. AI Games*, vol. 1, no. 1, pp. 73–89, 2009.

[13] 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, 2011.

[14] 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, South Korea, 2011, pp. 87–94.

[15] 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.

[16] 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, California, 2009.

[17] O. Teytaud and S. Flory, "Upper Confidence Trees with Short Term Partial Information," in *Proc. Applicat. Evol. Comput. 1, LNCS 6624*, 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.

[18] H. Finnsson and Y. Björnsson, "Simulation Control in General Game Playing Agents," in *Proc. Int. Joint Conf. Artif. Intell. Workshop Gen. Game Playing*, Pasadena, California, 2009, pp. 21–26.

[19] P. D. Drake, "The Last-Good-Reply Policy for Monte-Carlo Go," *ICGA Journal*, vol. 32, no. 4, pp. 221–227, 2009.

[20] Pagat, "Dou Dizhu," http://www.pagat.com/climbing/doudizhu.html, 2012.

[21] 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.

[22] Pagat, "Hearts," http://www.pagat.com/reverse/hearts.html, 2012.

[23] N. R. Sturtevant, "An Analysis of UCT in Multi-Player Games," in *Proc. Comput. and Games, LNCS 5131*, Beijing, China, 2008, pp. 37–49.

[24] BoardGameGeek, "Lord of the Rings: The Confrontation," http://boardgamegeek.com/boardgame/3201/lord-of-the-rings-the-confrontation, 2011.

[25] D. Silver and G. Tesauro, "Monte-Carlo Simulation Balancing," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, Montreal, Canada, 2009, pp. 945–952.

[26] G. M. J.-B. Chaslot, C. Fiter, J.-B. Hoock, A. Rimmel, and O. Teytaud, "Adding Expert Knowledge and Exploration in Monte-Carlo Tree Search," in *Proc. Adv. Comput. Games, LNCS 6048*, Pamplona, Spain, 2010, pp. 1–13.

[27] H. Baier and P. D. Drake, "The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 303–309, 2010.

[28] H. Finnsson and Y. Björnsson, "Learning Simulation Control in General Game-Playing Agents," in *Proc. 24th AAAI Conf. Artif. Intell.*, Atlanta, Georgia, 2010, pp. 954–959.

[29] J. A. M. Nijssen and M. H. M. Winands, "Enhancements for Multi-Player Monte-Carlo Tree Search," in *Proc. Comput. and Games, LNCS 6515*, Kanazawa, Japan, 2011, pp. 238–249.

[30] S. Gelly and D. Silver, "Combining Online and Offline Knowledge in UCT," in *Proc. 24th Annu. Int. Conf. Mach. Learn.* Corvalis, Oregon: ACM, 2007, pp. 273–280.