Sample Evaluation for Action Selection in Monte Carlo Tree Search

Dirk Brand
Computer Science Division
Stellenbosch University
7602 Matieland
South Africa
dirkbrand@ml.sun.ac.za

Steve Kroon
Computer Science Division
Stellenbosch University
7602 Matieland
South Africa
kroon@sun.ac.za

ABSTRACT

Building sophisticated computer players for games has been of interest since the advent of artificial intelligence research. Monte Carlo tree search (MCTS) techniques have led to recent advances in the performance of computer players in a variety of games. Without any refinements, the commonly-used upper confidence bounds applied to trees (UCT) selection policy for MCTS performs poorly on games with high branching factors, because an inordinate amount of time is spent performing simulations from each sibling of a node before that node can be further investigated. Move-ordering heuristics are usually proposed to address this issue, but when the branching factor is large, it can be costly to order candidate actions. We propose a technique combining sampling from the action space with a naïve evaluation function for identifying nodes to add to the tree when using MCTS in cases where the branching factor is large. The approach is evaluated on a restricted version of the board game Risk with promising results.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*

General Terms

Algorithms, Experimentation

Keywords

Monte-Carlo Tree Search, Risk, Evaluation Function, Sampling, Heuristics

SAICSIT2014, September 29-October 1 2014, Centurion, South Africa
Copyright 2014 ACM 978-1-4503-3246-0/14/09 ... $15.00
http://dx.doi.org/10.1145/2664591.2664612.

1. INTRODUCTION

Since the advent of artificial intelligence research, developing computer players for various games has received a large amount of attention. This can be partially attributed to the fact that a wide variety of games provide suitable abstractions of real-world tasks. Conceptually, the simplest games are deterministic two-player games with perfect information, such as Checkers, Chess, and Go. While the best action in most such games can theoretically be found by minimax game-tree search [33], in practice high branching factors make exact computation infeasible, so that the development of various search refinements and heuristics have been necessary to achieve performance comparable to humans.

A large improvement in the quality of Computer Go players can be attributed to the development of Monte Carlo tree search (MCTS) techniques in 2006 [22]. Subsequently, various refinements and heuristics have made considerable further improvements to the performance of Computer Go players making use of MCTS [13]. An important class of refinements are those modifying the exploration-exploitation tradeoff central to MCTS to de-emphasize exploration of new actions when branching factors are prohibitively large. Without such refinements, an inordinate amount of time is spent performing simulations from each sibling of a node before that node can be further investigated.

This work proposes such a technique using sampling from the action space in conjunction with a naïve evaluation function for identifying nodes to add to the tree during MCTS when the branching factor is large. The approach is evaluated on a restricted version of the board game Risk. The results indicate that the sampling approach can improve performance over classical approaches and MCTS implementations without this enhancement.

2. BACKGROUND AND RELATED WORK

This section highlights important aspects of MCTS and presents the restricted version of Risk we considered. Various related studies are then identified to place our work in context.

2.1 Basics of MCTS

MCTS is an any-time best-first tree search algorithm which uses stochastic simulations to both guide the growth of the search tree as well as evaluate its nodes. The process followed in MCTS is illustrated in Figure 1 and consists of iterations of the following phases, starting from a single root node, until the available time expires:

- **Selection** — Recursively selecting a best child node to explore, until a node to expand is identified.[1] Leaf nodes are always expanded if reached; internal nodes may be expanded if there are legal actions from the node not yet represented in the current search tree. The mechanism for identifying the best child is known as the tree policy.
- **Expansion** — Adding a new node to the search tree as child of the node found in the previous step. This node corresponds to a new legal action to be explored.
- **Simulation** — Performing a simulation from the newly expanded node. A simulation (also known as a playout or rollout) typically involves playing the rest of the game using some simple random strategy. This strategy is known as the simulation policy.
- **Backpropagation** — Updating nodes in the search tree with information obtained from the simulation. Most commonly, nodes maintain information on the success rate of simulations performed starting from their descendants; however, various other information may also be kept.

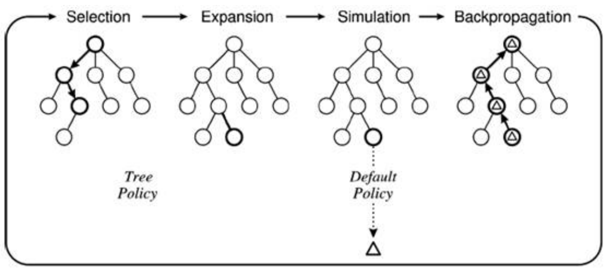

Figure 1: The MCTS Algorithm (Figure from [5]).

When the search stops, the actions that lead from the root are considered and the best-performing action is returned. Coulom [13] considered various options for identifying the best-performing action — based on his work, it is now fairly common to consider the action that has been visited the most as the best. We use this "most robust child" convention in the rest of this work.

2.1.1 *Tree policies for large branching factors*

The tree policy determines which nodes are selected in the selection phase of the MCTS algorithm. Generally, MCTS considers each node in the search tree as a multi-armed bandit (where each arm corresponds to a possible next action), and a tree policy applies some strategy for the multi-armed bandit problem at the node. One of the first MCTS algorithms, upper confidence bounds applied to trees (UCT), applied UCB1 [2], a classical multi-armed bandit algorithm, at each node [22]. The authors proved convergence of this algorithm to optimal behaviour, despite the fact that nodes do not behave exactly like multi-armed bandits in tree search due to nonstationarity of the reward distribution, as discussed in [13].

The UCT tree policy identifies a node for expansion if there are legal actions from the corresponding state not yet in the search tree; once all such legal actions have been added to the search tree as children for a node, the policy selects the child node with the highest UCT value (also known as urgency). The UCT value of a node is calculated as

$$\mathrm{UCT}(v) = \frac{Q(v)}{N(v)} + c\sqrt{\frac{\ln(N(p_v))}{N(v)}} \tag{1}$$

where $N(v)$ is the number of simulations from descendants of a node v, $Q(v)$ is the number of those simulations won, p_v denotes the parent node of v, and c is a tunable *exploration constant*. Thus, the first term estimates the probability of winning from v, while the second term represents uncertainty in the estimate based on the sample size. In this setting, a higher value for c will result in exploration being emphasised relative to exploitation. An appropriate value for c must typically be found experimentally — the experiments in this work used $c = 1.2$ (found experimentally [17]).

A practical problem with UCT is that it requires every legal action from a node to be added to the search tree before any action from that node is evaluated for a second time. This requirement is inconvenient when the game tree branching factor is large, so some heuristics bypass this behaviour. The best-known heuristics in this regard are first-play urgency (FPU) [16] and progressive widening [6, 12].

First-play urgency assumes that if some actions that have already been considered look promising enough, they can be explored further before trying other unexplored actions. This is controlled by means of a constant called the FPU value: if certain children's UCT values exceed the FPU value, further exploitation of those children is permitted. To ensure that all nodes are eventually considered in the limit, the FPU value is typically set to a value larger than one — a common value in practice (as used in Oakfoam [32]), and which we use in this work, is 1.1 [23]. Using this technique allows good-looking actions found early in positions to be investigated more thoroughly, while not hampering exploration when initial actions look poor.

Progressive widening avoids early expansion in UCT by specifying a schedule for adding unexplored actions to nodes: after a certain number of MCTS iterations descend through the node, another child becomes eligible to be added to the tree. Usually these schedules restrict the number of children to grow as the logarithm or some root of the number of simulations — examples of each case can be found in [12] and [27] respectively.

Unlike FPU, this approach could be very poor if legal actions for expansion are selected entirely randomly: even if initial actions look poor, the schedule prevents further exploration. For this reason, progressive widening orders the legal actions based on some quality heuristic [27] (such as an evaluation function), and expands them in decreasing order of the heuristic.

This ordering approach is often also applied to FPU to improve its performance. Common approaches which facilitate this, by modifying the UCT value formula, are progressive bias [6] and rapid action value estimation (RAVE) [15] — however, further discussion of these techniques is beyond the scope of this paper.

[1]We omit minor technical details involving the case where a node under consideration is a terminal node in the game tree.

2.1.2 *MCTS for stochastic domains*

The version of Risk we consider in our investigations is a perfect information game, but nevertheless certain outcomes are determined by chance. Various studies have considered approaches to dealing with stochasticity and uncertain information in MCTS [5, 3, 31, 8, 24].

We follow a fairly straightforward extension of MCTS for dealing with this stochasticity based on the approach originally proposed for backgammon in [31].

The approach modifies MCTS as follows: in the tree, actions with stochastic outcomes correspond to nodes, while the outcomes correspond to separate nodes. Stochastic action nodes are never selected for expansion during the selection step; instead, when the action node is added to the tree in an expansion step, all of the nodes corresponding to that action's outcomes are also added to the tree. Furthermore, in the selection process, the tree policy is ignored at nodes with stochastic outcomes — instead, an outcome is selected according to its probability of occurring.

This approach has drawbacks when there are many stochastic outcomes for some actions. However, in the case we considered, no node has more than 3 possible outcomes. When an attack node is expanded, all the possible outcomes are added to the tree, but are subsequently explored proportionally to their respective probabilities of occurring.

2.1.3 *Related work*

It generally seems that some kind of prior knowledge about candidate actions is needed in order to outperform the simple UCT algorithm combined only with vanilla FPU. When there are too many actions at a node, however, it is not feasible to obtain an explicit heuristic ordering on all these actions.

The work we discuss later presents an approach that uses sampling to avoid evaluating all candidate actions. Many implementations avoid this concern by abstracting the state and action spaces so that the branching factor is feasible for established approaches. We are not aware of previous studies directly addressing this problem in this context; however, related work has been done on similar problems.

Wang et al. [34] present algorithms for regret minimization of multi-armed bandits with infinitely many arms. Their work also applies to cases where the number of actions exceed the number of simulations that can be performed. Presumably this approach could be incorporated into a tree search, but we are not aware of a convergence result for such an approach like the one for UCB1 in [22]. Couëtoux et al. [10] point out a theoretical convergence issue with applying progressive widening to continuous (and hence infinite) state and action spaces with stochastic transitions, and suggest double progressive widening to address it. Couëtoux and Doghmen [9] give empirical evidence that MCTS using double progressive widening can outperform regular progressive widening on appropriate domains. The application of RAVE to continuous state and action spaces has also been considered: Couëtoux et al. [11] effectively apply Gaussian kernel density estimation to obtain interpolated values for applying RAVE.

Churchill et al. [7] propose scripted action ordering in the context of minimax tree search with alpha-beta pruning for real-time strategy games. This approach leverages scripts representing simple domain knowledge: actions predicted by the scripts are investigated first to enhance subsequent pruning in the rest of the search. One can envision similar use of scripts and similar heuristics for identifying initial nodes to expand when employing progressive widening.

Gibson et al. [18] investigated the use of MCTS and UCT for drafting territories in the setup phase in the game of Risk. They extracted candidate tactical features for guiding territory drafting, and used supervised learning to establish which combinations of features were important for initial territory drafting. They showed that the drafting strategy can have a notable effect on the strength of computer Risk players, and incorporated their techniques into existing computer players for a Risk variant called Lux Delux [29]. Another Risk variant for which computer players have been developed is Domination [37].

Wolf [36] used learning techniques in the design of his AI players for Risk. He developed an evaluation function for evaluating the game state. The evaluation function, coupled with a game tree search approach, formed the basis for his AI players. Tan [30] applied Markov chain theory to model the outcome of Risk battles, but made some erroneous independence assumptions. Osborne [26] addressed these errors, and provides a table predicting battle outcomes which we make use of in this work.

Other examples of enhancing MCTS with an evaluation function, include Coulom [13] who introduced a technique for mixing regular MCTS backup operators with minimax evaluations. Similar techniques have been shown to significantly improve [24] the quality of MCTS players in games like Breakout and LOA [35]. Lanctot et al. [24], introduced adding evaluation function values to nodes in the MCTS tree. These values are updated during the backup phase, by considering the minimax values of a node's children. This effectively mixes minimax-style leaf evaluations with Monte-Carlo simulation results.

2.2 Risk

Risk is a modern strategy board game for 2-6 players invented by French film director Albert Lamorisse in 1957. Players vie for global domination by recruiting and deploying troops (or armies) to various territories, and then using these armies to battle one another. The goal of a player is to eliminate all other players from the board (or, alternatively, occupy every territory on the board).

The game inherently includes stochasticity from dice rolls determining the outcomes of battles, as well as cards drawn from a deck during the game. There is also imperfect information, since the cards drawn remain hidden for some time. In this study, we consider a simplified form of Risk obtained by: (a) restricting the game to two players; (b) removing the deck of cards; (c) simplifying the initial setup of the game as detailed below; and (d) constraining the number of troops used to attack or defend in a battle (both players must always attack and defend with the maximum permitted number of armies)[2]. This eliminates the imperfect information component, as well as opponent modelling considerations required for dealing sensibly with many players. (For the complete rules of Risk, see [1]).

[2]In the original version of Risk, the attacking player could choose to attack with any number of armies between one and three. [36] showed that a player that attacks or defends with the maximum number of armies possible in an encounter, has the highest probability of victory in that encounter.

Risk has a huge branching factor [36]. As an example, during the recruitment phase, newly recruited troops are allocated to the territories owned by a player. Thus, each possible allocation of these troops to the territories owned corresponds to a legal action. This number of possible troop allocations is $\binom{n+m-1}{n}$, where n is the number of troops to place and m the number of territories that the player owns. If, for instance, a player owns 21 territories and recruits 7 troops (as is typical at the start of a two-player Risk game), there are 880070 different legal actions. Note that our simplified Risk version still has states with such large branching factors.

In our restricted Risk domain, game play begins with an initial setup phase, after which players take turns. A player's turn consists of three distinct game phases, namely:

1. the troop recruitment phase,
2. the attack phase, and
3. the troop manoeuvre phase.

As in the original game, the territories are initially randomly, but equally, divided between the players. Thereafter, each player receives a pool of sixty armies to allocate to those territories. They then proceed to place these armies simultaneously on their respective territories. This approach is simpler than the original game: the original game provisioned forty armies to each player and another forty to a 'neutral' player, and players would place their troops in a turn-based fashion. We simplified this, because the initial placement of troops was not of interest for the general strategies of players. In our engine, after the territories are allocated, one player gets randomly selected to place their troops first, then the other player places their troops.

In the attack phase, the player has the choice of where they want to initiate an attack from. The player then chooses a source and a neighbouring destination territory of the attack. A number of dice are then rolled (between one and three for the attacking player and one and two for the defending player, depending on the number of troops they are attacking/defending with) and the attack is resolved. If the attack leads to the player conquering a territory, the player may move as many troops as he wishes from the attacking territory to the conquered territory, subject to at least one troop remaining on the attacking territory.

Finally, the player can manoeuvre some troops by selecting a source and destination territory, as well as a number of troops to move from the source to the destination.[3]

The game ends when one of the players has successfully conquered every territory on the map.

3. PROPOSED APPROACH

We propose leveraging a crude evaluation function via sampling to guide the expansion phase of MCTS when the branching factor is so large that normal approaches are not feasible. This allows us to bypass issues with large action spaces by identifying a promising node to expand based on its estimated value. Our approach is proposed for situations where a significant portion of the MCTS running time is consumed by evaluating and ordering the possible actions from nodes. In such cases, we propose sampling a number K of unexplored legal actions, evaluating the resulting state, and expanding the action with the best evaluation. To obtain good performance, it is desirable that sampling from the action space is uniform, i.e. all possible actions are equally likely to be selected.

[3]For manoeuvring, it is required that the source and destination territories be connected to each other by a chain of territories owned by the manoeuvring player.

3.1 Implementation

We implemented an open-source MCTS-based computer player for our restricted version of Risk. We achieve uniform sampling [4] for all the phases in a player's turn in our Risk domain as follows. The attack and manouevre actions have somewhat reasonable branching factors, so it is reasonable to explicitly enumerate all possible attack actions, as well as the possible source-destination pairs for all manoeuvre actions. Attack actions can thus easily be sampled uniformly, while manoeuvre actions can be sampled uniformly using a binary search on an array constructed from the possible pairs and the number of units on the various territories. Moving after an attack is not dealt with as an MCTS action, but instead by considering each possible number of troops to move and selecting the number leading to the game state with the highest value according to the evaluation function. For recruitment, a scheme based on randomized partitioning was used to sample actions. n troops (tokens) must be placed into m territories (buckets), so that randomly generating such a partition effectively ensures a uniform random troop recruitment.

To ensure all actions can potentially be added to the tree, and to reduce inefficiency from redundant sampling when the number of remaining unexplored actions u becomes similar to K, the number of samples considered for expansion is set to $\max\{u/2, 1\}$ when $u/2 < K$.

When an action is identified as the best of a number of samples in the attack or manoeuvre phases, it is checked against the actions already in the tree, to ensure that duplicate actions are not added. Uniqueness is not checked during the recruitment phase, since selecting duplicates becomes very unlikely when the branching factor becomes sufficiently large. In this case, there are thus potentially multiple child nodes for the same recruitment action, and this may lead to not all recruitment actions being added to the tree. However, these effects should only play a noticeable role when a player has very few remaining territories, in which case the player has already effectively lost the game. As such, we believe this effect should not have a significant impact on our results.

3.1.1 Simulation policy

The simulation policy is a random playing strategy, enhanced with some heuristics to simplify decision making and to increase the speed of individual simulations. The heuristics are hand-crafted and essentially capture the core Risk strategy of one of the authors:

- *Recruitment phase*: all recruited troops are placed on a randomly selected frontier territory.[4]
- *Attack phase*: the territory used during recruitment is chosen as the attacking territory. The neighbouring enemy territories are then traversed (in a fixed order) and the first territory (if any) that can be conquered

[4]A frontier territory is one adjacent to an enemy territory.

by the attacking territory with a probability higher than 0.5 is attacked once. (The success probability of conquering a territory is obtained from the table in [26]). This is repeated until a territory is conquered (in which case a random number of troops is moved to the conquered territory, the conquered territory becomes the new active territory, and attacking continues from there) or there are no more such territories.

- *Manoeuvre phase*: troops are shifted from the territory with the most troops, to a random legal destination territory with the least troops. The number of troops is chosen such that the two territories have an equal number of troops after the manoeuvre (in the case of a tie, the source territory gets an additional troop). If the territory with the most troops is isolated, no manoeuvre is performed.

3.1.2 Evaluation function construction

For this work, we constructed a heuristic evaluation function for the Risk domain which is a linear function of the thirteen features designed and tested by Wolf [36]. More details on these features are in Appendix C. To determine the feature weights, we made use of confidence local optimisation (CLOP)[14], a noisy black-box parameter-tuning algorithm.

Training weights with CLOP was done by constructing a "Greedy AI": a simple one-ply game tree search using the evaluation function and best-of-50 sampling to determine the set of leaf nodes stemming from the root. It was limited to a maximum of 10 branches. This greedy player was ideal for weight training, as it makes decisions quickly, and relies entirely on the evaluation function for its selection. The training consisted of repeatedly playing games between this greedy AI and a baseline heuristic AI (detailed in Section 3.2.2). CLOP used the results of these games to optimise the feature weights for better general game performance.

3.2 Other agents

We implemented various other computer players to compare our proposed sampling-based MCTS agent ["MCTS(20)"] to. These were:

- a player using the MCTS simulation policy as a strategy ["Simulation Player"];
- a baseline player ["Baseline"], following an aggressive variation of the simulation policy;
- a player based on expectiminimax (EMM) search with a restricted branching factor ["EMM"];
- an MCTS player with no sampling or evaluation function (i.e. the player essentially expands random actions) ["MCTS(1)"];
- an MCTS player that first expands the node suggested by the baseline strategy and thereafter uses the proposed sampling method ["MCTS Baseline"].
- an MCTS player that samples 100 actions instead of the usual 20 ["MCTS(100)"].

Below we briefly outline EMM search and the approach used in our baseline player. The setup phase for all players is done in the same way (since it was not the focus of the investigation). During the setup phase, troops are placed on territories, in round-robin fashion, until every territory has either two or three troops.

3.2.1 EMM player

The EMM algorithm is a recursive depth-first search algorithm that builds a game tree to a predetermined depth, evaluating leaf nodes using an evaluation function, and propagating leaf values up the tree [25, 28]. It is an extension of the minimax tree search algorithm using *stochastic nodes* to model stochastic events. The value of a stochastic node is determined by calculating the mean value of its child nodes, thus making use of the probability of observing each stochastic outcome.

The EMM player in this work was restricted to examine only ten children per deterministic node, with these children identified using best-of-K sampling based on our evaluation function. This approach allowed a deeper tree to be built in the same time frame that the MCTS player received. The strategy of the EMM player for selecting the number of troops to move to a conquered territory was identical to that of the MCTS players.

3.2.2 Baseline player

The baseline player implements an aggressive version of the MCTS simulation policy as a playing strategy. The modification to the strategy is made to increase the playing strength of the player, while sacrificing some of the simulation speed.

During the recruitment phase, all recruited troops are placed on a frontier territory in order to maximize the resulting ratio of troops on the territory to the sum of opponent's troops on adjacent territories. (For example, if after placement there are three troops on the territory and the opponent has four neighbouring territories each with two troops, the ratio is $\frac{3}{2\times 4} = \frac{3}{8} = 0.375$). In the case of ties, one of the tied territories is chosen randomly.

In the attack phase, the player repeatedly attacks from its *active territory* to a (randomly selected) most vulnerable adjacent opponent territory (i.e. an adjacent opponent territory with the least number of troops) until further attack from the active territory is no longer possible. Initially, the active territory is the territory on which the player placed his recruited troops. If an opponent territory is defeated, all but one of the troops on the active territory are shifted to the defeated territory, which then becomes the active territory. When moving troops after successfully conquering an enemy territory, the baseline player moves all available troops from the attacking territory to the conquered territory.

The manoeuvre phase is handled exactly the same as in the simulation policy.

4. EXPERIMENTS AND RESULTS

We measured the relative playing strength of various computer players by using game outcomes to determine rating intervals for each player. The rating system used is the Glicko rating system developed by Glickman [19, 20]. This system is more desirable than the classical Elo rating system (which it generalizes), since each player x is assigned not only a rating μ_x, but also a rating deviation σ_x, which corresponds to the standard deviation of the estimated player rank. As with the Elo rating system, a difference of 100 rating points corresponds to an estimated 64% probability of winning for the higher rated player. To measure and keep track of the ratings, we used a Java library for Glicko-2 [21], a further generalization of the Glicko rating system. The Glicko system also caters for changes in rating deviation over time, but since computer players do not change in strength over time, we disabled this by setting each player's rating volatility as well as the Glicko-2 rating system constant τ to zero.

A game manager was implemented to schedule games between the players with the aim of identifying statistically significant differences in performance. Two pools of players were created, corresponding to two experiments. The first experiment compared the performance of MCTS(K) with other, (mostly) non-MCTS agents, while the second investigated the effect of K on the performance of MCTS(K). In each pool, players were given an initial Glicko rating of 1500 and a rating deviation of 350 (the default for an unranked player), and rating values were updated in the pool after each game was completed. The scheduler initially performed random pairings of players in each pool, to provide an initial indication of ratings. After that, games were typically scheduled by identifying pairs of players in the pool for which the evidence of a difference between the ratings was weak (to efficiently obtain statistical significance). This evidence for a pair of players x and y was quantified by calculating the standard score of the difference between their ratings:

$$Z_{xy} = \frac{|\mu_x - \mu_y|}{\sqrt{\sigma_x^2 + \sigma_y^2}} \tag{2}$$

This helped to identify matches that would be the most informative, as the results would potentially help significantly distinguish players. Note that the ratings obtained in different experiments are not directly comparable. In particular, there is no reason to expect a similar rating for the same agent in both experiments.[5]

For all matches, the starting player was randomly determined. The EMM agent uses best-of-50 sampling where applicable, and a five-ply search to select actions, which typically took a few seconds. The MCTS agents generated around 1000 playouts per second, and were given 2.5 seconds of computation time to choose each action. In order to get enough game results, we stopped games once one of the players had a ten territory advantage over the other player, subject to at least 100 actions having been performed.

4.1 Results

The final player ratings and rating deviations of the players in the first experiment are presented in Table 1. Figure 2 presents 95% confidence intervals for the various ratings, and Table 2 shows the P-values for detecting significant differences between players in this experiment.

We see that the best performing player is, surprisingly, the baseline player. At first, it seems the observed performance of the baseline strategy over our proposed MCTS strategies could possibly be explained by the fact that the baseline strategy's actions are good enough that even the best-of-K sampling strategy would be unlikely to sample actions better than the baseline strategy. Another possible explanation could be that the evaluation function consistently undervalues the results of actions generated by the baseline strategy. However, our MCTS approach augmented with the baseline action as the first candidate action is also significantly outperformed by the baseline strategy, indicating that both of these explanations do not adequately explain the observation. Instead, a likely explanation is that the difference in performance is because the baseline strategy moves all available troops into conquered territories, while all the MCTS approaches select the number of troops to move by opti-

[5]Although each experiment is reported on individually, the results of all the games were pooled to obtain a single rating and rating deviation for each agent once both experiments were completed. This makes it possible to compare the ratings of agents not involved in the same experiment to some extent — these results are presented in Appendix B.

Table 1: Ratings and rating deviations of our proposed approach and other agents.

	Rating	Rating Deviation
Baseline	1669.23	32.71
MCTS Baseline	1595.03	30.23
MCTS(100)	1421.14	40.56
MCTS(20)	1333.77	28.46
MCTS(1)	1319.97	39.33
Simulation Player	1270.66	27.29
EMM	1157.42	43.48

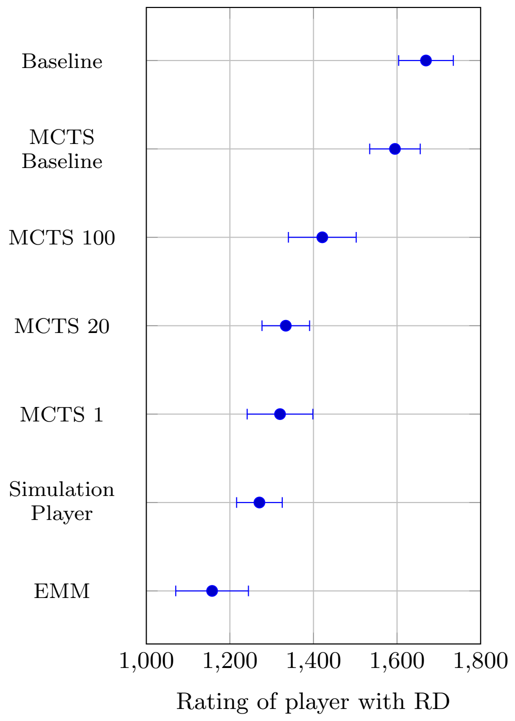

Figure 2: 95% confidence intervals for the ratings in Table 1. These are calculated as twice the ratings deviation on either side of the estimated rating, as recommended in [19].

Table 2: P-values for testing whether the player in the column is stronger than the player in the row (p-values, to 4 significant digits).

	Baseline	MCTS Baseline	MCTS(100)	MCTS(20)	MCTS(1)	Simulation Player
Baseline	-	-	-	-	-	-
MCTS Baseline	0.0485	-	-	-	-	-
MCTS(100)	$< 10^{-4}$	0.0003	-	-	-	-
MCTS(20)	$< 10^{-4}$	$< 10^{-4}$	0.0392	-	-	-
MCTS(1)	$< 10^{-4}$	$< 10^{-4}$	0.0367	0.3897		-
Simulation Player	$< 10^{-4}$	$< 10^{-4}$	0.0011	0.0548	0.1515	-
EMM	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	0.0003	0.0028	0.0139

mizing with respect to the evaluation function[6]. Thus the situation for the MCTS agent after selecting the baseline action is different to the situation for the baseline player. Another possibility, however, is that the poorer performance of MCTS augmented with the baseline action relative to the baseline action indicates that the MCTS search, which is guided by the simulation results, tends to favour the non-baseline actions in the tree. This seems unlikely because of the high similarity between the simulation policy and the baseline player.

Despite being outperformed by the baseline player, we see that using a naïve evaluation function with sampling to guide MCTS provides an improvement over the MCTS simulation strategy, naïve MCTS (i.e. MCTS(1)), as well as the EMM agent. Furthermore, we see that MCTS(100) significantly outperforms MCTS(20). A more thorough investigation of the effect of K on the performance was conducted in our second experiment. The results are presented in Table 3. We see that increasing the sample size improved the performance of the player (given the same time constraints). There is evidence of diminishing returns as K increases, however, so we expect that for larger K performance will plateau before beginning to degrade once the sampling overhead becomes too large: in general, it seems the optimal value of K should depend on the branching factor. In this experiment, the strength increases between successive values of K are generally not statistically significant. However, the trend is quite visible, and the differences between players with small K and those with large K are statistically significant. As far as we are aware, these are the first results reported in the literature illustrating and quantifying improvements in the performance of MCTS due to selective expansion of nodes.

[6]Since the submission of this paper, all the agents have been modified to employ MCTS-style search for selecting how many troops to move after conquering a territory. With this approach, initial experiments indicate no significant difference between the baseline and MCTS Baseline players, supporting this hypothesis.

Table 3: Performance comparison of various choices of K for our proposed sampling approach.

K	Rating	Rating Deviation
1	1393.98	95.42
2	1357.79	86.14
5	1483.57	76.66
10	1556.21	68.51
20	1582.86	81.48
50	1605.82	72.92
100	1621.76	73.16

5. CONCLUSIONS AND FUTURE WORK

This study presents a sampling-based approach to using an evaluation function for guiding MCTS during the expansion phase in domains with very large branching factors. Experiments in Risk show improved results over naïve MCTS, despite a hand-crafted strategy outperforming the proposed approach.

It would be valuable to investigate the performance of our approach with a better-tuned evaluation function (or in another domain with a good evaluation function): a weak evaluation function may well consistently rate good actions poorly and vice versa so that all the expanded actions in the MCTS are poor. Another avenue to investigate in this regard is not always selecting the sample with highest evaluation, but select samples stochastically based on the various samples' evaluations (for example, by using Boltzmann sampling).

6. ACKNOWLEDGEMENTS

The authors would like to thank Marc Lanctot for his feedback on a draft of this paper. Some of this work was done at the MIH Media Lab at Stellenbosch University.

7. REFERENCES

[1] Risk rule book. `http://www.hasbro.com/common/instruct/risk.pdf`, 1997. [Online; accessed 15-October-2013].

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[3] R. Bjarnason, A. Fern, and P. Tadepalli. Lower bounding Klondike Solitaire with Monte-Carlo planning. In *ICAPS*, 2009.

[4] D. Brand. Risk. `https://github.com/DirkBrand/Risk/releases/tag/v1.1`, 2013.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] G. M. J. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 04(03):343–357, 2008.

[7] D. Churchill, A. Saffidine, and M. Buro. Fast heuristic search for RTS game combat scenarios. In *AIIDE*, 2012.

[8] P. Ciancarini and G. P. Favini. Monte Carlo tree search in Kriegspiel. *Artificial Intelligence*, 174(11):670–684, 2010.

[9] A. Couëtoux and H. Doghmen. Adding double progressive widening to upper confidence trees to cope with uncertainty in planning problems. In *The 9th European Workshop on Reinforcement Learning (EWRL-9)*, 2011.

[10] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In *Learning and Intelligent Optimization*, pages 433–445. Springer, 2011.

[11] A. Couëtoux, M. Milone, M. Brendel, H. Doghmen, M. Sebag, and O. Teytaud. Continuous Rapid Action Value Estimates. In *The 3rd Asian Conference on Machine Learning (ACML2011)*, volume 20, pages 19–31, 2011.

[12] R. Coulom. Computing Elo ratings of move patterns in the game of Go. In *Computer Games Workshop*, 2007.

[13] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, pages 72–83. Springer, 2007.

[14] R. Coulom. CLOP: Confident local optimization for noisy black-box parameter tuning. In *Advances in Computer Games*, pages 146–157. Springer, 2012.

[15] S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning*, pages 273–280. ACM, 2007.

[16] S. Gelly and Y. Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop (December 2006)*, 2006.

[17] S. Gelly, Y. Wang, R. Munos, and O. Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical report, HAL-INRIA, 2006.

[18] R. G. Gibson, N. Desai, and R. Zhao. An automated technique for drafting territories in the board game Risk. In *AIIDE*, 2010.

[19] M. Glickman. Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics*, 48:377–394, 1999.

[20] M. Glickman. Glicko rating system, 2013. [Online; accessed 28-March-2013].

[21] J. Gooch. Java implementation of the Glicko-2 rating algorithm. `https://github.com/goochjs/glicko2`, 2010. [Online; accessed 17-October-2013].

[22] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[23] T. Kozelek. Methods of MCTS and the game Arimaa. Master's thesis, Faculty of Mathematics and Physics, Charles University, Prague, 2009.

[24] M. Lanctot, M. H. Winands, T. Pepels, and N. R. Sturtevant. Monte Carlo tree search with heuristic evaluations using implicit minimax backups. arXiv preprint arXiv:1406.0486, 2014.

[25] D. Michie. Game-playing and game-learning automata. *Advances in programming and non-numerical computation*, pages 183–200, 1966.

[26] J. A. Osborne. Markov chains for the Risk board game revisited. *Mathematics magazine*, pages 129–135, 2003.

[27] P. Rolet, M. Sebag, and O. Teytaud. Boosting active learning to optimality: a tractable Monte-Carlo, Billiard-based algorithm. In *Machine Learning and Knowledge Discovery in Databases*, pages 302–317. Springer, 2009.

[28] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 3rd edition, 2009.

[29] Sillysoft Games. Lux delux, 2013. [Online; accessed 20-June-2014].

[30] B. Tan. Markov chains and the RISK board game. *Mathematics Magazine*, 70:349–357, 1997.

[31] F. Van Lishout, G. Chaslot, and J. W. Uiterwijk. Monte-Carlo tree search in Backgammon. In *Computer Games Workshop*, 2007.

[32] F. van Niekerk. Oakfoam - Computer player for the game of Go. `http://oakfoam.com/`, 2012. [Online; accessed 18-March-2014].

[33] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[34] Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *NIPS*, volume 8, pages 1–8, 2008.

[35] M. H. M. Winands, Y. Bjornsson, and J.-T. Saito. Monte Carlo tree search in lines of action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):239–250, 2010.

[36] M. Wolf. *An Intelligent Artificial Player for the Game of Risk.* PhD thesis, Darmstadt University of Technology, 2005.

[37] *yuranet*. Domination (Risk board game). [Online; accessed 27-Mar-2013].

APPENDIX

A. SOURCE CODE

All source code that was used in this work is part of the open-source Risk framework [4]. Version 1.1 was used for the work in this paper and is tagged in the code repository. All default parameters were used.

B. POOLED EXPERIMENTAL RESULTS

Table 4 summarizes the results of the pooled experiments.

C. FEATURES

All the features are scaled to be between 0 and 1. This was not explicitly done in Wolf's work [36], but was deemed necessary for training the weights and for subsequent experiments.

1. **Armies**. The percentage of the total number of armies that the current player owns.
2. **Best Enemy**. The relative strength of the best enemy player. Since our version of Risk only allows for two-player games, this would just return the strength of the single opponent.

Table 4: Summary of game results and ratings of the various players, after pooling results from both experiments. The first two columns indicate the player rating and rating deviation, while the remaining columns show the numbers of wins and total games played per player pair. Here a cell's entry shows the number of wins of the row's player against the column's player, as well as the total number of games played between the pair. Entries of 0/0 occur when players were not in the same initial experiment. The last column shows total number of wins and games played by the player in each row.

	Rating	RD	EMM	MCTS(2)	Simulation AI	MCTS(5)	MCTS(1)	MCTS(20)	MCTS(50)	MCTS(10)	MCTS(100)	MCTS Baseline	Totals
EMM	1162.81	45.39	-	-	-	-	-	-	-	-	-	-	20/118
MCTS(2)	1213.32	99.29	0/0	-	-	-	-	-	-	-	-	-	6/19
Simulation AI	1305.03	25.99	7/21	0/0	-	-	-	-	-	-	-	-	119/305
MCTS(5)	1305.03	86.18	0/0	2/3	0/0	-	-	-	-	-	-	-	10/22
MCTS(1)	1334.42	37.60	12/16	0/2	10/18	1/3	-	-	-	-	-	-	49/135
MCTS(20)	1359.52	26.42	13/13	1/2	36/123	1/1	16/25	-	-	-	-	-	105/282
MCTS(50)	1382.54	75.40	0/0	2/3	0/0	4/5	2/3	2/3	-	-	-	-	14/27
MCTS(10)	1403.29	73.58	0/0	5/6	0/0	4/7	1/3	4/6	3/4	-	-	-	18/30
MCTS(100)	1446.75	35.23	28/28	3/3	6/18	1/3	14/22	9/22	6/9	3/4	-	-	86/147
MCTS Baseline	1646.28	28.14	17/18	0/0	88/91	0/0	13/16	26/37	0/0	0/0	9/19	-	189/282
Baseline	1724.37	30.53	21/22	0/0	32/34	0/0	24/27	39/50	0/0	0/0	13/19	65/101	194/253

3. **Continent Safety**. The relative threat from the opponent against continents completely occupied by the current player.
4. **Continent Threat**. The relative threat the current player poses against continents completely occupied by the opposing player.
5. **Distance To Frontier**. The average distance of troops from frontier territories. Effectively measures the army distribution of the player.
6. **Enemy Estimated Reinforcement**. An estimate of how many troops the enemy might recruit in the next turn.
7. **Enemy Occupied Continents**. The number of continents completely occupied by the opposing player.
8. **Hinterland**. The percentage of player territories that are not adjacent to any enemy territories.
9. **Maximum Threat**. A measure that looks at all possible attack source and destination combinations, calculates the victory probability of the battle and considers the maximum of these.
10. **More Than One Army**. The percentage of the current player's territories that have more than one troop on it.
11. **Occupied Territories**. The number of territories that the current player occupies in relation to the total number of territories.
12. **Own Estimated Reinforcement**. A measure that estimates how many troops the current player would recruit in their next turn.
13. **Own Occupied Continents**. The number of continents the current player occupies completely.

For more details on these feature calculations, see [36].