# Information capture and reuse strategies in Monte Carlo Tree Search, with applications to games of hidden information

Edward J. Powley, Peter I. Cowling, Daniel Whitehouse

*Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK*

**Abstract**

*Monte Carlo Tree Search (MCTS)* has produced many breakthroughs in search-based decision-making in games and other domains. There exist many general-purpose enhancements for MCTS, which improve its efficiency and effectiveness by learning information from one part of the search space and using it to guide the search in other parts. We introduce the *Information Capture And ReUse Strategy (ICARUS)* framework for describing and combining such enhancements. We demonstrate the ICARUS framework's usefulness as a frame of reference for understanding existing enhancements, combining them, and designing new ones.

We also use ICARUS to adapt some well-known MCTS enhancements (originally designed for games of perfect information) to handle information asymmetry between players and randomness, features which can make decision-making much more difficult. We also introduce a new enhancement designed within the ICARUS framework, *EPisodic Information Capture and reuse (EPIC)*, designed to exploit the episodic nature of many games. Empirically we demonstrate that EPIC is stronger and more robust than existing enhancements in a variety of game domains, thus validating ICARUS as a powerful tool for enhancement design within MCTS.

*Keywords:* Game tree search, hidden information, information reuse, machine learning, Monte Carlo Tree Search (MCTS), uncertainty

## 1. Introduction

*Monte Carlo Tree Search (MCTS)* is a decision tree search algorithm that has produced a huge leap in AI player strength for a range of two-player zero-sum games and proven effective in a wide range of games and decision problems [1]. In particular, MCTS is effective when it is difficult to evaluate non-terminal states

*Email addresses:* `edward.powley@york.ac.uk` (Edward J. Powley), `peter.cowling@york.ac.uk` (Peter I. Cowling), `dw830@york.ac.uk` (Daniel Whitehouse)

so that traditional depth-limited search methods perform poorly. For example, MCTS has advanced the state of the art in computer Go from the level of weak amateur to approach that of professional players in only a few years [2, 3]. MCTS has also produced state-of-the-art performance in many other domains, with over 250 papers published since the algorithm's invention in 2006 [1]. MCTS shows promise in real-time games, being the basis of winning competition entries for both Ms. Pac-Man [4] and the Physical Travelling Salesman Problem [5].

Generally speaking, MCTS algorithms heuristically build an asymmetric partial search tree by applying machine learning, using the weak reward signal given by randomly simulating a playout to the end of the game from nodes representing intermediate positions. The tree is descended by recursively applying a multi-armed bandit formula (such as UCB1 [6]) to each tree node's counts of simulation wins and visits.

While MCTS has provided effective and even state-of-the-art decision-making in its "vanilla" form (particularly UCT [7]), it is often enhanced [1]. Some of these enhancements incorporate external knowledge into the search, whereas others are *general purpose* enhancements which can be applied to any domain without specific knowledge. In some cases these enhancements are crucial aspects of successful MCTS programs, for example the RAVE enhancement [8] used in champion Go [9] and Hex [10] programs. In vanilla MCTS, the only information retained from a playout is the terminal reward, and the only use for that information is to update the nodes visited during the playout. Many enhancements aim to extract more data from each playout and spread the influence of that data across more of the search tree, thus increasing the value of each playout.

In this work we investigate the use of general purpose enhancements to improve the performance of MCTS. In some games[1] a move that is good in one state may be good in other similar states, and we argue that general purpose MCTS enhancements improve the performance of the algorithm by exploiting opportunities for learning in these situations. The enhancements in this paper bootstrap the learning of whether states and actions are good or bad by using analogy with similar states and actions elsewhere in the search tree. A substantial contribution of this work is to develop a framework which formalises the correlation between states and actions, and the effects that this has on the tree and default policies of MCTS. Further, we develop and empirically investigate combination operators for MCTS enhancements, and show how we can use our framework and operators to understand, categorise and invent new enhancements. Hence we can explain the effectiveness of MCTS enhancements by understanding how information is shared between states and actions and how this information is used to improve the MCTS selection and simulation policies. Additionally we show that enhancements developed for games of *perfect information* (where the state is fully observable to all players and state transitions

[1]The word *games* in this paper includes multiplayer games, single player puzzles and decision problems, although most work to date is on two-player noncooperative games.

are deterministic) can also be effective in games of *imperfect information* (where the state is partially observable with different observations for different players, and state transitions may be stochastic).

The framework in this paper aims to unify MCTS and its various enhancements, whereas other authors have sought to unify MCTS and related search techniques. Keller and Helmert [11] propose a framework for finite horizon Markov decision processes (i.e. single-player games). This framework can express UCT as well as other heuristic search and dynamic programming techniques. By interchanging the component parts of the methods within the framework, new methods are derived. Maes et al [12] define a grammar over Monte Carlo search algorithms for single-player games (including UCT and Nested Monte Carlo Search [13]), and use this to evolve new algorithms. Saffidine [14] presents a framework for "best first search" methods in two-player games, which encompasses methods such as MCTS-Solver [15] and Proof-Number Search [16] and guarantees that methods expressible in this framework must converge to the minimax solution of the game.

The idea of enhancing an algorithm to better capture and reuse information as it executes is used in a number of search and learning algorithms. The efficiency of the $\alpha$–$\beta$ pruning strategy in minimax search is largely dependent on the order in which actions are visited in the tree [17]. Enhancements such as the killer heuristic [18], history heuristic [19] and iterative deepening [20] use information gathered during the search to refine this ordering as the search progresses. Even $\alpha$–$\beta$ pruning itself can be seen as an information reuse enhancement, as it uses information gathered in one part of the tree to influence the search in other parts (specifically, to prune other parts entirely). Machine learning algorithms can also bootstrap learning through reuse. In *transfer learning* [21] or *lifelong learning* [22], the learner uses information learned from previous problems to bootstrap learning for the present problem. In *multitask learning* [23], the system learns to solve several problems in parallel. In both cases the system can be thought of as "learning to learn", thus these approaches are often termed *meta-learning* [24]. Typically meta-learning systems work by learning reusable features or representations, or by adjusting the parameters of an underlying learning algorithm. Although the actual methods used are different, the idea of a learning system acquiring knowledge over its lifetime as it is confronted by different problems is similar to the idea of a tree search algorithm transferring knowledge from one part of the game tree to another over the "lifetime" of a single search.

Most general purpose MCTS enhancements derive knowledge by comparing and combining simulations from different states. We show that these general purpose enhancements do not always work and are sometimes detrimental to the performance of MCTS, adding to existing observations that certain enhancements which are effective in some domains fail to provide any benefit in other domains (e.g. [25, 26]). The most effective enhancements correctly identify which states have correlated action values. This suggests that even if a general purpose enhancement is knowledge-free, there is implicit knowledge contained in the AI designer's decision of whether or not to use that enhancement.

As well as letting us choose between existing enhancements, consideration of correlated states allows us to design entirely new enhancements. In this paper we present a new enhancement, *EPisodic Information Capture and reuse (EPIC)*, that was designed by considering correlation between states in the card game Dou Di Zhu. Dou Di Zhu has an episodic structure, where a game consists of a sequence of somewhat independent rounds, and EPIC is designed to correlate states in analogous positions within different episodes. Many games have an episodic structure, and we demonstrate that EPIC is an effective general purpose enhancement for other games.

Capturing information in the correct way is important, but reusing it in the correct way is equally crucial. Our framework separates reuse from capture, enabling us to study the effectiveness of different information reuse techniques. In [27] we show that the precise information reuse method has an impact on the performance of an enhancement, and in particular we show that policies designed to balance exploitation and exploration, such as $\varepsilon$-greedy and UCB1 [6], produce strong simulation policies. In the case of UCB1, this leads to an elegant MCTS algorithm which uses a bandit algorithm to select all moves in the playout, where in the MCTS tree the action value estimates correspond to information about a single state and in simulations the action value estimates correspond to information reused between many states. Thus the only difference between the "in tree" (selection) and "out of tree" (simulation) modes of MCTS is whether the context in which the bandit algorithm executes is specific to a single state or general across a larger collection of states.

The structure of this paper is as follows. In Section 2 we give a brief overview of MCTS, and Section 3 introduces the definitions and notations we use throughout the paper. In Section 4 we define the ICARUS framework and show that many existing MCTS enhancements can be defined within this framework. We cast the best-known MCTS enhancements in this framework, adapting them to games of imperfect information in the process, and consider operators which allow us to combine information reuse enhancements. In Section 5 we define a new enhancement, *EPisodic Information Capture and reuse (EPIC)*, which captures information in such a way as to exploit the episodic nature of the search tree. In Section 6 we use the ICARUS framework to identify similarities and differences between enhancements. In particular we argue that MCTS enhancements differ in two ways: how information is captured, and how the captured information is reused within the MCTS algorithm. Section 7 introduces the three games we use in this paper as experimental domains, and Section 8 empirically compares the performance in these domains of EPIC and existing enhancements from the literature, studying a wide range of combinations using our ICARUS combination operators. Finally Section 9 gives some concluding remarks and directions for future work.

## 2. Monte Carlo Tree Search (MCTS)

### *2.1. The MCTS algorithm*

*Monte Carlo Tree Search (MCTS)* is a class of decision tree search algorithms discovered independently by several authors [28, 7, 29]. The most common MCTS implementations are based on the UCT algorithm [7], although there are many different versions of the algorithm [1]. MCTS builds a search tree iteratively where on each iteration the following four steps are performed:

1. *Selection*: The *tree policy* (often UCB1 [6]) is used to descend the existing search tree (the "tree search" in MCTS).
2. *Expansion*: A child of the final selected node is added, if possible.
3. *Simulation*: A simulation is run to estimate the outcome of the game. Typically this is done by playing random actions from the position reached during selection/expansion, until the end of the game (the "Monte Carlo" in MCTS).
4. *Backpropagation*: The result of the simulation is used to update all nodes visited during selection and expansion.

We refer collectively to the selection, expansion and simulation stages as the *playout*. The playout can be seen as a sequence of actions from the current (root) state to a terminal state.

### *2.2. Information Set MCTS*

*Information Set MCTS (ISMCTS)* is a variant of MCTS that handles imperfect information [30, 31]. This is achieved by building a tree of information sets (sets of states indistinguishable from one player's view point) rather than individual states, and dealing with the increased branching factor by restricting each MCTS iteration to a random *determinization* (a state sampled at random from the current information set). In this paper we use the MO-ISMCTS version of the algorithm, which deals with games that have partially observable moves by constructing a separate search tree (a "projection" of the underlying game tree) to reflect each player's observation of the game.

Each ISMCTS iteration uses a different determinization, and restricts selection and expansion to actions legal in that determinization. This leads to the *subset-armed bandit problem*: the set of children available for selection can differ between visits to the same node. To avoid over-exploration of "rare" children (corresponding to actions that are legal in relatively few determinizations), we use as the number of trials in the UCB1 formula the number of times the action was available for selection, rather than the number of times the parent node was visited [30].

ISMCTS is an effective algorithm for handling imperfect information, producing strong play and outperforming other determinization-based approaches [30].

In particular it overcomes the problem of *strategy fusion*, where a simpler approach incorrectly assumes it can tailor its future strategy to the opponent's private information [32, 33]. MCTS approaches combined with determinization have proven successful in games such as Klondike Solitaire [34], Skat [35] and Kriegspiel [36], as well as in General Game Playing for games of imperfect information [37].

## 3. Definitions and notation

For a set $X$, a sequence over $X$ is written as $\langle x_1, \dots, x_n \rangle$ for $x_i \in X$. The empty sequence is denoted $\langle\rangle$. The set of all sequences over $X$ is denoted $X^*$. The concatenation of two sequences $\boldsymbol{x} = \langle x_1, \dots, x_n \rangle$ and $\boldsymbol{y} = \langle y_1, \dots, y_n \rangle$ is $\boldsymbol{x} +\!\!+ \boldsymbol{y} = \langle x_1, \dots, x_n, y_1, \dots, y_n \rangle$. We also use the concatenation operator for prepending or appending single elements to a sequence, for example $\boldsymbol{x} +\!\!+ x_{n+1} = \langle x_1, \dots, x_n, x_{n+1} \rangle$ for $x_{n+1} \in X$.

Let $X$ be a set and let $\sim$ be an equivalence relation on $X$. Then $[x]^\sim$ is the $\sim$-class of $x \in X$, and $X/\!\sim$ is the set of all $\sim$-classes.

We now describe our terminology and notation for games. The notation is described in more detail in [30], and more detail on the concepts behind it can be found in [38] or other standard textbooks on game theory.

**Definition 1.** A *game* is defined by the following elements:

- $(S, \Lambda)$ is a finite nonempty directed graph, with $S$ the set of *states* and $\Lambda$ 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 *terminal states*
- $\rho : S \to \{0, 1, \dots, \kappa\}$ defines the *player about to act* in each state;
- $\pi_0 : \Lambda \to [0, 1]$, 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*;
- $\sim_i$, for each player $i = 0, 1, \dots, \kappa$, is an equivalence relation on $S$, whose classes are player $i$'s *information sets*;
- $\smile_i$, for each player $i = 0, 1, \dots, \kappa$, is an equivalence relation on $\Lambda$, whose classes are *moves* as observed by player $i$, such that for all $q, r, s \in S$, $(q, r) \smile_{\rho(q)} (q, s)$ implies $r = s$.

A game can be described as a sequential decision problem, where the players collectively choose a path through $(S, \Lambda)$ from $s_0$ to a terminal state. When the current state is $s_t$, player $\rho(s_t)$ chooses an edge $(s_t, s_{t+1})$ and the process continues from state $s_{t+1}$. If $\rho(s_t) = 0$ then the edge is instead selected according to the probability distribution induced by $\pi_0$; this models chance events such as dice rolls or card deals. In practice the players do not choose edges but *actions*: these are moves from the point of view of the moving player, and represent sets of analogous edges from different states (e.g. the set of edges that correspond to playing the card $Q\spadesuit$ from all states in which that is legal).

In a game of imperfect information, players do not observe the current state but observe the information set that contains it. Likewise they do not observe state transitions or actions but moves. (Note the distinction between state transitions, actions and moves in this paper: a player chooses an *action*, which induces a *state transition*, and the other players observe a *move*.) An information set consists of all states that are indistinguishable from the player's point of view; a move consists of all actions that are indistinguishable from the player's point of view. Thus a player's choices of action can depend only on the information sets and moves that he observes, not on the underlying states and actions.

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

$$M_i(s) = \{[(s,u)]^{\smile_i} \ : \ (s,u) \in \Lambda\} \ . \tag{1}$$

The set of *all moves from player $i$'s point of view* is the set of all moves legal in at least one state:

$$M_i = \Lambda/\smile_i = \bigcup_{s\in S} M_i(s) \,. \tag{2}$$

The set of *all moves* is the set of all moves from all players' points of view:

$$M = \bigcup_{i=1,\dots,\kappa} M_i \,. \tag{3}$$

The set of *legal actions* from $s$ is

$$A(s) = M_{\rho(s)}(s) \,, \tag{4}$$

i.e. the set of legal moves from the point of view of the player about to act. The set of *all actions* is the set of all actions legal in at least one state:

$$A = \bigcup_{s\in S} A(s) \,. \tag{5}$$

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 $\Gamma$ is the function $f : B \to S$ such that given $s \in S$, we have that $\forall a \in A(s)$, $(s,s') \in a \Rightarrow f(s,a) = s'$. In other words: $f(s,a)$ is the state reached by starting from $s$ and traversing the edge corresponding to $a$; $f(s,a)$ is the state resulting from performing action $a$ in state $s$.

**Definition 3.** An *action history*[2] from state $s$ is a sequence of actions $\langle a_1, \dots, a_n \rangle \in A^*$, such that

$$\begin{align} a_1 &\in A(s) \tag{6}\\ a_2 &\in A(f(s, a_1)) \tag{7}\\ a_3 &\in A(f(f(s, a_1), a_2)) \tag{8}\\ &\vdots \tag{9}\\ a_n &\in A(f(\dots(f(s, a_1), \dots), a_{n-1}))\,. \tag{10}\end{align}$$

Denote the set of all action histories from $s$ by $H(s)$. Extend the transition function $f$ to operate on action histories by defining

$$\begin{align} f(s, \langle\rangle) &= s \tag{11}\\ f(s, \langle a_1, \dots, a_n\rangle) &= f(f(s, \langle a_1, \dots, a_{n-1}\rangle), a_n)\,. \tag{12}\end{align}$$

An action history $\boldsymbol{h}$ is *terminal* if $f(s, \boldsymbol{h})$ is a terminal state. Denote the set of terminal action histories from $s$ by $H_T(s)$.

**Definition 4.** A *move history* for player $i$ from state $s$ is a sequence of moves from player $i$'s point of view, $\langle [a_1]^{\smile_i}, \dots, [a_n]^{\smile_i}\rangle \in M_i^*$, where $\langle a_1, \dots, a_n\rangle$ is an action history from $s$. Denote the set of all move histories for player $i$ from $s$ by $H_i^{\smile}(s)$, and the set of all move histories for all players by $H^{\smile}(s)$. If $\boldsymbol{h} = \langle a_1, \dots, a_n\rangle$ is an action history then the corresponding move history from player $i$'s point of view is denoted $[\boldsymbol{h}]^{\smile_i}$. Let $\rho = \rho(f(s, \langle a_1, \dots, a_{n-1}\rangle))$, so $\rho$ is the player who played the last action $a_n$ in the history. Then the move history from player $\rho$'s point of view is denoted by omission of the player number, i.e. $[\boldsymbol{h}]^{\smile}$.

Tree search algorithms operate on trees of histories. The history at a node is precisely the sequence of moves or actions that label the edges from the root to that node. Perfect information MCTS operates on trees of action histories, whereas MO-ISMCTS operates on trees of move histories.

## 4. Information Capture And ReUse Strategies (ICARUSes)

An *Information Capture And ReUse Strategy (ICARUS)* is an enhancement to MCTS that collects information from visits to one area of the game tree and uses that information to inform the future policy in other areas. The ICARUS framework introduced in this section allows us to define and analyse such enhancements and their combinations in an instructive, formal and consistent way. Furthermore, the framework is generic enough to be able to express any kind

[2]Note that a history from state $s$ begins, not ends, at state $s$. If we consider $s$ to be the current point in time, a "history" could more correctly be called a "future".

of information reuse enhancement (for example consulting an oracle of arbitrary complexity is permitted), but imposes a structure on how information is captured and used. This allows the structure of different enhancements to be easily compared, and provides useful pointers towards the design of future enhancements.

### *4.1. Defining ICARUSes*

The sharing of information between different parts of the tree is facilitated by *records*. These can be any objects. During the search, each record has a piece of *information* associated. The piece of information can also be any object; for example, it may be a tuple of numbers representing rewards and visit counts. The ICARUS defines three functions: the *policy function* specifying how the information is used during each MCTS playout, the *capture function* specifying which records are to be updated in response to the playout, and the *backpropagation function* specifying how each record's information is updated. This is similar to reinforcement learning, where the policy function is to be optimised, playouts provide a performance measure and the capture and backpropagation functions define a learning mechanism. Depending on the enhancement, records can be updated for different reasons: for example some records may be updated because they were selected, and others because they were available for selection but not actually selected. We use *capture contexts* to communicate this between the capture function and the backpropagation function.

**Definition 5.** Given a game as defined in Section 3, an *information capture and reuse strategy (ICARUS)* is a 7-tuple $(R, \Theta, \theta_{\text{initial}}, \alpha, \Psi, \xi, \omega)$ where

1. $R$ is a nonempty set of *records*. The elements of $R$ can be any objects.

2. $\Theta$ is a nonempty set, the *information domain*. The elements of $\Theta$ can be any objects.

3. $\theta_{\text{initial}} : R \to \Theta$ is the *initial information function*, which maps each record to a piece of information.

4. $\alpha : M^* \times (R \to \Theta) \times 2^A \to (A \to [0,1])$ is the *policy function*. This function takes three arguments (the current move history, the current mapping of records to information, and the legal action set for the current state) and returns a probability distribution over the action set. The same function $\alpha$ is used during selection and simulation phases of the playout.

5. $\Psi$ is a nonempty set of *capture contexts*. The elements of $\Psi$ can be any objects, and are used to communicate contextual information between $\xi$ and $\omega$ defined below.

6. $\xi : S \times M^* \to (R \times \Psi)^*$ is the *capture function*. This function takes two arguments (the root game state and the current move history) and maps them to a sequence of (record, capture context) pairs which are to be updated following a playout. The capture function returns a sequence

rather than a set to allow the same record to be updated more than once for the playout, and to specify the order of updates with different contexts in cases where this matters.

7. $\omega : \Theta \times \Psi \times \mathbb{R}^{\kappa} \to \Theta$ is the *backpropagation function.* This function takes three arguments (the current information for a record, the capture context specified by the capture function, and the reward vector from the simulation) and returns the new information for the record following a playout.

Algorithm 1 shows an MCTS algorithm using ICARUS to choose the best action from information set $I_{\text{root}}$. The algorithm begins by initialising the information associated with each record (lines 2–4); however, a practical implementation would initialise these values lazily as and when they are needed. Each iteration begins at the root node corresponding to the empty history (line 7), and samples a determinization (state) $s_{\text{root}}$ from the root information set (line 8) which becomes the current state $s$ for this iteration (line 9).

Each step of the playout uses the policy function $\alpha$ to choose an action $a$, depending on the current move history $[\boldsymbol{h}]^{\smile}{}_{\rho(s)}$ for the player about to act from state $s$, the current information mapping $\theta$, and the set of available actions $A(s)$ (line 11). The current history $\boldsymbol{h}$ is updated by appending $a$, and the current state $s$ is updated by applying $a$.

After the playout has reached a terminal state, the capture function is applied to the root determinization $s_{\text{root}}$ and the terminal history $\boldsymbol{h}$ to obtain the sequence of (record, context) pairs to be updated (line 16). For each of these pairs, the backpropagation function $\omega$ is used to update the information associated with the record (line 17).

The experimental domains in this paper are games of imperfect information, thus Algorithm 1 is designed to handle imperfect information using the approach of Information Set MCTS [30]. However it is equally applicable to games of perfect information. In this case the information set $I_{\text{root}}$ is a singleton $\{s_{\text{root}}\}$ and line 8 can be omitted.

## *4.2. Baseline ICARUS definition*

Specification 1 describes the baseline ICARUS definition used by an unenhanced search algorithm, defining the functions used in Algorithm 1. The resulting algorithm is equivalent to UCT [7] in the perfect information case and MO-ISMCTS with the UCB1 selection policy [30] in the imperfect information case. The algorithm uses reward vectors and assumes that each player tries to maximise his own reward in a max$^n$ fashion [39, 40], thus the algorithm can handle games with $\kappa > 2$ players as well as single-player and two-player games.

Each history has its own record (Base-1), and the information associated with a record is a total reward, a number of visits and an availability count (Base-2, Base-3). The policy is defined to use the subset-armed UCB1 algorithm (Base-4). During expansion all unexpanded actions have $n = 0$ and thus UCB1 value $\infty$, and so the policy chooses between them uniformly. Similarly

**Algorithm 1** The MCTS algorithm using ICARUS. The algorithm takes an information set $I_{\text{root}}$ as input and returns a legal action from that information set.

```
 1: function MCTS($I_{\text{root}} \in S/\sim_i$)
 2:     // Initialisation
 3:     for each record $r$ do
 4:         $\theta(r) = \theta_{\text{initial}}(r)$

 5:     for many iterations do
 6:         // Playout
 7:         $\boldsymbol{h} \leftarrow \langle\rangle$
 8:         choose $s_{\text{root}} \in I_{\text{root}}$ uniformly at random
 9:         $s \leftarrow s_{\text{root}}$
10:         repeat
11:             choose $a \in A(s)$ with probability $\alpha([\boldsymbol{h}]^{\smile_{\rho(s)}}, \theta, A(s))(a)$
12:             $\boldsymbol{h} \leftarrow \boldsymbol{h} +\!\!+ a$
13:             $s \leftarrow f(s, a)$
14:         until $s$ is terminal

15:         // Backpropagation
16:         for each $(r, \psi) \in \xi(s_{\text{root}}, \boldsymbol{h})$ do
17:             $\theta(r) \leftarrow \omega\left(\theta(r), \psi, \mu(s)\right)$

18:     return the $a \in A(I_{\text{root}})$ that was selected most often from the root
```

$$R^{\text{base}} = M^* \tag{Base-1}$$

$$\Theta^{\text{base}} = \mathbb{R}^\kappa \times \mathbb{N}_0 \times \mathbb{N}_0 \tag{Base-2}$$

$$\theta^{\text{base}}_{\text{initial}}(\boldsymbol{h}) = (\mathbf{0}, 0, 0) \tag{Base-3}$$

$$\alpha^{\text{base}}(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\operatorname*{arg\,max}_{a \in A_s} v(\theta([\boldsymbol{h} \mathbin{+\!\!\!+} a]^\smile))\right] \tag{Base-4}$$

$$\text{where } v((\boldsymbol{q}, n, m)) = \begin{cases} \dfrac{\boldsymbol{q}_\rho}{n} + c\sqrt{\dfrac{\log(m)}{n}} & \text{if } n > 0 \text{ and } m > 0 \\ +\infty & \text{if } n = 0 \text{ or } m = 0 \end{cases}$$

where $\boldsymbol{q}_\rho$ is the component of $\boldsymbol{q}$ corresponding to the player about to act at the end of $\boldsymbol{h}$

$$\Psi^{\text{base}} = \{\psi_{\text{avail}}, \psi_{\text{visit}}\} \tag{Base-5}$$

$$\begin{aligned} \xi^{\text{base}}(s, \langle a_1, \dots, a_t \rangle) = {} & \langle ([\langle a_1, \dots, a_i \rangle]^\smile, \psi_{\text{visit}}) \ : \ 0 \le i \le t_e \rangle \\ & \mathbin{+\!\!\!+} \langle ([\langle a_1, \dots, a_{i-1}, a \rangle]^\smile, \psi_{\text{avail}}) \ : \ 0 < i \le t_e, \\ & \qquad a \in A(f(s, \langle a_1, \dots, a_{i-1} \rangle)), a \ne a_i \rangle \end{aligned} \tag{Base-6}$$

where $t_e$ is minimal such that $\theta([\langle a_1, \dots, a_{t_e} \rangle]^\smile) = (\boldsymbol{q}, 0, m)$ for some $\boldsymbol{q}, m$, or $t_e = t$ if no such $t_e$ exists

$$\omega^{\text{base}}((\boldsymbol{q}, n, m), \psi, \boldsymbol{\mu}) = \begin{cases} (\boldsymbol{q} + \boldsymbol{\mu}, n+1, m+1) & \text{if } \psi = \psi_{\text{visit}} \\ (\boldsymbol{q}, n, m+1) & \text{if } \psi = \psi_{\text{avail}} \end{cases} \tag{Base-7}$$

where $\boldsymbol{q}$ denotes the total reward, $n$ denotes the number of visits and $m$ denotes the availability count.

**Specification 1:** The baseline ICARUS definition during simulation, all actions have UCB1 value $\infty$ and so the simulation policy is uniform random. The capture function specifies that the records to be updated during backpropagation are those that were selected, and those that were available to be selected due to being compatible with the current determinization; this is restricted to the portion of the playout corresponding to selection and expansion, i.e. the first $t_e$ actions (Base-6). These two collections of records are labelled with contexts $\psi_{\text{visit}}$ and $\psi_{\text{avail}}$ respectively (Base-5). Selected records have their rewards, visits and availabilities updated in the natural way: the simulation reward is added to the record's total reward, and the visit and availability counts are incremented by 1. Available records have their availability count incremented by 1, with reward and visit count remaining unchanged (Base-7).

Many ICARUSes apply different policies during selection, expansion and simulation. Let $\theta_n^{\text{base}}$ denote the visit count component of $\theta^{\text{base}}$, i.e. $\theta_n^{\text{base}}([\boldsymbol{h}]^{\smile})$ denotes the number of visits to history $[\boldsymbol{h}]^{\smile}$. A history $\boldsymbol{h}$ with available action set $A_s$ is said to be

- a *selection node* if $\theta_n^{\text{base}}([\boldsymbol{h}]^{\smile}) > 0$ and $\theta_n^{\text{base}}([\boldsymbol{h} + a]^{\smile}) > 0$ for all $a \in A_s$;
- an *expansion node* if $\theta_n^{\text{base}}([\boldsymbol{h}]^{\smile}) > 0$ but $\theta_n^{\text{base}}([\boldsymbol{h} + a]^{\smile}) = 0$ for at least one $a \in A_s$;
- a *simulation node* if $\theta_n^{\text{base}}([\boldsymbol{h}]^{\smile}) = 0$.

It is important to note that when this terminology is used in the definitions of ICARUSes, it always relates to the baseline statistics and not to the information maintained by the ICARUS itself.

### *4.3. Enhancements in the ICARUS framework*

This section casts some well-known information reuse enhancements from the literature into the ICARUS framework.

#### *4.3.1. All moves as first (AMAF)*

The *all moves as first (AMAF)* heuristic was introduced by Brügmann [41] in the context of Monte Carlo methods for Go, and was first combined with MCTS by Gelly and Silver [8] and independently by Drake and Uurtamo [42]. The underlying idea is that the value of an action is somewhat independent of the time at which it is played. This time independence is particularly true for games with pieces that rarely or never move once played, such as Go and Hex. AMAF and its variants have proven highly successful in these [3, 10] and other similar games. AMAF updates statistics for each action in the playout not just at the point when that action was played, but also at all earlier points when the action could legally have been played.

Specification 2 formulates AMAF in the ICARUS framework. Each history has its own record (AMAF-1), and the information associated with a record is a total reward and a number of visits (AMAF-2, AMAF-3). The policy uses a UCB1 formula based on the AMAF information (AMAF-4), here using as the number of trials the sum of visit counts for all currently available actions.

$$R = M^* \tag{AMAF-1}$$
$$\Theta = \mathbb{R}^\kappa \times \mathbb{N}_0 \tag{AMAF-2}$$
$$\theta_{\text{initial}}(\boldsymbol{h}) = (\boldsymbol{0}, 0) \tag{AMAF-3}$$
$$\alpha(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\arg\max_{a \in A_s} v(\theta([\boldsymbol{h} +\!\!\!+ a]^\smile))\right] \tag{AMAF-4}$$
$$\text{where } v((\boldsymbol{q}, n)) = \begin{cases} \dfrac{\boldsymbol{q}_\rho}{n} + c_{\text{AMAF}}\sqrt{\dfrac{\log\left(\sum_{b \in A_s} \theta_2([\boldsymbol{h} +\!\!\!+ b]^\smile)\right)}{n}} & \text{if } n > 0 \\ +\infty & \text{if } n = 0 \end{cases}$$
where $\theta_2$ denotes the component of $\theta$ in $\mathbb{N}_0$
$$\Psi = \{\psi\} \tag{AMAF-5}$$
$$\begin{aligned}\xi(s, \langle a_1, \ldots, a_t\rangle) = \langle ((\langle a_1, \ldots, a_{i-1}, a_j\rangle, \psi) \ :\ & 0 \le i < j \le t, \\ & a_j \in A(f(s, \langle a_1, \ldots, a_{i-1}\rangle)) \\ & \text{and } \langle a_1, \ldots, a_i\rangle \text{ is a selection node}\rangle\end{aligned} \tag{AMAF-6}$$
$$\omega((\boldsymbol{q}, n), \psi, \boldsymbol{\mu}) = (\boldsymbol{q} + \boldsymbol{\mu}, n + 1) \tag{AMAF-7}$$

**Specification 2:** All moves as first (AMAF)

The capture function specifies that the nodes to be updated are those siblings of nodes visited during tree descent that correspond to actions played later in the playout (AMAF-6). This is the key property of the AMAF algorithm. Backpropagation updates the rewards and visits in the natural way (AMAF-7), and does not require any contextual information (AMAF-5).

One well-known variant of AMAF is *rapid action value estimation (RAVE)* [8, 3], in which the influence of the AMAF value decays the more a node is visited. In Section 4.4 we define composition operators on ICARUSes, and express RAVE as a composition of baseline and AMAF ICARUSes.

### *4.3.2. Move-average sampling technique (MAST)*

*Move-average sampling technique (MAST)*[3] was introduced by Finnsson and Björnsson [43] and used in their CadiaPlayer general game player [44]. The idea is to maintain average reward statistics for each action independently of where it occurs in the game tree, and use these statistics to bias the simulation policy.

MAST is defined in Specification 3. There is a record for each combination of an action and a player who plays that action (MAST-1). The information associated with a record is a total (scalar) reward and a visit count (MAST-2, MAST-3). The policy selects actions according to a Gibbs distribution, using

[3]What Finnsson and Björnsson [43] call "moves" are "actions" in our terminology.

$$R = A \times \{1, \dots, \kappa\} \tag{MAST-1}$$

$$\Theta = \mathbb{R} \times \mathbb{N}_0 \tag{MAST-2}$$

$$\theta_{\text{initial}}(a, i) = (0, 0) \tag{MAST-3}$$

$$\alpha(\langle a_1, \dots, a_t \rangle, \theta, A_s)(a) = \frac{e^{v(a)/\tau}}{\sum_{b \in A_s} e^{v(b)/\tau}} \tag{MAST-4}$$

$$\text{where } v(a) = \begin{cases} \frac{q}{n} & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases} \text{ for } \theta(a, \rho_n) = (q, n)$$

$$\text{where } \rho_i = \rho(f(s, \langle a_1, \dots, a_i \rangle))$$

$$\Psi = \{1, \dots, \kappa\} \tag{MAST-5}$$

$$\xi(s, \langle a_1, \dots, a_t \rangle) = \langle (a_i, \rho_i) \ : \ i = 1, \dots, t \rangle \tag{MAST-6}$$

$$\omega((q, n), \rho, (\mu_1, \dots, \mu_\kappa)) = (q + \mu_\rho, n + 1) \tag{MAST-7}$$

**Specification 3:** Move-average sampling technique (MAST)

the average reward calculated from the total reward and visit count (MAST-4). Backpropagation updates the records associated with the actions played during the playout (MAST-6), with the player who played each action as contextual information (MAST-5). The total reward and number of visits are updated in the natural way (MAST-7). If the same (action, player) pair appears more than once in the playout, it is updated more than once during backpropagation.

This formulation of MAST applies the same policy throughout the playout, whereas [43] applies the Gibbs policy during expansion and simulation only. This behaviour can be implemented within the ICARUS framework by use of composition operators (Section 4.4).

In its original formulation, MAST uses a policy based on a Gibbs distribution. Tak et al [45] propose instead using an $\varepsilon$-greedy policy, i.e. replacing the policy function in Specification 3 with

$$\alpha(\langle a_1, \dots, a_t \rangle, \theta, A_s)(a) = \varepsilon \mathbb{U}[A_s] + (1 - \varepsilon) \mathbb{U}\left[\operatorname*{arg\,max}_{b \in A_s} v(b)\right] \tag{MAST-$\varepsilon$-greedy-4}$$

for a constant $\varepsilon$. With probability $\varepsilon$ this policy chooses uniformly over all available actions; with probability $1 - \varepsilon$ it chooses uniformly over the actions whose average value is maximal.

Another possibility is to use a roulette wheel policy, in which the probability for each move is proportional to its average reward:

$$\alpha(\langle a_1, \dots, a_t \rangle, \theta, A_s)(a) = \frac{v(a)}{\sum_{b \in A_s} v(b)} \tag{MAST-Roulette-4}$$

Yet another possibility is to use a bandit policy such as UCB1, requiring us to keep track of availability counts for each action and update these during backpropagation.

### *4.3.3. Variants of MAST*

Finnsson and Björnsson [46] describe a variant of MAST called *tree-only MAST (TO-MAST)*, in which only statistics for the actions played during selection and expansion (i.e. not during simulation) are updated. This can be defined by modifying the capture function of Specification 3:

$$\begin{aligned}\xi(s, \langle a_1, \dots, a_t \rangle) = \langle (a_i, \rho_i) \ : \ & i = 1, \dots, t \\ & \text{and } \langle a_1, \dots, a_i \rangle \text{ is a selection or expansion node} \rangle \end{aligned} \tag{TO-MAST-6}$$

Finnsson and Björnsson [46] describe two refinements of MAST to enable embedding of domain specific knowledge. In *predicate-average sampling technique (PAST)*, states are labelled using a list of predicates; instead of maintaining average rewards for actions, rewards are maintained for (predicate, action) pairs consisting of a predicate that holds in a state and the action played from that state. PAST can be represented in the ICARUS framework by modifying Specification 3, including the predicate as an element of the record tuple and modifying the policy and capture functions to take predicates into account.

The second refinement is *features-to-action sampling technique (FAST)*. This uses the $TD(\lambda)$ temporal difference learning algorithm to learn a value function for actions, both offline before the search begins and online based on the MCTS playouts. In the ICARUS framework, the values learned offline can be encoded in the initial information function $\theta_{\text{initial}}$, and the online learning by embedding $TD(\lambda)$ in the backpropagation function $\omega$.

### *4.3.4. Last good reply (LGR)*

*Last good reply (LGR)* is a simulation policy introduced by Drake [47]. When playing a game, each action can be thought of as a reply to the opponent's previous move. If the replying player goes on to win the game, this gives us some evidence that the reply was good. LGR records good replies from MCTS playouts; during simulation, if a good reply is recorded for the previous move then it is played deterministically. LGR has been shown to improve the performance of MCTS for Go [47, 48], Havannah [49] and General Game Playing [45].

Specification 4 gives LGR as an ICARUS. Each record specifies a move to be replied to, and the player making the reply (LGR-1). The information associated with a record is the last good action played in reply to that move by that player, or $\perp \notin A$ if no reply has yet been recorded (LGR-2, LGR-3). The policy examines the most recent move $[a_t]^{\smile \rho_t}$ from the point of view of the player about to act $\rho_t$. If a reply has been recorded, and that reply is compatible with the current determinization, then it is played. Otherwise, a legal action is chosen uniformly at random (LGR-4). During backpropagation, the records updated are those corresponding to the actions in the playout, each

$$R = M \times \{1, \dots, \kappa\} \tag{LGR-1}$$

$$\Theta = A \cup \{\perp\} \tag{LGR-2}$$

$$\theta_{\text{initial}}(m, i) = \perp \tag{LGR-3}$$

$$\alpha(\langle a_1, \dots, a_t \rangle, \theta, A_s)(a) = \begin{cases} \mathbb{U}[A_s] & \text{if } \theta\left([a_t]^{\smile \rho_t}, \rho_t\right) = \perp \\ & \text{or } \theta\left([a_t]^{\smile \rho_t}, \rho_t\right) \notin A_s \\ 1 & \text{if } \theta\left([a_t]^{\smile \rho_t}, \rho_t\right) = a \\ 0 & \text{otherwise} \end{cases} \tag{LGR-4}$$

$$\text{where } \rho_i = \rho(f(s, \langle a_1, \dots, a_i \rangle))$$

$$\Psi = A \times \{1, \dots, \kappa\} \tag{LGR-5}$$

$$\xi(s, \langle a_1, \dots, a_t \rangle) = \langle (([a_i]^{\smile \rho_i}, \rho_i), (a_{i+1}, \rho_i)) \ : \ i = 1, \dots, t-1 \rangle \tag{LGR-6}$$

$$\omega(a_{\text{old}}, (a_{\text{new}}, \rho), \boldsymbol{\mu}) = \begin{cases} a_{\text{new}} & \text{if } \boldsymbol{\mu}_\rho > 0 \\ a_{\text{old}} & \text{if } \boldsymbol{\mu}_\rho \leq 0 \end{cases} \tag{LGR-7}$$

**Specification 4:** Last good reply (LGR)

action observed from the point of view of the player immediately following it (LGR-6). The context specifies the action with which that player replied, as well as the identity of the player (LGR-5). If the player won the simulated game (i.e. achieved a reward greater than zero), the action is recorded as the last good reply; if not, the existing information is retained (LGR-7).

In [47], the reply information is used only during simulation, whereas Specification 4 has it used for the entire playout. This is likely to be very weak. However we define it in this way so that the stage at which the reply information is used can be specified naturally by composition operators (Section 4.4) rather than as a part of the ICARUS itself.

Baier and Drake [48] describe a variant of LGR called *last good reply with forgetting (LGRF)*, in which replies that led to a loss are deleted from the reply table. Specification 4 can be modified to describe LGRF simply by modifying the backpropagation function:

$$\omega(a_{\text{old}}, (a_{\text{new}}, \rho), \boldsymbol{\mu}) = \begin{cases} a_{\text{new}} & \text{if } \boldsymbol{\mu}_\rho > 0 \\ \perp & \text{if } a_{\text{old}} = a_{\text{new}} \text{ and } \boldsymbol{\mu}_\rho \leq 0 \\ a_{\text{old}} & \text{otherwise} \end{cases} \tag{LGRF-7}$$

### *4.3.5. n-gram average sampling technique (NAST)*

*n-gram average sampling technique (NAST)* was introduced by Powley et al [27], based on previous work by Stankiewicz et al [49] and Tak et al [45]. NAST generalises the notion of MAST: instead of learning values for single moves, NAST learns values for sequences of consecutive moves (indeed, MAST can be thought of as the $N = 1$ case for NAST).

$$R = M^n \times \{1, \ldots, \kappa\} \tag{NAST-1}$$

$$\Theta = \mathbb{R} \times \mathbb{N}_0 \tag{NAST-2}$$

$$\theta_{\text{initial}}(m_1, \ldots, m_n, i) = (0, 0) \tag{NAST-3}$$

$$\alpha(\langle a_1, \ldots, a_t \rangle, \theta, A_s)(a) = \mathbb{U}\left[\operatorname*{arg\,max}_{a \in A_s} v(\theta(\langle a_{t-n+2}, \ldots, a_t, a\rangle^{\smile}, \rho_t))\right] \tag{NAST-4}$$

$$\text{where } \rho_i = \rho(f(s, \langle a_1, \ldots, a_i \rangle))$$

$$\text{and } v((q, n)) = \begin{cases} \dfrac{q}{n} + c_{\text{NAST}}\sqrt{\dfrac{\log \Sigma}{n}} & \text{if } n > 0 \\ +\infty & \text{if } n = 0 \end{cases}$$

$$\text{where } \Sigma = \sum_{b \in A_s} \theta_2(\langle a_{t-n+2}, \ldots, a_t, b\rangle^{\smile})$$

where $\theta_2$ denotes the component of $\theta$ in $\mathbb{N}_0$

$$\Psi = \{1, \ldots, \kappa\} \tag{NAST-5}$$

$$\xi(s, \langle a_1, \ldots, a_t \rangle) = \langle (\langle a_i, \ldots, a_{i+n-1} \rangle, \rho_{i+n-1}) \ : \ i = 1, \ldots, t-n+1 \rangle \tag{NAST-6}$$

$$\omega((q, n), \rho, \boldsymbol{\mu}) = (q + \boldsymbol{\mu}_\rho, n+1) \tag{NAST-7}$$

**Specification 5:** *n*-gram average sampling technique (NAST)

NAST is defined in Specification 5. Each record is an $n$-gram, i.e. a sequence of $n$ moves (NAST-1). Note that $n$ is a parameter here; Specification 5 defines a family of enhancements for $n = 1, 2, 3, \ldots$. The information associated with a record is the total reward and number of visits (NAST-2, NAST-3). The policy uses these to select actions according to UCB1 (NAST-4). Backpropagation updates the records associated with each sequence of $n$ moves in the playout (NAST-6), with the player who played the last move in the sequence as contextual information (NAST-5). The total reward and number of visits are updated in the natural way (NAST-7). Note that NAST with $n = 1$ is equivalent to MAST (Section 4.3.2) with the UCB1 policy.

Stankiewick et al [49] demonstrate the effectiveness of $n$-gram techniques in MCTS for the game Havannah, and Tak et al [45] show that similar techniques work in a General Game Playing setting. In [50] we show that NAST works for the three imperfect information games studied in the present paper (Section 7), with $n = 2$ typically giving the strongest performance.

### *4.3.6. Other examples*

The literature contains many other examples of MCTS enhancements that involve either using information from external sources or capturing and reusing information within the search. All such approaches designed to date can be represented in the ICARUS framework. We chose AMAF, MAST, LGR and NAST as examples because they capture and reuse information in significantly different ways, whereas many enhancements are modifications of existing ones (for example the different AMAF variants described in [51]). Furthermore, these four enhancements have led to significant increases in the power of the MCTS algorithm for diverse application domains. This section briefly describes how some other enhancements from the literature can be defined within the ICARUS framework.

Chaslot et al [52] introduce *progressive bias* and *progressive unpruning*, which use a heuristic value function to bias selection and restrict expansion respectively. In the ICARUS framework this can be achieved by encoding the heuristic in the initial information function $\theta_{\text{initial}}$ and modifying the policy function $\alpha$ appropriately.

Nijssen and Winands [53] propose a modification of progressive bias called *progressive history*, which replaces the heuristic function with values extracted from simulations. Within the ICARUS framework this is similar to progressive bias, except that the information is updated by the backpropagation function $\omega$ instead of being initialised heuristically.

Rimmel and Teytaud [54] introduce *contextual MCTS*, which works by mapping each terminal history to several "tiles", where a tile corresponds to a pair of (not necessarily consecutive) actions played by the same player. During backpropagation the average values of tiles are updated, and these values are used to bias simulations. When contextual MCTS is encoded as an ICARUS, the tiles become records and the policy and backpropagation functions are defined in the natural way.

$$R = R_1 \sqcup R_2 \tag{COMB-1}$$

$$\Theta = \Theta_1 \sqcup \Theta_2 \tag{COMB-2}$$

$$\theta_{\text{initial}}(r) = \begin{cases} \theta^1_{\text{initial}}(r) & \text{if } r \in R_1 \\ \theta^2_{\text{initial}}(r) & \text{if } r \in R_2 \end{cases} \tag{COMB-3}$$

$$\alpha(\boldsymbol{h}, \theta, A_s) = \begin{cases} \alpha_1(\boldsymbol{h}, \theta, A_s) & \text{if } \boldsymbol{h} \text{ is a selection or expansion node} \\ \alpha_2(\boldsymbol{h}, \theta, A_s) & \text{if } \boldsymbol{h} \text{ is a simulation node.} \end{cases} \tag{COMB$^{\triangleright}$-4}$$

$$\Psi = \Psi_1 \sqcup \Psi_2 \tag{COMB-5}$$

$$\xi(s, \boldsymbol{h}) = \xi_1(s, \boldsymbol{h}) \mathbin{+\!\!+} \xi_2(s, \boldsymbol{h}) \tag{COMB-6}$$

$$\omega(\theta, \psi, \boldsymbol{\mu}) = \begin{cases} \omega_1(\theta, \psi, \boldsymbol{\mu}) & \text{if } \theta \in \Theta_1 \\ \omega_2(\theta, \psi, \boldsymbol{\mu}) & \text{if } \theta \in \Theta_2 \end{cases} \tag{COMB-7}$$

**Specification 6:** Sequential composition ($\triangleright$)

The *MCTS-Solver* enhancement introduced by Winands et al [15, 55] works by backpropagating game-theoretic values through the tree. A terminal state is always known to be a win or a loss; at a decision node for player $p$, if one of the children is a known win then the node itself is a known win; if all of the children are known losses then the node itself is a known loss. This can be implemented by allowing nodes to take reward values of $+\infty$ and $-\infty$ to represent known wins and losses respectively, and modifying backpropagation to handle these values appropriately.

## *4.4. Combining ICARUSes*

For a particular domain, the most effective information reuse approach is often a combination of other approaches. Thus it is useful to have well-defined ways to combine ICARUSes.

In this paper we consider three ways of combining ICARUSes. The first is *sequential combination.* For two ICARUSes $I_1 = (R_1, \Theta_1, \theta^1_{\text{initial}}, \alpha_1, \Psi_1, \xi_1, \omega_1)$ and $I_2 = (R_2, \Theta_2, \theta^2_{\text{initial}}, \alpha_2, \Psi_2, \xi_2, \omega_2)$, the combination $I_1 \triangleright I_2$ is defined in Specification 6. Here $\sqcup$ denotes disjoint union: the sets are assumed to be disjoint, by relabelling elements if necessary. Each enhancement maintains its own records and information; the policy functions are combined so that $I_1 \triangleright I_2$ uses the policy from $I_1$ during selection and expansion, and the policy from $I_2$ during simulation. Selection and expansion nodes are defined in Section 4.2.

The second way of combining enhancements is *linear combination.* For two ICARUSes $I_1$ and $I_2$ as above, and a function $\lambda : \Theta^{\text{base}} \to [0, 1]$ (the mixing coefficient, which is a function of the information for the baseline ICARUS as defined in Specification 1), the combination $\lambda I_1 + (1 - \lambda) I_2$ is defined as in

Specification 6 with the exception of the policy function:

$$\begin{aligned} \alpha(\boldsymbol{h}, \theta, A_s) &= \lambda \alpha_1(\boldsymbol{h}, \theta, A_s) + (1-\lambda)\alpha_2(\boldsymbol{h}, \theta, A_s) \\ &\text{where } \lambda = \lambda\left(\theta^{\text{base}}\left([\boldsymbol{h}]^{\smile}\right)\right) . \end{aligned} \tag{COMB$^{+}$-4}$$

We can generalise this to define any convex combination of two or more enhancements in the natural way.

The third combination type is *maxilinear combination*. This is valid only for ICARUSes where the policy function has the form

$$\alpha(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\operatorname*{arg\,max}_{a \in A_s} v(a)\right] \tag{13}$$

for some function $v : A \to \mathbb{R}$. For two ICARUSes $I_1$ and $I_2$ satisfying this condition with functions $v_1$ and $v_2$ respectively, and a function $\lambda : \Theta^{\text{base}} \to [0,1]$, the combination $\lambda I_1 \oplus (1-\lambda) I_2$ is defined as in Specification 6 with the exception of the policy function:

$$\begin{aligned} \alpha(\boldsymbol{h}, \theta, A_s) &= \mathbb{U}\left[\operatorname*{arg\,max}_{a \in A_s} \left(\lambda v_1(a) + (1-\lambda) v_2(a)\right)\right] \\ &\text{where } \lambda = \lambda\left(\theta^{\text{base}}\left([\boldsymbol{h}]^{\smile}\right)\right) . \end{aligned} \tag{COMB$^{\oplus}$-4}$$

For example, this allows us to define RAVE [8] as

$$I_{\text{RAVE}} = \lambda_{\text{RAVE}} I_{\text{AMAF}} \oplus (1-\lambda_{\text{RAVE}}) I_{\text{Baseline}} \tag{14}$$

where

$$\lambda_{\text{RAVE}}(\boldsymbol{q}, n, m) = \sqrt{\frac{k}{3n+k}} \tag{15}$$

for some constant $k$ (which specifies the number of visits, i.e. the value of $n$, for which $\lambda_{\text{RAVE}} = 0.5$). Again, maxilinear combination can be generalised to combine more than two ICARUSes.

All ways of combining ICARUSes make use of information from the baseline definition (Section 4.2) in some way, whether to determine the current stage (selection, expansion or simulation) of the playout or to vary the combination coefficient. Thus for a combination to make sense, it must incorporate the baseline ICARUS.

### *4.5. Convergence properties*

Kocsis and Szepesvári [7] prove that, for games of perfect information, UCT converges on the optimal move in the limit. That is, as the number of iterations tends to infinity, the probability of selecting a suboptimal move tends to zero.

**Definition 6.** Consider a history $\boldsymbol{h}$, which when applied to the initial game state $s_0$ gives a state $f(s_0, \boldsymbol{h}) = s$ with legal actions $A_s$. Let $A_s^* \subseteq A_s$ be the set of optimal actions from state $s$. An ICARUS $I$ with policy $\alpha$ is *convergent* if, for all $a \in A_s \setminus A_s^*$, we have

$$\lim_{\text{iterations}\to\infty} \alpha(\boldsymbol{h}, \theta, A_s)(a) \to 0\,. \tag{16}$$

That is, for every suboptimal action $a$, the probability assigned to $a$ by the playout policy tends to zero in the limit.

For the baseline ICARUS (Specification 1) applied to a game of perfect information, we have the following two results:

**Lemma 1.** *The baseline ICARUS is convergent.*

*Proof.* It follows immediately from [7, Theorem 5] that (16) holds for $\alpha = \alpha^{\text{base}}$. □

**Lemma 2.** *There exists an iteration number $t$ such that, after $t$ iterations, $\boldsymbol{h}$ is a selection node.*

*Proof.* From [7, Theorem 3], there exists a constant $k$ such that, after $t$ iterations, the number of visits to $\boldsymbol{h}$ is at least $\lceil k \log t \rceil$. In particular there is a $t$ such that $\lceil k \log t \rceil \geq 2$, which implies that $\boldsymbol{h}$ is expanded and is now a selection node. □

From these results, we can easily show that certain combinations of ICARUS are convergent:

**Theorem 1.** *Let $I_1$ and $I_2$ be ICARUSes such that $I_1$ is convergent. Let $\lambda : \Theta^{\text{base}} \to [0, 1]$ such that $\lambda(\boldsymbol{q}, n, m) \to 0$ as $n \to \infty$. Then the following ICARUSes are convergent:*

*(i)* $\lambda I_2 + (1 - \lambda) I_1$*;*

*(ii)* $\lambda I_2 \oplus (1 - \lambda) I_1$ *(if defined);*

*(iii)* $I_1 \triangleright I_2$.

*Proof.* The convergence of (i) and (ii) follows from the fact that $\lambda$ tends to 0 as the number of visits to a node tends to infinity. This ensures that $I_1$ dominates in the limit, so the combination inherits its convergent behaviour.

The convergence of (iii) follows from Lemma 2: after some finite number of iterations, all nodes are selection nodes (recall from Definition 1 that we require games to have a finite number of states). At this point, $I_1 \triangleright I_2$ behaves identically to $I_1$ and thus converges. □

It follows from Lemma 1 and Theorem 1 (ii) that RAVE (14) converges. The ICARUS combinations used in the experiments in Section 8 (Table 1) all have the form $I_1 \triangleright I_2$ for $I_1 \in \{I_{\text{Baseline}}, I_{\text{RAVE}}\}$, and so also converge.

Note that these convergence results only apply to games of perfect information. For games of imperfect information, we have no proof equivalent to that of Kocsis and Szepesvári [7] that ISMCTS converges. Indeed, we have some empirical evidence that ISMCTS does not converge in the sense of Definition 6, either oscillating between several policies or settling on a policy which does not form part of a Nash equilibrium. Nevertheless, designing enhancements that converge in the perfect information case seems to be a useful way to obtain plausible play across all domains.

## 5. EPisodic Information Capture and reuse (EPIC)

Many games are episodic in nature: multiplayer games have a sequence of opponents' turns; ladder games such as Dou Di Zhu [31], President and Cheat [56] have a sequence of moves until a "reset" action occurs; strategic board and card games such as Lord Of The Rings: The Confrontation [30] and Magic: The Gathering [57] have compound turns consisting of several individual decisions.

In this section we introduce *EPisodic Information Capture and reuse (EPIC)*, an enhancement designed within the ICARUS framework. The unique feature of EPIC is how information is captured, i.e. which states are considered to be correlated. We consider a game to be divided into a number of time windows called *episodes*, and share information between states that correspond to the same position in different episodes. That is, states reached by the same sequence of actions from the beginning of their respective episodes, but where the starting points of those episodes may be different. The aim of information capture and reuse is to exploit the correlations between the values of nodes in different parts of the game tree. EPIC is designed to exploit the correlation between *subtrees* rather than individual nodes.

If the episodes truly are independent, this implies that the strength of a policy for a particular episode does not depend on the context of where that episode occurs in the game. Thus strong play overall can be achieved by constructing a good policy for each episode, and combining these policies to obtain a policy for the full game. The fact that the same episode occurs in several different parts of the game tree implies that a naïve tree search algorithm must rediscover the strong episode policy many times. EPIC aims to discover the episode policy only once, and reapply it throughout the game tree.

The assumption that episodes are independent of context may be reasonable but is never strictly true in real games. In this paper we combine EPIC with the baseline player, with EPIC used only as a simulation policy. This ensures that the baseline tree policy can tailor itself to the context of the current episode if that context matters, whilst the simulation policy that uses episode information but ignores context is still likely to be much stronger than a random policy.

The idea of episodes is not specific to any particular game, but it is also not universal. Games such as Chess and Go do not have a natural episodic structure, or rather the highly spatial nature of these games means that a purely temporal notion of episode does not make sense. However, even for these games, notions such as combinations in Chess [58] and joseki or tesuji in Go [59, 60] are a type of spatial episode. In this paper we will consider only temporal episodes, consisting of consecutive game turns. Nevertheless, a spatially episodic nature could conceivably be exploited by enhancements similar to EPIC.

## *5.1. Definition of EPIC*

Although episodes do not feature in the formal definition of a game given in Section 3, the division into episodes is usually highly intuitive when it exists, for example identifying the start of a new round or hand in a card game or a particular event happening such as a piece being captured. In other words, it is a piece of domain knowledge which does not require expert insight into the game. In most cases the episode information can be read directly from an implementation's representation of the game state.

We now introduce some notation for dealing with episodes. Let $E$ be a finite nonempty set of *episode labels*. Define $e : S \to E \cup \{\text{〃}\}$, the *episode function*. The element $\text{〃} \notin E$ is the *continuation label*.

Consider a game $\Gamma$. An *episode* of $\Gamma$ is a subtree of $\Gamma$'s game tree such that

1. for the subtree's root node $s_r$ we have $e(s_r) \in E$;
2. for all leaf nodes $s_l$ of the subtree, either $e(s_l) \in E$ or $s_l$ is a terminal state;
3. for all other nodes $s$ we have $e(s) = \text{〃}$.

If the initial state has episode label in $E$ then the episodes partition the game tree, as illustrated in Figure 1.

The *position-in-episode* of a history $\langle a_1, \ldots, a_t \rangle \in A^*$ is the pair

$$\eta(\langle a_1, \ldots, a_t \rangle) = (e(s_i), \langle a_{i+1}, \ldots, a_t \rangle) \tag{17}$$

where $s_i = f(s_0, \langle a_1, \ldots, a_i \rangle)$ and $i$ is maximal such that $e(s_i) \neq \text{〃}$. The position-in-episode specifies the label of the current episode and the suffix of the history restricted to that episode. If a state has a position-in-episode of $(e_1, \langle a_1, \ldots, a_t \rangle)$, then that state can be reached by starting from some state with episode label $e_1$ and applying actions $a_1, \ldots, a_t$. In particular if $e(s) \neq \text{〃}$, the position-in-episode of state $s$ is $(e(s), \langle\rangle)$. The two circled nodes in Figure 1 have the same position-in-episode. The position-in-episode of a move history is defined similarly to that of a state. Positions-in-episode are always defined relative to the initial state $s_0$ of the game, regardless of the current state.

Conceptually an episode should be somewhat independent of its context: when episodes with the same root label appear in different parts of the game tree, they should look similar and have similar outcomes for all players. In other words, if two different histories have the same positions-in-episode, they should have similar rewards. This statement is deliberately vague: the assignment of episode labels to states is a decision to be made when designing the AI agent rather than a part of the formal game definition. Examples of episode functions for our experimental domains are given in Section 7.

Specification 7 defines EPIC with game specific episodes as an ICARUS. The records used by EPIC are the positions-in-episode (EPIC-1), each of which has

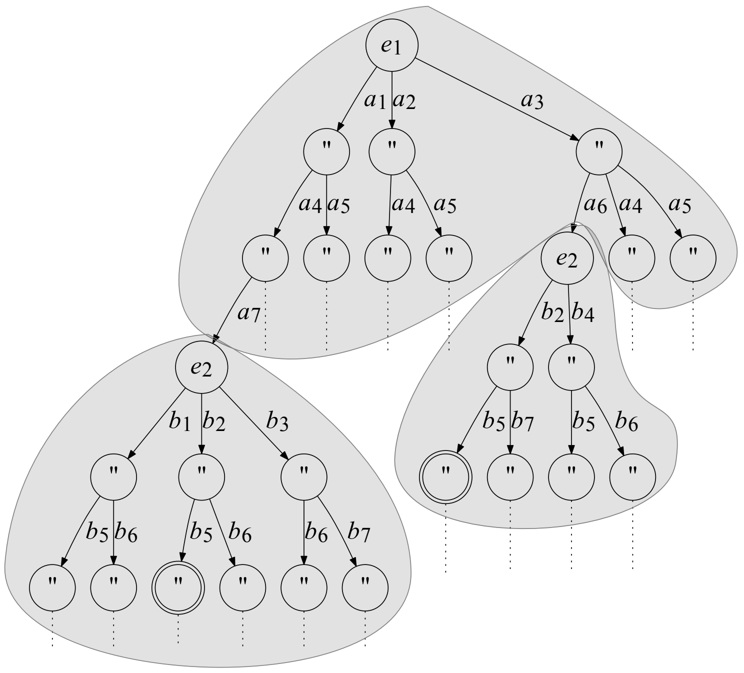

Figure 1: In this game tree, each node is labelled with its episode label. The shaded regions show the partitioning of the tree into episodes. The circled nodes have the same position-in-episode, namely $(e_2, \langle b_2, b_5\rangle)$.

$$R = E \times M^* \tag{EPIC-1}$$

$$\Theta = \mathbb{R}^\kappa \times \mathbb{N}_0 \times \mathbb{N}_0 \tag{EPIC-2}$$

$$\theta_{\text{initial}}(e, \boldsymbol{h}) = (\boldsymbol{0}, 0, 0) \tag{EPIC-3}$$

$$\alpha(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\operatorname*{arg\,max}_{a \in A_s} v(\theta(\eta([\boldsymbol{h} +\!\!+ a]^{\smile})))\right] \tag{EPIC-4}$$

$$\text{where } v((\boldsymbol{q}, n, m)) = \begin{cases} \dfrac{\boldsymbol{q}_\rho}{n} + c_{\text{EPIC}}\sqrt{\dfrac{\log m}{n}} & \text{if } n > 0 \\ +\infty & \text{if } n = 0 \end{cases}$$

$$\Psi = \{\psi_{\text{avail}}, \psi_{\text{visit}}\} \tag{EPIC-5}$$

$$\begin{aligned} \xi(s, \langle a_1, \dots, a_t \rangle) = {} & \langle (\eta([\langle a_1, \dots, a_i \rangle]^{\smile}), \psi_{\text{visit}}) \ : \ 0 \le i \le t \rangle \\ & +\!\!+ \langle (\eta([\langle a_1, \dots, a_{i-1}, a \rangle]^{\smile}), \psi_{\text{avail}}) \ : \ 0 < i \le t, \\ & \qquad a \in A(f(s, \langle a_1, \dots, a_{i-1} \rangle)), a \neq a_i \rangle \end{aligned} \tag{EPIC-6}$$

$$\omega((\boldsymbol{q}, n, m), \psi, \boldsymbol{\mu}) = \begin{cases} (\boldsymbol{q} + \boldsymbol{\mu}, n+1, m+1) & \text{if } \psi = \psi_{\text{visit}} \\ (\boldsymbol{q}, n, m+1) & \text{if } \psi = \psi_{\text{avail}} \end{cases} \tag{EPIC-7}$$

**Specification 7:** Episodic information capture and reuse (EPIC)

the standard ISMCTS information of total reward, visit count and availability count (EPIC-2, EPIC-3). The positions-in-episode for a particular episode label can be organised into a tree structure. Each history is mapped to its position-in-episode. If EPIC is combined with the baseline algorithm using sequential combination then during simulation, subset-armed UCB1 selection is applied according to the current position-in-episode (EPIC-4): effectively this means that the simulation policy for the overall search is provided by the tree policy in the episode trees. Rewards are backpropagated as in the baseline case, but in the episode trees rather than the full tree (EPIC-5, EPIC-6, EPIC-7).

## 6. Comparing ICARUSes

Having a common notation for information reuse enhancements provides a tool for their analysis and comparison. We can identify common themes that occur in several enhancements, easily see which enhancements deviate from them, and use them as building blocks to define new enhancements. For example, we can make the following observations from Specifications 1–4 and 7:

- Baseline, AMAF and EPIC all have records that are, or contain, move histories (Base-1, AMAF-1, EPIC-1). This implies that their records have a tree or forest structure. In contrast, MAST and LGR have a flat record structure with one record per move or action (MAST-1, LGR-1). NAST is between the two extremes, with one record per sequence of $n$ moves (NAST-1).

- For all enhancements studied here with the exception of LGR, the information associated with a record consists of a total reward (vector or scalar), a visit count, and in some cases an availability count (Base-2, AMAF-2, MAST-2, NAST-2, EPIC-2).

- Baseline, AMAF, NAST and EPIC all use some variation on the UCB1 policy (Base-4, AMAF-4, NAST-4, EPIC-4).

- LGR is the only enhancement whose policy explicitly sets the probabilities of some moves to 1, giving a simulation policy that is more deterministic than for other enhancements (LGR-4).

- If we ignore updates for availability, most of these ICARUSes update one record per state visited in the playout (Base-6, MAST-6, LGR-6, NAST-6, EPIC-6). The exception is AMAF, which potentially updates several records per playout step (AMAF-6).

At a higher level, we can identify sets of nodes which share information. There are two ways in which information can be shared between nodes via records:

1. nodes use information from the same record during playout and update that record during backpropagation; or

2. nodes use information from different records but a visit to one node causes both records to be updated.

In other words, to share information we can either write to one record that is read by many nodes, or write to many records that are each read by one node. MAST, NAST, LGR and EPIC are of the former type, whereas AMAF is of the latter type.

Figure 2 illustrates this information sharing. In each tree, information is shared between the nodes connected by a dotted line. Comparing these types of pictures with our intuition about the game, particularly which states we expect to be correlated, can give us insight into which enhancements are likely to work well for which games. Looking at Figure 2 and the corresponding ICARUS definitions, we can make the following observations about the type of games for which each enhancement should be effective:

- AMAF: Works well for games where the strength of an action is often independent of time. This has shown to be true in games such as Go [3] and Hex [10], where the RAVE enhancement works well. Both of these games are characterized by moves which, with a few exceptions, are fixed in place once made.

- MAST: Effective when the quality of an action is independent of the state from which it is played. This is effective in General Game Playing where little is known about the state and it is somewhat true for most games. (Arguably any interesting game that people would play will at least slightly

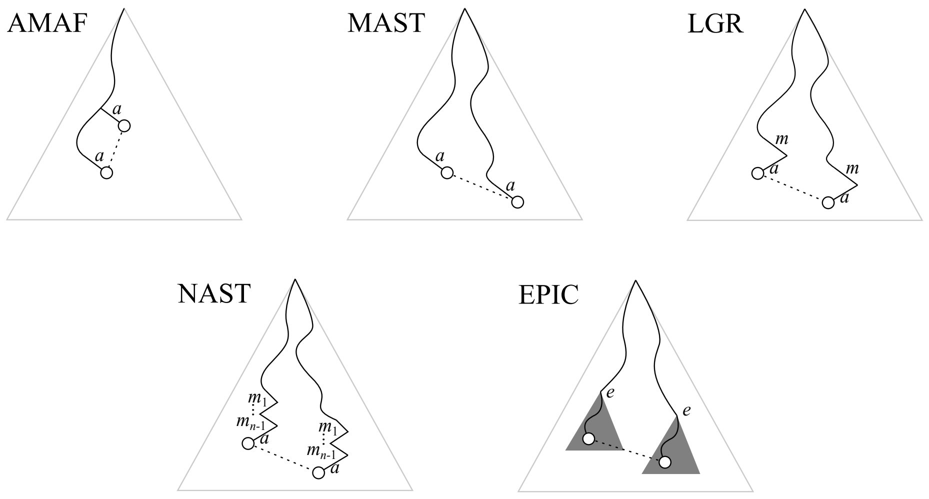

Figure 2: A pictorial representation of node correlation for various information capture and reuse strategies. In each case, the strategy is based on the notion that the values of the nodes connected by a dotted line are correlated, and thus sharing information between those nodes is beneficial.

satisfy this property, as it would otherwise be difficult for human players to develop an intuition of strategy). This is similar to AMAF, but assumes even less about the dependence of action value on context.

- LGR: Works well for games with a notion of "sente" (such as Go), where some moves imply a certain reply is necessary.
- NAST: Generalises the notion of replies in LGR, so is useful when the reply is to the previous $n-1$ moves rather than the previous move. Note that LGR and NAST with $n = 2$ both use the same correlation structure, so any difference in their performance is a result of the exact way in which the captured information is reused in the playouts.
- EPIC: Useful in games with an episodic nature (e.g. ladder-based card games) or complex compound turns (such as LOTR:C; see Section 7.3), where contextual information can be learned from these episodes.

The information capture methods of MAST, LGR, NAST and EPIC all belong to a class which learns the value of actions given some episode preceding the action. In the case of MAST, LGR and NAST the episodes are of fixed length (1, 2, and a fixed parameter $n$ respectively) whereas EPIC uses episodes that are aligned to the natural episodic structure of a particular game. MAST, LGR and NAST are based on the game-independent notion of $n$-grams with a fixed $n$, whereas EPIC allows $n$ to differ over the playout according to game-specific criteria.

There are also possibilities for other new types of enhancement that can be built with the ICARUS framework. Figure 2 illustrates enhancements which exploit a particular type of correlation between states in a game, but any other type of correlation can be encoded as an information capture enhancement within the ICARUS framework. Similarly there are many ways of injecting knowledge into the MCTS algorithm, with tree selection biasing and heavy playouts being amongst the most common [1, 61]. New techniques for injecting knowledge into MCTS can be combined with any existing information reuse enhancement. One potential application for the ICARUS framework could be to use techniques from machine learning (for example genetic programming) to automatically search the space of possible enhancements to discover new ones which work for a particular game. There are currently no widely used enhancements to MCTS which deal explicitly with hidden information and uncertainty in games. ICARUS provides a framework for exploring new enhancements in this area, for example considering information asymmetry when performing information capture.

## 7. Experimental domains

In this section we present the games that form the basis of our empirical analysis. For each game, we also present the corresponding EPIC episode structure.

### *7.1. Dou Di Zhu*

Dou Di Zhu is a three-player ladder-based (or *climbing*) card game, played with a standard deck of 52 cards plus two jokers, hugely popular in China [31, 62]. One of the players is designated the *Landlord*, with the other two players in coalition against him. At the beginning of the game, the entire deck is dealt to the players: 20 cards to the Landlord, and 17 to each of the other players. The game is played in several rounds, with each round consisting of a *leading play* (a combination of cards from one of several categories: singleton, pair, straight etc.) after which players take turns to play cards of the same category but higher rank, or pass if they are unable or unwilling to play cards. The winner is the first player to play out all of the cards from their hand; if the winner is a non-Landlord player then both non-Landlord players win. For a complete description of the rules see [62].

Our previous work on this game [63, 31, 30] showed that ISMCTS is competitive with existing commercial AI [30], although a simpler multiple tree determinization approach (similar to those used for Bridge [64] and Klondike Solitaire [34]) performs slightly better. The main strategic element in Dou Di Zhu is subdividing one's hand into non-overlapping groups, given that some groups may render others unplayable (for example if holding cards $3, 4, 4, 5, 6, 7$, playing the pair $4, 4$ means that the straight run $3, 4, 5, 6, 7$ is unavailable). All cards must eventually be played, so playing groups in such a way that one is not left with a weak hand towards the end of the game is key.

We define the EPIC episode function for Dou Di Zhu as follows. Set $E = \{L_1, L_2, L_3\}$ and

$$e(s) = \begin{cases} L_i & \text{if player } i \text{ makes a leading play from state } s \\ \text{''} & \text{otherwise.} \end{cases} \tag{18}$$

Here an episode is a stream from one leading play to the next. EPIC aims to improve on a weakness of ISMCTS for Dou Di Zhu caused by the high branching factor associated with leading plays (several hundred in some cases [31]). EPIC helps to more accurately model the sequence of plays that follows each leading play, and thus give a more accurate evaluation of each leading play, without having to rediscover this sequence in several parts of the tree.

## *7.2. Hearts*

Hearts is a four-player trick taking card game played with a standard 52-card deck. Cards in the $\heartsuit$ suit have a point value of 1 each and the $Q\spadesuit$ card has a point value of 13. The goal is to score as few points as possible, i.e. to avoid winning tricks with those cards in them. There is one exception to this goal: taking all thirteen $\heartsuit$ cards and $Q\spadesuit$ in a round is called *shooting the moon*, and causes every player to score 26 points except the player who shot the moon. Shooting the moon is a risky strategy: the reward for success is high, but so is the cost of failure. For a complete description of the rules see [65].

Previous work on MCTS for Hearts [66] has treated the game as one of perfect information, i.e. played with all cards face up. We use ISMCTS to handle the imperfect information explicitly. Other approaches applied to Hearts include $\max^n$ search with various pruning mechanisms and reinforcement learning [67, 68].

Hearts is played over several rounds, the game ending when any player reaches a score threshold (50 points in our experiments). It would be possible to simulate only to the end of the current round and have our players seek to minimise their per-round score. However this removes some of the strategic richness from the game, as certain decisions (such as whether to force $Q\spadesuit$ upon a certain player, or whether to attempt shooting the moon) can depend on the overall game score. To capture this, we simulate offline a large number of rounds (10 000 for our experiments), and construct a database of per-round scores. During the search, we simulate to the end of the current round as usual, and then sample round scores from this database until the score threshold is reached. This is equivalent to simulating to the end of the game, but much more efficient.

To define the episode function for EPIC, we set $E = \{D, L_1, L_2, L_3, L_4\}$ and

$$e(s) = \begin{cases} D & \text{if } s \text{ is the chance state for the deal in a new round} \\ L_i & \text{if player } i \text{ begins a trick from state } s \\ \text{''} & \text{otherwise.} \end{cases} \tag{19}$$

The first episode in a game of Hearts encompasses the dealing and card passing stages; subsequent episodes are single tricks. Here EPIC makes use of the fact that similar tricks may appear in many different places in the search tree.

## *7.3. Lord of the Rings: The Confrontation*

Lord of the Rings: The Confrontation (LOTR:C) [69] is a two-player strategy board game. Each player has nine character pieces, each with its own strength value and special ability. Each player can see the identities and locations of his own pieces, but only the locations of his opponent's pieces. If a player moves a piece into a square occupied by his opponent, combat ensues: the identities of the attacking and defending pieces are revealed, and the players simultaneously choose a card each which affects the outcome of combat. The two players have asymmetric win conditions: the "Light" player wins by moving one of his pieces (Frodo) into the opponent's home square (Mordor), whereas the "Dark" player wins by killing Frodo in combat.

ISMCTS performs strongly for LOTR:C [30], both against determinized MCTS approaches and against human players. We also found that the asymmetric nature of the game has an influence on the relative strengths of AI techniques: handling hidden information is important to the Dark player, whereas for the Light player it is more beneficial to concentrate on planning one's own moves further ahead.

For EPIC's episode function, we set $E = \{M_1, M_2\}$ and

$$e(s) = \begin{cases} M_i & \text{if there is no combat in progress and } \rho(s) = i \\ \text{〃} & \text{otherwise.} \end{cases} \tag{20}$$

An episode begins when a player moves a piece. If this movement does not result in combat, the episode ends immediately. Otherwise, the episode continues until combat is resolved. The benefit of EPIC here is twofold: it collects statistics for each movement action in a manner similar to MAST, and it refines the simulation policy for combat.

## *7.4. Checkers*

Checkers, or Draughts, is a two-player game of perfect information, played on an $8 \times 8$ board with 12 pieces per player. Pieces may be moved forwards to a diagonally adjacent empty square, or may jump diagonally forwards by two squares if the target square is empty and the intervening square contains an opponent piece. Jumping over an opponent's piece causes it to be *captured*, and removed from the game. Captures may be chained together, if the jumping piece can immediately capture another piece. Otherwise the turns alternate between the two players after each move. In the variant of Checkers studied in this paper, captures are forced: if a capture move is available then it must be played, although if more than one is available the player may choose which one to take. If a piece moves onto the opponent's home row, it becomes *crowned* and may subsequently move and capture backwards as well as forwards. A player wins by leaving their opponent with no legal moves, i.e. by blocking or capturing all their pieces.

Draws (stalemates) are common in checkers; indeed, perfect play by both sides will always lead to a draw [70]. AI programs capable of perfect play exist, such as Chinook [70]. As Checkers was solved more than a decade before the invention of MCTS, there has been little work on developing strong MCTS players. However Checkers is often used as a test domain for enhancements in General Game Playing systems [46, 45].

To apply EPIC to Checkers, we set $E = \{M_1, M_2\}$ and

$$e(s) = \begin{cases} M_i & \text{if a non-chained capture move is available, and } \rho(s) = i \\ \text{''} & \text{otherwise.} \end{cases} \tag{21}$$

Here a "non-chained" capture is one which does not continue a chain of captures, but may start one. An episode begins when a capture is made, and ends the next time a capture is made (by either player) on a subsequent turn.

## *7.5. Othello*

Othello, or Reversi, is a two-player game with perfect information, played on an $8 \times 8$ board. The game starts with the centre four squares of the board containing two black and two white pieces placed diagonally opposite each other. A move consists of placing a piece on the board; for the move to be legal, it must sandwich a horizontal, vertical or diagonal line of one or more opponent pieces between the newly placed piece and an existing own piece. The sandwiched pieces are captured, and converted to the colour of the player who moved. If (and only if) a player has no legal moves, he must pass; when both players pass consecutively, the game is over. The player with the most pieces on the board wins the game.

Strong Othello programs exist which are capable of beating the strongest human players, one of the first such programs being Logistello [71]. More recently, MCTS has been combined with offline learning methods to produce strong play [72, 73, 74].

The EPIC episode function is defined by $E = \{M_1, M_2\}$ and

$$e(s) = \begin{cases} M_i & \text{if the previous move was on the edge of the board, and } \rho(s) = i \\ \text{''} & \text{otherwise.} \end{cases} \tag{22}$$

An episode begins on the turn after a player places a new piece on one of the 28 squares around the edge of the board. This captures the strategic notion that controlling the edges of the board is important: pieces placed on the edge are difficult to capture but create opportunities to capture opponent pieces.

## *7.6. Backgammon*

Backgammon is a two-player game which has stochasticity in the form of dice rolls, but otherwise has perfect information (i.e. there is no information hidden from one player but visible to another). The board has 24 spaces, which are numbered 1–24 in opposite directions for the two players. Each player begins with 15 pieces in a standard initial setup. The aim of the game is to move all of one's pieces towards space 1 and off the end of the board. A player's turn begins with a roll of two dice. The player then takes two moves, one for each of the two rolled numbers, moving a piece forward the given number of spaces. The same piece can be moved twice in one turn. If the two dice have the same number, the player makes four moves instead of two.

A piece cannot be moved to a space occupied by two or more opponent pieces. However a piece can be moved to a space occupied by a single opponent piece, in which case the opponent piece is captured and moved to the *bar*, equivalent to space number 25. If a player has pieces on the bar, they must move them back onto the board before they may move any other pieces. A common basic strategy in Backgammon is to force the opponent to skip several turns, by capturing a piece having blocked the spaces into which it could be moved back onto the board. When all of a player's pieces are on spaces 1–6, pieces may be moved off the board (beyond point 1) and removed from the game. The first player to remove all their pieces in this way is the winner.

Strong AI players for Backgammon, such as TD-Gammon [75], are capable of beating the strongest human players. MCTS has also been demonstrated to produce strong decisions in Backgammon [76].

The compound turns of Backgammon give a natural episode structure. We set $E = \{R\}$ and

$$e(s) = \begin{cases} R & \text{if the dice are about to be rolled} \\ {\prime\prime} & \text{otherwise.} \end{cases} \tag{23}$$

Thus an episode consists of a full turn: the dice roll and the two or four moves that follow it.

## 8. Experiments

A wide range of experiments were conducted to compare ICARUSes and combinations of ICARUSes for the six domains listed in Section 7. We aim to compare the performance of enhancements for games of perfect and imperfect information, and investigate whether combinations of enhancements can be greater than the sum of their parts.

In each experiment our opponent was an unenhanced player using MO-ISMCTS (for games of imperfect information) or UCT (for games of perfect information), with 5000 playouts per decision in all cases. In [30] we showed the MOSIMCTS player was on a par with the strongest setting of a commercial Dou Di Zhu AI developed by AI Factory Ltd[4]. In [30] we also showed that MO-ISMCTS is on a par with intermediate-to-expert level human players for

[4]`www.aifactory.co.uk`.

LOTR:C. For Hearts we conducted a trial of our MO-ISMCTS player against the AI for Hearts which ships with the Microsoft Windows 7 operating system (arguably one of the most played AI opponents of all time), again showing strength parity. For Checkers, Othello and Backgammon, unenhanced UCT is significantly worse than state-of-the-art AI players, but largely because the state-of-the-art is so advanced for these games. We have found unenhanced UCT to be around the level of a novice human player for these games.

We compare the sixteen combinations of ICARUSes listed in Table 1, for the six games described in Section 7. The component enhancements used are:

- the baseline ICARUS (Section 4.2);
- RAVE (Section 4.4), itself based on AMAF (Section 4.3.1);
- MAST (Section 4.3.2);
- LGRF, i.e. LGR with forgetting (Section 4.3.4);
- NAST (Section 4.3.5) with an $n$-gram length of $n = 2$.

Where the enhancements use parameters, we performed an initial round of parameter tuning experiments to set the values for the main experiment. The parameter values are $c = 0.7$ for the exploration constant in both the baseline player and NAST, $k = 250$ for RAVE, and $\tau = 1$ for MAST. These values give consistently good performance (relative to other parameter settings) across all our games.

In all games, a win has a reward value of 1 and a loss a value of 0. If a draw is possible, it has a value of 0.5. In Hearts, finishing in first place has a value of 1, in last place a value of 0, and in second or third place a value of 0.5. These values are used by the MCTS players, but not in the reporting of results in this section: that is, Figures 3–5 count the win rate (i.e. number of wins), not the cumulative reward over all trials.

The ICARUSes listed in Table 1 cover all subsets of the four enhancements tested here. Each enhancement was originally designed for a specific phase of the MCTS iteration: RAVE [8] for selection, and MAST [43], LGR [47] and NAST [27] for simulation. The sequential combinations, using the $\triangleright$ operator, are defined to use each enhancement only for the appropriate phase. Where we have more than one enhancement for one of the phases, we combine them using a linear combination (as described in Section 4.4) with equal weights. Equivalently, at each step in the simulation we choose one of the enhancements at random and then choose the action according to that enhancement's policy. For example if the simulation enhancement is $\frac{1}{2}(I_{\mathrm{LGRF}} + I_{\mathrm{NAST}})$, the simulation plays according to LGRF with probability $\frac{1}{2}$ and according to NAST with probability $\frac{1}{2}$. The definition (Equation 14) of $I_{\mathrm{RAVE}}$ as a maxilinear combination of $I_{\mathrm{AMAF}}$ and $I_{\mathrm{Baseline}}$ with decaying weight $\lambda_{\mathrm{RAVE}}$ is as used by Gelly and Silver [8].

For a $\kappa$-player game, we play one instance of the ICARUS combination in question against $\kappa - 1$ instances of the baseline player (unenhanced MO-ISMCTS). Each algorithm (enhanced and baseline) uses 5 000 MCTS iterations

| Name | NAST | RAVE | MAST | LGRF | ICARUS specification |
|---|---|---|---|---|---|
| – | | | | | $I_{\text{Baseline}}$ |
| N | ● | | | | $I_{\text{Baseline}} \rhd I_{\text{NAST2}}$ |
| R | | ● | | | $I_{\text{RAVE}}$ |
| NR | ● | ● | | | $I_{\text{RAVE}} \rhd I_{\text{NAST2}}$ |
| M | | | ● | | $I_{\text{Baseline}} \rhd I_{\text{MAST}}$ |
| NM | ● | | ● | | $I_{\text{Baseline}} \rhd \frac{1}{2}(I_{\text{MAST}} + I_{\text{NAST2}})$ |
| RM | | ● | ● | | $I_{\text{RAVE}} \rhd I_{\text{MAST}}$ |
| NRM | ● | ● | ● | | $I_{\text{RAVE}} \rhd \frac{1}{2}(I_{\text{MAST}} + I_{\text{NAST2}})$ |
| L | | | | ● | $I_{\text{Baseline}} \rhd I_{\text{LGRF}}$ |
| NL | ● | | | ● | $I_{\text{Baseline}} \rhd \frac{1}{2}(I_{\text{LGRF}} + I_{\text{NAST2}})$ |
| RL | | ● | | ● | $I_{\text{RAVE}} \rhd I_{\text{LGRF}}$ |
| NRL | ● | ● | | ● | $I_{\text{RAVE}} \rhd \frac{1}{2}(I_{\text{LGRF}} + I_{\text{NAST2}})$ |
| ML | | | ● | ● | $I_{\text{Baseline}} \rhd \frac{1}{2}(I_{\text{MAST}} + I_{\text{LGRF}})$ |
| NML | ● | | ● | ● | $I_{\text{Baseline}} \rhd \frac{1}{3}(I_{\text{MAST}} + I_{\text{LGRF}} + I_{\text{NAST2}})$ |
| RML | | ● | ● | ● | $I_{\text{RAVE}} \rhd \frac{1}{2}(I_{\text{MAST}} + I_{\text{LGRF}})$ |
| NRML | ● | ● | ● | ● | $I_{\text{RAVE}} \rhd \frac{1}{3}(I_{\text{MAST}} + I_{\text{LGRF}} + I_{\text{NAST2}})$ |

Table 1: ICARUS combinations for the main experiment in Section 8

per decision. For Dou Di Zhu the ICARUS player plays as the Landlord against two non-Landlord baseline players. For Hearts the ICARUS player plays against three baseline players. For LOTR:C the ICARUS player plays as both Light and Dark against a baseline player (and due to the asymmetry of the game we consider the results for each player separately). For Checkers, Othello and Backgammon the ICARUS player plays against a baseline player, playing half the games as white and half as black. For each experiment we played a large number of games (between 1000 and 2500) across a cluster of PCs, using just over one CPU-year in total. Win rates are calculated with Clopper-Pearson intervals [77] at the 95% confidence level. Each game of Dou Di Zhu, LOTR:C and Backgammon has a clear winner with no possibility of a draw. In Hearts we count finishing in first or equal first place as a win, and any other outcome as a loss. In Checkers (where draws are common) and Othello (where draws are possible but rare), we only count the number of wins; draws are counted as losses.

Average results for each enhancement are presented in Figure 3. For each enhancement we aggregate the results for those combinations in Table 1 which feature the given enhancement, and compare with the results for those combinations which do not. For example in the pairs of bars labelled "RAVE", the left-hand (diagonally shaded) bar sums the results for the eight combinations in Table 1 for which the RAVE column is blank, and the right-hand (solid shaded) bar for the eight combinations where the RAVE column is marked •. Where each combination in Table 1 was tested for 1000 trials, each of the bars in Figure 3 represents 8000 trials.

Results for individual combinations are shown in Figure 4. The dotted lines indicate 95% confidence intervals for evenly-matched UCT (perfect information) or MO-ISMCTS (imperfect information) players. Any AI with results above this band is significantly better than an unenhanced player.

In Figure 3 we see that NAST provides a significant improvement for almost all games (the exceptions being LOTR:C as Light and Checkers, where there is no significant difference between players with and without NAST). Analysis of variance (ANOVA) over all results shows that NAST yields an improvement significant at the 99.9% level. It is interesting to note that NAST provides an improvement even in cases where LGRF does not, despite the two enhancements being based on a similar principle of learning the value of moves as replies to previous moves. This shows the influence of the precise way in which this information is used by the simulation policy.

Figure 3 also shows that RAVE is not effective in the imperfect information games. It is actually detrimental to performance in Hearts, and as the Dark player in LOTR:C, where the order in which moves are played is important to the strategy of the game. This makes intuitive sense: for example in Hearts the value of a particular card is very much dependent on when that card is played. For example, leading a trick with $K\spadesuit$ will be either a reasonably good or a terribly bad idea, depending whether the high-scoring $Q\spadesuit$ has already been played, so that AMAF/RAVE will fail to correctly learn whether $K\spadesuit$ is a good move. For Dou Di Zhu and LOTR:C as the Light player, RAVE did not make matters worse, but nor did it significantly improve playing strength. The only game where RAVE is significantly beneficial in our experiments is Checkers.

MAST, LGRF and NAST are more robust enhancements than RAVE: they are sometimes beneficial, and never detrimental to a statistically significant degree. As observed by Tom and Müller [25], RAVE performs poorly in situations where relative ordering of moves is important. However MAST is based on a similar principle, and is more robust. One intuition for this is that RAVE alters the tree policy in a way that can reinforce the assumption that move ordering is not important in situations where this assumption is false. MAST on the other hand learns whether a move is good or bad on average. This implies that if move ordering is not important to the goodness of a move, MAST will reinforce selection of this move, whereas if move ordering is important MAST will enforce no preference over selecting the move, falling back to the behaviour of the default random policy of MCTS. Thus the potential for MAST to learn the wrong thing, as opposed to learning nothing at all, is much less than for RAVE.

For MAST we conducted further experiments to investigate the impact of different reuse methods [27]. The policies in question were the four described in Section 4.3.2: Gibbs distribution, $\varepsilon$-greedy, roulette wheel and UCB1. We observed that $\varepsilon$-greedy and UCB1 offered the most consistently strong simulation policy, and that the Gibbs sampling approach which is standard in MAST is generally slightly worse than the other methods, although sensitivity to the sampling method is not terribly marked.

In Figure 4 we see that linear combinations of the enhancements are often greater than the sum of their parts. LGRF produced limited improvements on its own, but it enhanced other techniques (e.g. consider NL beat both N and

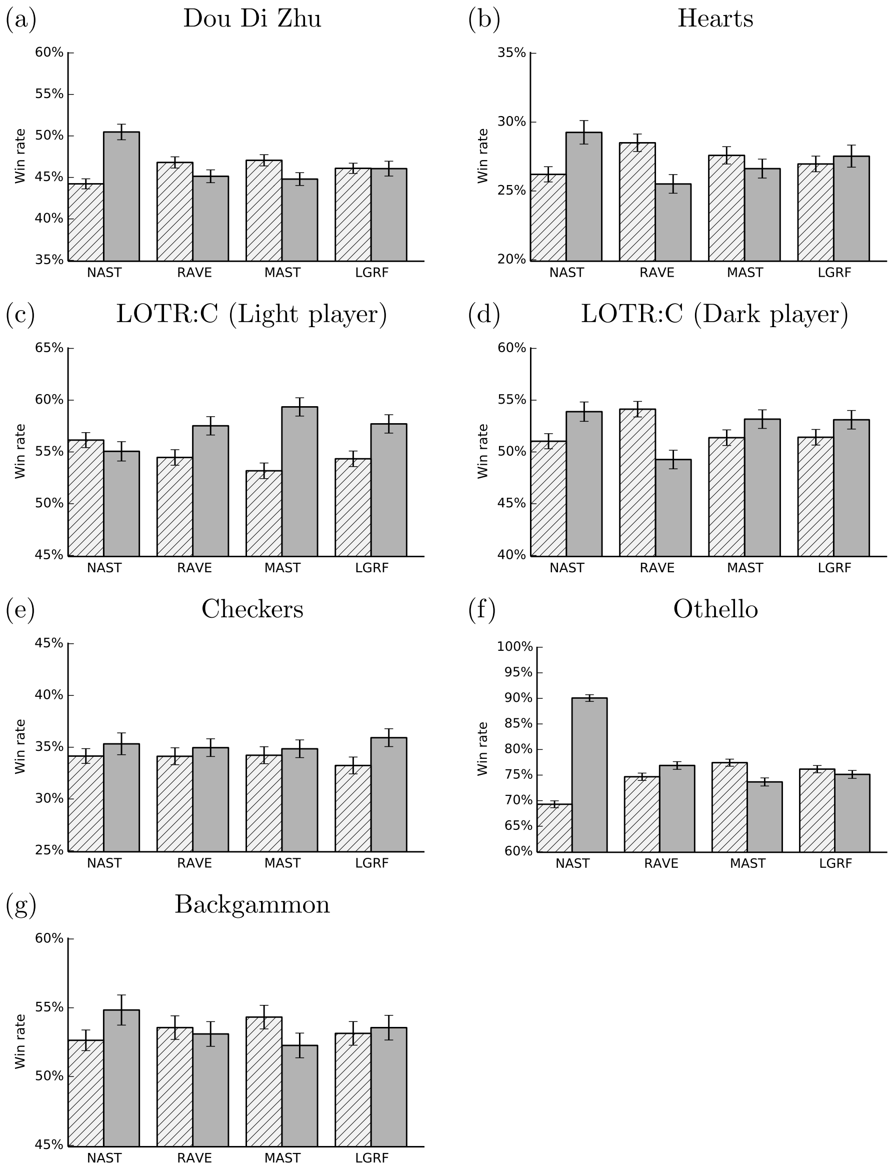

Figure 3: Average playing strength results for the four ICARUSes tested for each game. In each pair of bars, the left-hand bar is the average win rate over the eight combinations (out of the 16 listed in Table 1) not featuring the enhancement in question, and the right-hand bar the average over the eight combinations that do feature the enhancement.

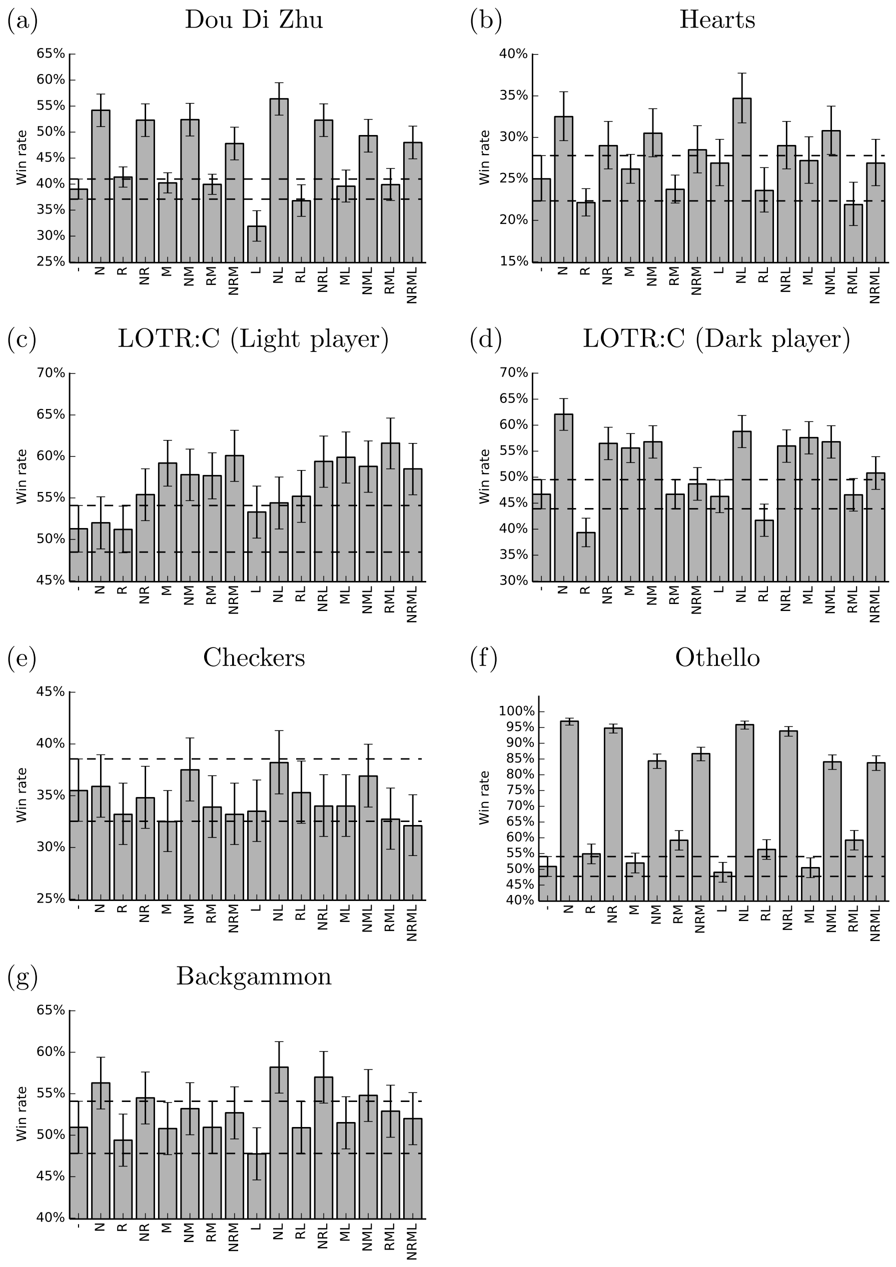

Figure 4: Playing strength results for ICARUS combinations (the names are listed in Table 1). For each ICARUS the percentage of games won is presented along with 95% confidence intervals. In each experiment 5000 MCTS iteration were used and the opponents were all baseline players (−). Dashed lines show the empirically measured 95% confidence interval for evenly-matched UCT or MO-ISMCTS players; any player with results above this baseline is significantly stronger than unenhanced UCT or MO-ISMCTS.

L for Dou Di Zhu, despite L being significantly worse than −, and ML beat both M and L for LOTR:C as Dark despite no significant difference between − and L). MAST was generally effective across all games, but proved even more effective in combination (e.g. consider RML beat all of R, M and L for LOTR:C as Light and ML beat both M and L for LOTR:C as Dark). NAST performed strongly across all games, but was most effective in combination for Dou Di Zhu (NL beat both N and L) and LOTR:C as Light (NRL beat all of N, R and L).

Analysis of the MCTS trees shows that the final number of visits for the chosen action is significantly (often 10–30%) higher for NAST than for the baseline and LGRF, RAVE or MAST. Hence NAST is converging more quickly to an action which is generally stronger. The average reward assigned by NAST to the chosen action is also markedly different from that in trees using other enhancements, and presumably this value is more accurate given the increased playing strength due to NAST. Of the enhancements tested here, only RAVE has a significant impact on the depth of the MCTS tree. RAVE tends to construct deeper trees, with the deepest node being on average one ply deeper than for the other algorithms. That RAVE is detrimental despite this suggests that it is expanding too deeply the "wrong" areas of the tree, although the fact that the degree of exploitation at the root is the same as without RAVE suggests that this mostly occurs at levels in the tree below the root.

Our experimental results for NAST shows that learning contextual values for moves is an effective way of improving the simulation policy. EPIC provides a way of refining this context in a game-specific way. To determine whether this refinement is helpful, we tested EPIC against NAST with $n = 2$. Both enhancements use UCB1 as a simulation policy, the difference being the context in which the multi-armed bandit statistics are collected: for EPIC the context is game-specific, whereas for $n = 2$ NAST the context is the previous move in the game (similar to LGRF). EPIC uses the episode functions for each game described in Section 7. Results are shown in Figure 5. In Othello we find that NAST is significantly better than EPIC; in all other cases, NAST and EPIC have the same performance within 95% significance. As a corollary to this, we can conclude that EPIC is at least as robust as NAST for the games studied (even in the case of Othello, EPIC significantly outperforms the baseline player).

The effectiveness of EPIC is particularly marked for Dou Di Zhu, which has a strongly episodic nature, since during an episode (a *ladder*) many cards cannot be played, but these unplayable cards *will* be played in future episodes. Furthermore, the order of episodes is not often important for Dou Di Zhu. There is also some independence between tricks in Hearts and between turns in LOTR:C and Backgammon, though less than between ladders in Dou Di Zhu, so that the improvements due to EPIC, while significant, are not so marked. Checkers and Othello lack a natural episodic structure, so instead we use basic notions of strategically important moves (captures in Checkers and edge piece placement in Othello) to delimit episodes. This works in both games, but for Othello proves less effective than simply fixing the episode length to 2 (i.e. using NAST).

EPIC remains an interesting method for injecting knowledge into search in

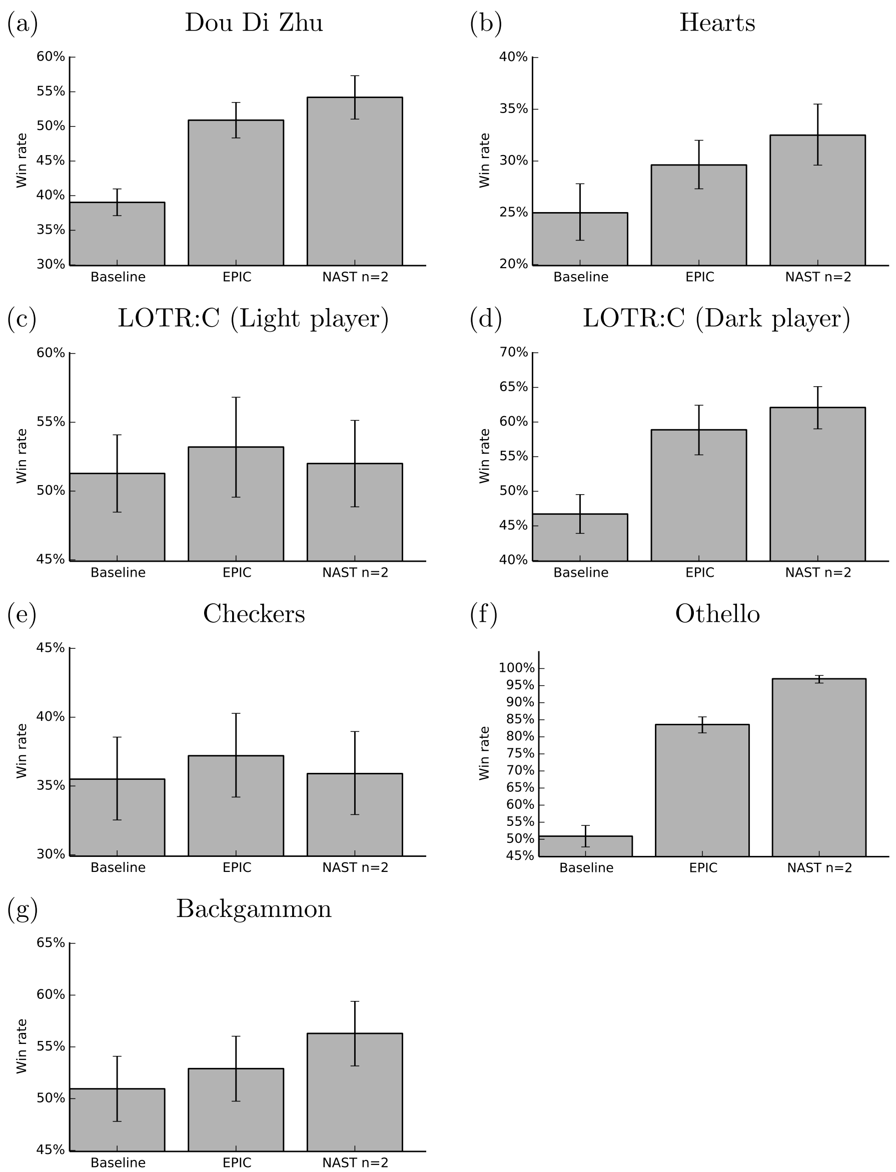

Figure 5: Playing strengths (against baseline players) for a baseline player ($I_{\text{Baseline}}$), a player using EPIC for simulations ($I_{\text{Baseline}} \triangleright I_{\text{EPIC}}$), and a player using NAST with $n$-gram length 2 ($I_{\text{Baseline}} \triangleright I_{\text{NAST2}}$, with $I_{\text{NAST2}}$ as defined in Specification 5 with $n = 2$) over 1000 trials each.

other game domains, but for these games it is clear that the episodes do not need to be so carefully chosen. Both EPIC and NAST are methods for learning useful lines of play for the playout policy; EPIC achieves this by learning fragments of lines delimited by game-specific episodes, whereas NAST with $n = 2$ essentially learns a Markov model. For the games tested here the Markov model is rich enough to capture useful information, but for other games a more sophisticated model such as EPIC may be required.

## 9. Conclusion and future work

In this paper we present the ICARUS framework, a powerful tool for expressing information capture and reuse enhancements in MCTS. The ICARUS framework is expressive enough to enable all existing MCTS enhancements to be defined and provides a new tool for discovery of novel types of enhancement. ICARUS provide a consistent method for expressing how an MCTS enhancement captures and reuses information, enabling us to easily analyse similarities and differences between enhancements and define composition operators which are compatible with all ICARUSes. We have presented information capture and reuse as the principles upon which all general purpose enhancements to MCTS are based, and separated the mechanisms for capture and reuse as a tool for understanding existing enhancements and designing new ones. We found the $\varepsilon$-greedy and UCB1 algorithms to be particularly effective at balancing the exploration and exploitation of moves during MCTS simulations.

Considering which states map to which records during playout and backpropagation gives a clear indication as to which parts of the tree share information. We conjecture that the effectiveness of information capture is determined by the degree of correlation of state values in these regions of the underlying game tree. In Section 6 we discussed how the effectiveness of these enhancements can be explained in the context of sharing information between states.

We developed the EPIC enhancement within the ICARUS framework by considering the notion of episodes, which turns out to generalise readily to several other games. Using episodes based on the episodic structure of each game proved to be effective across our test domains. Many games have a natural episodic nature and EPIC may prove to be useful in exploiting this. MAST, LGR and NAST may be viewed as techniques which reuse information based on short episodes, of length 1 for MAST, length 2 for LGR, and arbitrary fixed length $n \geq 1$ for NAST. The fact that NAST with $n = 2$ performs at least as well as EPIC in our experiments suggests that, for the games studied here, the choice of episode does not require careful injection of knowledge. This is unlikely to be true for all games.

The enhancements we consider are *general purpose*, in the sense that they can be applied to any game without injection of knowledge. RAVE, MAST, LGR and NAST are general purpose; strictly speaking EPIC is not, but the degree of domain knowledge required is small. General purpose enhancements are useful tools for tackling new domains where expert knowledge is not available, and essential for domains such as General Game Playing where input of external knowledge is not possible. No general purpose enhancement has yet been discovered that is beneficial in all domains, and the existence of such an enhancement seems unlikely, but some are more *robust* than others: even in domains where they provide no clear benefit, they are usually not detrimental. Robustness is an essential criterion in choosing a general purpose enhancement. We have demonstrated for our test domains that MAST, LGR, NAST and EPIC are robust while RAVE is not.

The ICARUS framework enables combination operators for enhancements to be defined, with the strongest play often coming from a combination of enhancements. One possible direction for future work is to develop more robust composition operators, for example ones based on majority voting rather than weighted sums, effectively adopting an ensemble approach to enhancement combination.

We have also shown that enhancements designed for perfect information games can be effective in imperfect information games, despite the increased level of uncertainty and sparsity of search. Current MCTS enhancements do not explicitly address information asymmetry and stochasticity in games. However we could define new ICARUSes that consider information asymmetry in information capture, for example by sharing information between states that are distinguishable to a player but indistinguishable to their opponent. The MCTS algorithm is also easily parallelisable [78], which suggests a new class of enhancements that capture information in one search thread and reuse it in others running concurrently. This would require modification of Algorithm 1 to perform a multithreaded variant of MCTS, and locking mechanisms to ensure the same record is not updated while another thread is reading or updating it, but otherwise ICARUS enhancements could be used without modification.

In future work we plan to investigate the automation of designing and choosing enhancements for a particular game, and ICARUS provides a framework for doing this. For example, we could directly measure the correlation between different areas of the tree and use this information to select the most appropriate enhancements from a predefined library. This could be done offline before the search begins, or online to dynamically activate and deactivate enhancements as the search progresses. Additionally, we can automatically discover new enhancements using evolutionary techniques; the ICARUS framework could give a compact yet expressive representation for genetic programming or other evolutionary algorithms. This kind of dynamically self-enhancing system combined with MCTS would take us several steps further towards a truly general purpose AI system for acting in challenging games and complex sequential decision problems.

## Acknowledgments

We thank Jeff Rollason of AI Factory Ltd (`www.aifactory.co.uk`) for providing software and invaluable input for our work on Dou Di Zhu. We also 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, S. Colton, A Survey of Monte Carlo Tree Search Methods, IEEE Trans. Comp. Intell. AI Games 4 (1) (2012) 1–43.

[2] C.-S. Lee, M. Müller, O. Teytaud, Guest Editorial: Special Issue on Monte Carlo Techniques and Computer Go, IEEE Trans. Comp. Intell. AI Games 2 (4) (2010) 225–228.

[3] S. Gelly, D. Silver, Monte-Carlo tree search and rapid action value estimation in computer Go, Artif. Intell. 175 (11) (2011) 1856–1875.

[4] T. Pepels, M. H. M. Winands, Enhancements for Monte-Carlo Tree Search in Ms Pac-Man, in: Proc. IEEE Conf. Comput. Intell. Games, Granada, Spain, 2012, pp. 265–272.

[5] E. J. Powley, D. Whitehouse, P. I. Cowling, Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem, in: Proc. IEEE Conf. Comput. Intell. Games, Granada, Spain, 2012, pp. 234–241.

[6] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time Analysis of the Multi-armed Bandit Problem, Mach. Learn. 47 (2) (2002) 235–256.

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

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

[9] C.-S. Lee, M.-H. Wang, G. M. J.-B. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, T.-P. Hong, The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments, IEEE Trans. Comp. Intell. AI Games 1 (1) (2009) 73–89.

[10] B. Arneson, R. B. Hayward, P. Henderson, Monte Carlo Tree Search in Hex, IEEE Trans. Comp. Intell. AI Games 2 (4) (2010) 251–258.

[11] T. Keller, M. Helmert, Trial-based Heuristic Tree Search for Finite Horizon MDPs, in: Proc. Int. Conf. Automat. Plan. Sched., 2013, pp. 135–143.

[12] F. Maes, D. L. St-Pierre, D. Ernst, Monte Carlo Search Algorithm Discovery for Single-Player Games, IEEE Trans. Comp. Intell. AI Games 5 (3) (2013) 201–213.

[13] T. Cazenave, Nested Monte-Carlo Search, in: Proc. 21st Int. Joint Conf. Artif. Intell., Pasadena, California, 2009, pp. 456–461.

[14] A. Saffidine, Solving games and all that, Ph.D. thesis, Université Paris-Dauphine (2013).

[15] M. H. M. Winands, Y. Björnsson, J.-T. Saito, Monte-Carlo Tree Search Solver, in: Proc. Comput. and Games, LNCS 5131, Beijing, China, 2008, pp. 25–36.

[16] L. V. Allis, M. van der Meulen, H. J. van den Herik, Proof-Number Search, Artif. Intell. 66 (1) (1994) 91–124.

[17] D. E. Knuth, R. W. Moore, An analysis of alpha-beta pruning, Artif. Intell. 6 (4) (1975) 293–326.

[18] S. G. Akl, M. M. Newborn, The Principal Continuation and the Killer Heuristic, in: Proc. ACM Annu. Conf., 1977, pp. 466–473.

[19] J. Schaeffer, The History Heuristic and Alpha-Beta Search Enhancements in Practice, IEEE Trans. Pattern Anal. Mach. Intell. 11 (11) (1989) 1203–1212.

[20] R. E. Korf, Depth-first iterative-deepening: an optimal admissible tree search, Artif. Intell. 27 (1) (1985) 97–109.

[21] S. J. Pan, Q. Yang, A Survey on Transfer Learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.

[22] S. Thrun, Is Learning The n-th Thing Any Easier Than Learning The First?, in: Proc. Neur. Inform. Process. Sys., 1996, pp. 640–646.

[23] R. Caruana, Multitask Learning, Mach. Learn. 28 (1997) 41–75.

[24] R. Vilalta, Y. Drissi, A Perspective View and Survey of Meta-Learning, Artif. Intell. Rev. 18 (2002) 77–95.

[25] D. Tom, M. Müller, Computational Experiments with the RAVE Heuristic, in: Proc. Comput. and Games, LNCS 6515, Kanazawa, Japan, 2010, pp. 69–80.

[26] T. Kozelek, Methods of MCTS and the game Arimaa, M.S. thesis, Charles Univ., Prague (2009).

[27] E. J. Powley, D. Whitehouse, P. I. Cowling, Bandits all the way down: UCB1 as a simulation policy in Monte Carlo Tree Search, in: Proc. IEEE Conf. Comput. Intell. Games, Niagara Falls, Ontario, Canada, 2013, pp. 81–88.

[28] 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.

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

[30] P. I. Cowling, E. J. Powley, D. Whitehouse, Information Set Monte Carlo Tree Search, IEEE Trans. Comp. Intell. AI Games 4 (2) (2012) 120–143.

[31] D. Whitehouse, E. J. Powley, 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.

[32] I. Frank, D. Basin, Search in games with incomplete information: a case study using Bridge card play, Artif. Intell. 100 (1-2) (1998) 87–123.

[33] J. R. Long, N. R. Sturtevant, M. Buro, 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.

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

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

[36] P. Ciancarini, G. P. Favini, Monte Carlo tree search in Kriegspiel, Artif. Intell. 174 (11) (2010) 670–684.

[37] S. Edelkamp, T. Federholzner, P. Kissmann, Searching with Partial Belief States in General Games with Incomplete Information, in: Proc. KI: Adv. Artif. Intell., LNCS 7526, Saarbrücken, Germany, 2012, pp. 25–36.

[38] R. B. Myerson, Game Theory: Analysis of Conflict, Harvard University Press, 1997.

[39] C. A. Luckhardt, K. B. Irani, An Algorithmic Solution of N-Person Games, in: Proc. Assoc. Adv. Artif. Intell., Philadelphia, Pennsylvania, 1986, pp. 158–162.

[40] R. E. Korf, Multi-player alpha-beta pruning, Artif. Intell. 48 (1) (1991) 99–111.

[41] B. Brügmann, Monte Carlo Go, Tech. rep., Max-Planke-Inst. Phys., Munich (1993).

[42] P. D. Drake, S. Uurtamo, Heuristics in Monte Carlo Go, in: Proc. Int. Conf. Artif. Intell., Las Vegas, Nevada, 2007, pp. 171–175.

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

[44] Y. Björnsson, H. Finnsson, CadiaPlayer: A Simulation-Based General Game Player, IEEE Trans. Comp. Intell. AI Games 1 (1) (2009) 4–15.

[45] M. J. W. Tak, M. H. M. Winands, Y. Björnsson, N-Grams and the Last-Good-Reply Policy Applied in General Game Playing, IEEE Trans. Comp. Intell. AI Games 4 (2) (2012) 73–83.

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

[47] P. D. Drake, The Last-Good-Reply Policy for Monte-Carlo Go, ICGA Journal 32 (4) (2009) 221–227.

[48] H. Baier, P. D. Drake, The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go, IEEE Trans. Comp. Intell. AI Games 2 (4) (2010) 303–309.

[49] J. A. Stankiewicz, M. H. M. Winands, 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.

[50] E. J. Powley, D. Whitehouse, P. I. Cowling, Monte Carlo Tree Search with Macro-Actions and Heuristic Route Planning for the Multiobjective Physical Travelling Salesman Problem, in: Proc. IEEE Conf. Comput. Intell. Games, Niagara Falls, Ontario, Canada, 2013, pp. 73–80.

[51] D. P. Helmbold, A. Parker-Wood, All-Moves-As-First Heuristics in Monte-Carlo Go, in: Proc. Int. Conf. Artif. Intell., Las Vegas, Nevada, 2009, pp. 605–610.

[52] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, B. Bouzy, Progressive Strategies for Monte-Carlo Tree Search, New Math. Nat. Comput. 4 (3) (2008) 343–357.

[53] J. A. M. Nijssen, 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.

[54] A. Rimmel, F. Teytaud, Multiple Overlapping Tiles for Contextual Monte Carlo Tree Search, in: Proc. Applicat. Evol. Comput. 1, LNCS 6624, Torino. Italy, 2010, pp. 201–210.

[55] M. H. M. Winands, Y. Björnsson, J.-T. Saito, Monte Carlo Tree Search in Lines of Action, IEEE Trans. Comp. Intell. AI Games 2 (4) (2010) 239–250.

[56] Pagat, Card Games: Climbing Games, `http://www.pagat.com/climbing/`.

[57] P. I. Cowling, C. D. Ward, E. J. Powley, Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering, IEEE Trans. Comp. Intell. AI Games 4 (4) (2012) 241–257.

[58] I. Chernev, Combinations: The Heart of Chess, Dover Publications, 1967.

[59] Sensei's Library, Joseki, `http://senseis.xmp.net/?Joseki` (2012).

[60] Sensei's Library, Tesuji, `http://senseis.xmp.net/?Tesuji` (2012).

[61] A. Rimmel, O. Teytaud, C.-S. Lee, S.-J. Yen, M.-H. Wang, S.-R. Tsai, Current Frontiers in Computer Go, IEEE Trans. Comp. Intell. AI Games 2 (4) (2010) 229–238.

[62] Pagat, Dou Dizhu, `http://www.pagat.com/climbing/doudizhu.html`.

[63] E. J. Powley, D. Whitehouse, 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.

[64] M. L. Ginsberg, GIB: Imperfect Information in a Computationally Challenging Game, J. Artif. Intell. Res. 14 (2001) 303–358.

[65] Pagat, Hearts, `http://www.pagat.com/reverse/hearts.html`.

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

[67] N. R. Sturtevant, Multi-player games: algorithms and approaches, Ph.D. thesis, Univ. California Los Angeles (2003).

[68] N. R. Sturtevant, A. M. White, Feature Construction for Reinforcement Learning in Hearts, in: Proc. Comput. and Games, 2006, pp. 122–134.

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

[70] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, S. Sutphen, Checkers Is Solved, Science 317 (2007) 1518–1522.

[71] M. Buro, From Simple Features to Sophisticated Evaluation Functions, in: Proc. Comput. and Games, LNCS 1558, Tsukuba, Japan, 1998, pp. 126–145.

[72] D. Robles, P. Rohlfshagen, S. M. Lucas, Learning Non-Random Moves for Playing Othello: Improving Monte Carlo Tree Search, in: Proc. IEEE Conf. Comput. Intell. Games, Seoul, South Korea, 2011, pp. 305–312.

[73] P. Hingston, M. Masek, Experiments with Monte Carlo Othello, in: Proc. IEEE Congr. Evol. Comput., Singapore, 2007, pp. 4059–4064.

[74] J. A. M. Nijssen, Playing Othello Using Monte Carlo, B.S. Thesis, Maastricht Univ., Netherlands (2007).

[75] G. Tesauro, Temporal Difference Learning and TD-Gammon, Commun. ACM 38 (3) (1995) 58–68.

[76] F. van Lishout, G. M. J.-B. Chaslot, J. W. H. M. Uiterwijk, Monte-Carlo Tree Search in Backgammon, in: Proc. Comput. Games Workshop, Amsterdam, Netherlands, 2007, pp. 175–184.

[77] C. J. Clopper, E. S. Pearson, The use of confidence or fiducial limits illustrated in the case of the binomial, Biometrika 26 (4) (1934) 404–413.

[78] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, Parallel Monte-Carlo Tree Search, in: Proc. Comput. and Games, LNCS 5131, Beijing, China, 2008, pp. 60–71.