# Revisiting Move Groups in Monte Carlo Tree Search

Gabriel Van Eyck and Martin Müller

University of Alberta, Edmonton, Canada
vaneyck@ualberta.ca
mmueller@ualberta.ca

**Abstract.** The UCT (Upper Confidence Bounds applied to Trees) algorithm has allowed for significant improvements in a number of games, most notably the game of Go. Move groups is a modification that greatly reduces the branching factor at the cost of increased search depth and as such may be used to enhance the performance of UCT. From the results of the experiments, we conclude the general structure of good move groups and the parameters to use for enhancing the playing strength.

**Keywords:** Monte Carlo tree search, move groups, UCT, game tree search

## 1 Introduction

Using Monte Carlo search as a basis, the UCT algorithm has greatly improved the players for many games, including all state of the art Go programs, such as Fuego [8], and some Amazons program, such as Invader [7]. The UCT algorithm provides a method of selecting the next node to simulate in tree searches and has proven to be an effective way of guiding the large number of simulations that Monte Carlo search uses.

However, in several games, the complexity resulting from high branching factor makes it difficult to find the few good moves among thousands. Notably, Amazons on a tournament-sized board initially has a branching factor of 2176 and an average branching factor of approximately 500 [7]. Enhancements implemented to solve this problem tend to be game specific in nature. One potential refinement is the UCT algorithm's default policy for first-play urgency (FPU). First-play urgency is a value representing how soon to simulate a node for the first time. In basic UCT, the value is infinity; all nodes are simulated once before any further simulations [6]. In the game of Go, this value has been adjusted based on the playing performance to achieve a greater playing strength by Wang and Gelly [10].

Rapid Action Value Estimation (RAVE) is another refinement of UCT. RAVE has been used in many Go programs [5], including Fuego, MoGo, and Erica, to great success. Using RAVE modifies the original UCT algorithm to include an additional term that is based on statistics gathered for playing a stone at any point in the game, rather than just considering the next move.

When looking for enhancements that are not game-specific, an example is iterative widening. With iterative widening, the goal is to begin searching in a subset of possible moves, then gradually widening the search window if time allows. This has been used to improve Monte Carlo tree search by Chaslot et al. [3]. The enhanced algorithm won significantly more often with iterative widening, or Progressive Unpruning as they called it, turned on. Against GnuGo, they won 58% of games with iterative widening and another strategy turned on and only 25% with both turned off. Prior knowledge in the form of a pattern database was used in this case, but there is potential for less game-specific methods of selecting the move subset, such as using a RAVE value.

Another general strategy would be incorporating move groups into the game tree. A move group is a selection of available moves to introduce another layer in the tree. A player's turn is split into phases: first, select a move group, and then, select a move within that group. This method has been used in Amazons and Go. In Amazons, the largest gain is in move generation savings, as shown by Richard Lorentz [7]. In his program Invader, a turn consists of selecting an amazon, selecting a move for that amazon, then selecting a destination for its arrow. He notes that this saves time on the move generation itself by a factor of 10 simply because it does not have to generate all possible moves. For Go, Childs et al. were able to increase the playing strength of their modified libEGO program by grouping moves according to their Manhattan distance from both of the previous moves [4]. In this case the groups were not disjoint, unlike in the Amazons example. Another Go article by Saito et al. used disjoint groups composed first of moves within two points of the last move, then moves on the border of the game board, and lastly a group with all other moves [9]. Besides finding an increase in playing performance, they note that their player seems to shift focus better and play non-local moves when appropriate.

Given the need for a general enhancement when working with UCT, this paper explores move groups in detail. After analyzing the results from several experiments, an overall structure of move groups that perform well is found. This structure is then used to describe several potential applications that would effectively use this structure.

In this paper, first a quick overview of the UCB algorithm is presented in Section 2. Then, research questions to be answered are presented in Section 3. Next, the experimental framework is described in Section 4. Section 5 contains the experiments and their results. Lastly, Section 6 describes potential applications for move groups.

## 2 The Upper Confidence Bounds Algorithm

The Upper Confidence Bounds (UCB) algorithm is the basis of the UCT algorithm. UCB was designed for a finite-time policy for the multi-armed bandit problem [2]. At each time step, select the machine $j$ that maximizes the following formula:

$$\text{value}_j = \begin{cases} \bar{x}_j + C\sqrt{\frac{\ln(n)}{n_j}} & \text{if } n_j > 0 \\ FPU & \text{if } n_j = 0 \end{cases}$$

Here, $\bar{x}_j$ is the average reward from machine $j$, $n_j$ is the number of times $j$ has been played, and $n$ is the total number of plays. $C$ is the bias constant which is $\sqrt{2}$ by default. The FPU value is determined by the first-play urgency policy and is $\infty$ by default.

This algorithm has been proven to achieve logarithmic regret with any number of playouts. This was then extended to work with trees in the paper that describes the UCT algorithm [6]. Manipulating the bias term affects exploration; a higher constant means more exploration. When first encountering unexplored nodes, the default policy is to explore all children once; alternatively, a customized variation of first-play urgency is implemented. After the search is terminated, the machine or move with the most simulations is selected.

In this paper, UCB parameterized with $C$, the bias constant, and $FPU$, the first-play urgency, is the competitor. However, since the addition of move groups creates another level, UCB is used at both levels, which is essentially UCT with a fixed tree.

## 3 Research Questions

Previous efforts exploring the efficacy of move groups [4] [9] have tried a small subset of possible move groups. This paper examines the question of the perfect move group; one that performs better than all other possible move groups. In order to do this, all possible move groups need to be tested and compared to one another. Of course, the perfect move group will be defined by the underlying payoffs of the children. Therefore, the next question is to find whether or not there is a common structure among good move groups so that this might be applied to other problems. Finally, the last question is regarding the efficacy of completely random move groups. If we know absolutely nothing about the underlying nodes, will a random move grouping perform better than none at all?

## 4 Experimental Framework

Given the questions to be answered, an artificial game called the 'Multi-armed Bandit Game with Move Groups' was created:

1. The entire game tree is the root node with $N$ children.
2. Each child has a fixed probability of paying off.
3. Using UCB on this tree competes with other trees where move groups have been introduced; an extra level is added into the tree where first a move group is selected by UCB, then a child is selected UCB. No child is duplicated.

Simplicity was required here due to the nature of the questions. Given that structure and behavior of the move groups needed to be investigated, the simplicity allowed for larger experiments to be run to gather information in detail. Especially when $N$ is small, many different move groups can be analyzed.

The policy for selecting the next node to simulate was the modified UCB mentioned before. The bias constant was varied during the experiments with the following values used for $C$: 0.0, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0, and 100.0. The policy for first-play urgency used was to have a set value for unexplored nodes. In the default case, $FPU$ was set to 0.5; some of the experiments tested the effects of modifying this value.

## 5 Experiments and Results

Two quality criteria were used: probability that the best node was selected after a set number of simulations and total regret over time. All experiments were run 5000 times with average results shown.

### 5.1 Baseline Group Performance

The first experiment exhaustively tried every possible grouping of nine nodes into three groups of three, 280 combinations total. This allowed for detailed comparison of groups with different payoffs, and values of $C$ and $FPU$. Each grouping along with standard UCB was run for 8192 simulations. At various intervals, the node that would have been selected was tracked. Performance was measured by the percentage of time that the best node was selected. In practice, knowing the exact value or ranking of nodes is unrealistic, but this experiment provides a baseline to study the potential of groups.

For the first study, the payoff probabilities of the nine nodes were 0.1, 0.2, ..., 0.9. For comparison, results of UCB without move groups are given in Table 1. A 95% confidence interval for the previous selection rates is $\pm 0.71\%$ for 50% and $\pm 1.22\%$ for 75%. The best performing values of $C$ for basic UCB were 0.5 and 1.0 with 0.5 performing slightly better than 1.0 at lower simulation counts and slightly worse at higher simulation counts. To look at the effect of introducing move groups without changing any parameters, the performance of basic UCB was compared to all move groups with the same parameters. The results are shown in Table 2. When the bias term was near its optimal value for basic UCB, introducing move groups almost always decreased performance. So in order to find out how to use move groups to increase performance, comparisons must be made when also varying $C$.

Comparing all group and $C$ value pairs gives a better idea of the potential of move groups. When looking at the 32 simulations mark, there are 181 group/$C$ value pairs of a total 2520 (280 groups, 9 values for $C$) that performed better than the best basic UCB. The most common value for $C$ was 0.5 with a few 1.0 values being present as well. When looking at the 512 simulations mark, there are 224 group/$C$ value pairs that performed better than the best basic UCB.

Counting only the number of unique groups, there were 136. For values of $C$, 1.0, 2.0, 5.0, and 10.0 were all present, with 5.0 performing the best. The two most common structures of groups that performed better than basic UCB at both simulation marks were those that had the best child in a group with the 2nd or 3rd best child or the three best children split among the three groups. The best performing groups had the second structure more often than the first. For example, the grouping {{0.1, 0.2, 0.8} {0.3, 0.4, 0.7} {0.5, 0.6, 0.9}} selected the best node 62.24% of the time at 32 simulations compared to 54.22% for basic UCB and 100% of the time at 512 simulations compared to 99.66% for basic UCB.

| Bias Term | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.327 | 0.372 | 0.396 | 0.407 | 0.412 | 0.414 | 0.414 | 0.416 | 0.416 | 0.416 |
| 0.1 | 0.354 | 0.441 | 0.507 | 0.550 | 0.576 | 0.597 | 0.613 | 0.616 | 0.617 | 0.621 |
| 0.2 | 0.371 | 0.497 | 0.633 | 0.760 | 0.817 | 0.844 | 0.863 | 0.871 | 0.873 | 0.875 |
| 0.5 | 0.413 | 0.542 | 0.733 | 0.880 | 0.950 | 0.982 | 0.996 | 0.998 | 0.998 | 0.999 |
| 1.0 | 0.214 | 0.379 | 0.630 | 0.832 | 0.957 | 0.996 | 1.000 | 1.000 | 1.000 | 1.000 |
| 2.0 | 0.098 | 0.333 | 0.546 | 0.759 | 0.914 | 0.986 | 0.999 | 1.000 | 1.000 | 1.000 |
| 5.0 | 0.000 | 0.003 | 0.387 | 0.628 | 0.856 | 0.956 | 0.997 | 1.000 | 1.000 | 1.000 |
| 10.0 | 0.000 | 0.000 | 0.076 | 0.395 | 0.767 | 0.925 | 0.993 | 0.999 | 1.000 | 1.000 |
| 100.0 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.040 | 0.676 | 0.991 | 1.000 | 1.000 |

**Table 1.** Basic UCB performance based on percentage of the trials when the best arm was selected at various simulation counts with children payoffs ranging from 0.1 to 0.9.

| Bias Term | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 12.15 | 1.43 | 0.00 | 0.00 | 0.36 | 0.72 | 1.08 | 1.08 | 1.79 | 1.79 |
| 0.1 | 6.79 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.2 | 11.08 | 1.08 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.5 | 50.36 | 45.36 | 8.22 | 2.51 | 3.58 | 2.86 | 2.86 | 2.51 | 2.51 | 3.58 |
| 1.0 | 100.00 | 98.58 | 86.08 | 69.29 | 51.79 | 25.72 | 25.72* | 50.72* | 73.93* | 90.36* |
| 2.0 | 100.00 | 87.86 | 68.58 | 63.22 | 61.79 | 67.15 | 73.22 | 99.65* | 100.00* | 100.00* |
| 5.0 | 100.00 | 100.00 | 70.72 | 61.08 | 55.36 | 56.08 | 57.51 | 65.36* | 98.58* | 100.00* |
| 10.0 | 100.00 | 100.00 | 96.08 | 71.79 | 56.08 | 53.22 | 51.79 | 56.08 | 65.01* | 81.79* |
| 100.0 | 100.00 | 100.00 | 44.29 | 100.00 | 100.00 | 85.01 | 64.29 | 33.93 | 52.51* | 57.15* |

**Table 2.** Percentage of all groups that performed equally or better than basic UCB with the same parameters. An asterisk marks where basic UCB was selecting the best node 100% of the time.

### 5.2 Groups study in Difficult Move Selection

The second iteration of this experiment was run with the exact same parameters except the best child had a payoff probability of 0.81 rather than 0.9. This was to make the problem harder and decrease the occurence of 100% success rates seen previously. Basic UCB's performance is shown in Table 3. Again, the best observed bias term for basic UCB was around 0.5 and 1.0. When comparing move groups to basic UCB without changing parameters, the results are very similar to those found for the previous experiment shown in Table 2.

This time, when comparing all group and $C$ value pairs together, the results are more in favor of the groups. Using 32 simulations as the comparison point, 248 pairs and 162 unique groups performed better than the best basic UCB. The best performing $C$ value of those groups was 0.5, with 0.1, 0.2, and 1.0 present as well. In this case, the top groups have a very obvious structure in common: the best three children are split among the three groups. Using 512 simulations as the next comparison point, 435 pairs and 174 unique groups performed better than the best basic UCB. Interestingly, the same groups as with 32 simulations are present at the top, but with different bias terms. This time, 5.0 and 10.0 are the most common $C$ values and perform the best. At 8192 simulations, the same groups are still present at the top, with a majority having bias terms of 10.0. The group described in the last section is again among the top performers.

| Bias Term | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.273 | 0.306 | 0.320 | 0.329 | 0.333 | 0.335 | 0.336 | 0.336 | 0.338 | 0.338 |
| 0.1 | 0.274 | 0.332 | 0.374 | 0.400 | 0.424 | 0.444 | 0.456 | 0.460 | 0.461 | 0.466 |
| 0.2 | 0.286 | 0.358 | 0.419 | 0.455 | 0.478 | 0.498 | 0.515 | 0.532 | 0.551 | 0.570 |
| 0.5 | 0.278 | 0.335 | 0.436 | 0.501 | 0.551 | 0.593 | 0.645 | 0.690 | 0.754 | 0.842 |
| 1.0 | 0.165 | 0.248 | 0.360 | 0.463 | 0.545 | 0.595 | 0.642 | 0.707 | 0.779 | 0.864 |
| 2.0 | 0.079 | 0.216 | 0.316 | 0.411 | 0.484 | 0.542 | 0.611 | 0.667 | 0.749 | 0.842 |
| 5.0 | 0.000 | 0.002 | 0.200 | 0.332 | 0.432 | 0.515 | 0.555 | 0.617 | 0.696 | 0.801 |
| 10.0 | 0.000 | 0.000 | 0.067 | 0.202 | 0.375 | 0.478 | 0.550 | 0.607 | 0.675 | 0.757 |
| 100.0 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.268 | 0.415 | 0.551 | 0.667 |

**Table 3.** Basic UCB performance based on percentage of the trials when the best arm was selected at various simulation counts with children payoffs ranging from 0.1 to 0.81.

From these two experiments, we start to see a pattern emerge. In both experiments, groups that did better than basic UCB were those that had the best three children split among the groups. Then, for low simulation counts, a $C$ value of 0.5, near-optimal for basic UCB, performed best. For higher simulation counts, increasing the value by a few orders of magnitude had the best results. This is to counter for the increased selectivity that results from grouping.

Running the same experiment but with payoff probabilities of $0.3, 0.35, ..., 0.7$ had overall performance between the other two experiments and similar conclusions. Changing $FPU$ to 0.3 from 0.5 decreased the performance of all groups

and basic UCB. Changing it again to 0.7 from 0.5 had little effect with the largest deviation being 3%.

### 5.3 Grouping in Games with Large Branching Factor

In order to check if the best performing groups would have similar structure with a larger number of children, the experiment was modified slightly for 60 nodes and 10 move groups of 6 nodes each. Values for the nodes were randomly generated and ranged from 0.7% to 92.2%. The second best node had a value of 88.4%. The number of simulations was scaled up to 65536. Since there would be too many groups to exhaustively test them, 20 random groups as well as two pre-defined groups were generated. The first pre-defined group was the nodes "in order," that is, the best 6 children in one group, the next best 6 children in another group, and so on. The second pre-defined group was the "distributed" group where the best 10 nodes were split among the groups.

Results on the most part were similar to the 9-node experiments. A change from the previous results was the $C$ values that performed the best. For basic UCB, 0.2 performed best at simulations counts less than 1024 and 0.5 performed best at higher simulations counts. The best-performing $C$ values for groups also decreased; 0.2 and 0.5 were most common at low simulation counts and 1.0 and 2.0 were most common at high simulation counts.

The move group structure of the best groups was also different than expected. Two of the best-performing groups had the best child grouped with the 4th best child. This suggests that the move groups work best when distinguishing between a few good arms rather than many. The fact that the distributed group performed worse than basic UCB in all but a few cases reinforces this. These scaling results require further study.

### 5.4 Measuring Group Performance with Regret

Since the UCB algorithm was designed to minimize regret, it was important to look at how the regret of different move groups performs in comparison. Regret for one simulation was calculated by taking the difference of the best node and the chosen node's payoff probabilities. This amount was tracked at every time step and totalled. The values shown are the average total regret over the 5000 trials.

When the payoff probabilities were 0.1, 0.2, ..., 0.9 as in the first experiment, the range of total regret at simulation counts 512 and 8192 for all groups and basic UCB is shown in Figure 1 and 2 respectively. It is interesting to note that the relative shape between the bias terms stays the same no matter the simulation count. A bias term of 0.5 has the least regret throughout the simulations. However, this does not correspond to selecting the best node more often. For example, once the simulation count reaches 1024, some groups select the best node 100% of the time, but only if their bias term is 5.0 or 10.0. This is despite the fact that the regret of those bias terms is very high relative to that of 0.5. Higher bias implies more time is spent simulating lower value nodes and

therefore accumulates more regret. However, this allows the algorithm to better distinguish the best node.

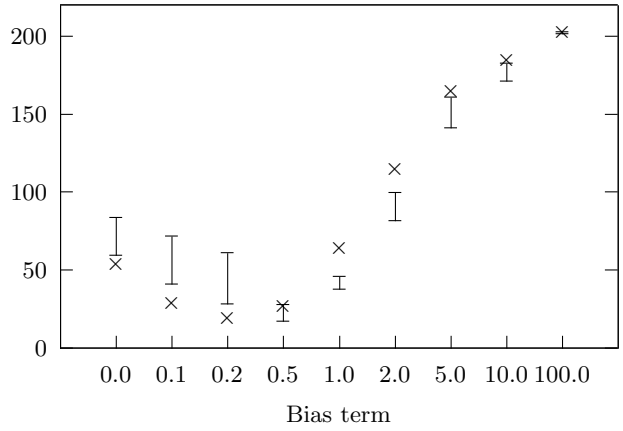


**Fig. 1.** Total regret of groups, range denoted by bars, and basic UCB, denoted by an x, after 512 simulations with payoff probabilities ranging from 10 to 90%. Data is separated according to the $C$ value.

Changing the largest payoff probability to 0.81 yields similar results. The range of total regret at a simulation count of 512 for all groups and basic UCB is shown in Figure 3. Similar results were also found for the 60 node experiment with slightly lower $C$ values having better regret.

From these results, cumulative regret is not the ideal metric for measuring performance of move groups. Even while relative total regret stayed the same between simulation counts, a group's preference of $C$ value changed. Since total regret considers all simulations performed rather than the end result, it is biased towards configurations that find good moves quickly and do not deviate. In game tree searches, we are more concerned with simple regret: finding the best move regardless of where simulation time was spent. This is similar to work done by Audibert et al. where they endeavor to develop an algorithm that can deal with situation where the end result is all that matters while still keeping exponentially decreasing regret [1].

### 5.5 Introducing Move Groups and Using Prior Knowledge

The previous experiments used predetermined groups in order to determine the group structures that performed well given the relative values of the nodes. One possible way to use these structures without knowing the exact value of the underlying nodes is to introduce move groups after a number of simulations. The

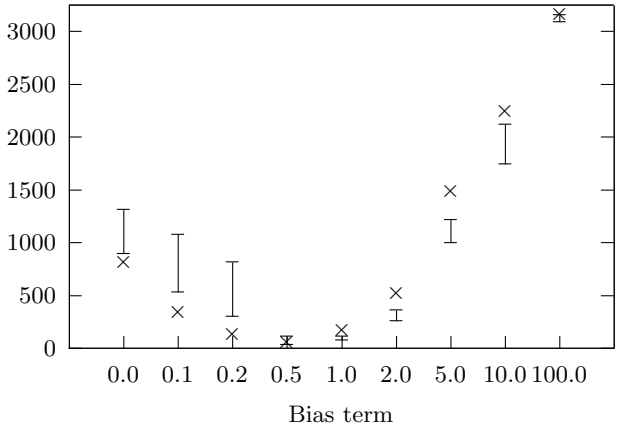


**Fig. 2.** Total regret of groups, range denoted by bars, and basic UCB, denoted by an x, after 8192 simulations with payoff probabilities ranging from 10 to 90%. Data is separated according to the $C$ value.

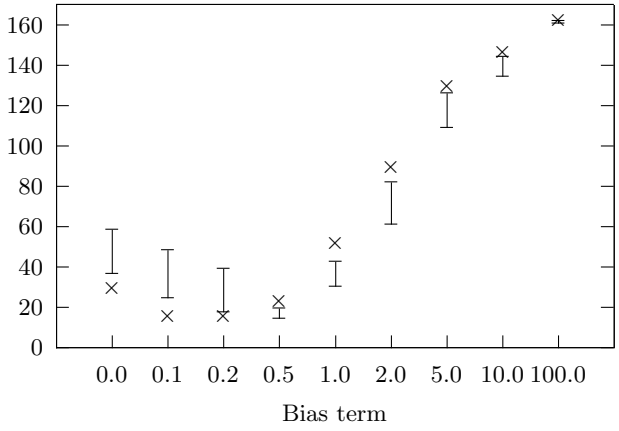


**Fig. 3.** Total regret of groups, range denoted by bars, and basic UCB, denoted by an x, after 512 simulations with payoff probabilities ranging from 10 to 81%. Data is separated according to the $C$ value.

values used for the nine nodes again was 10-81%. Basic UCB with a bias term of 0.5 was the competitor. Groups were created with various conditions: create groups once after 64, 128, or 512 simulations, create and recreate groups every 50, 100, or 200 simulations, create and recreate groups after each performance measurement. It is interesting to note that the run times of all grouping methods after 8192 simulations were approximately 25% faster than basic UCB due to the fact that fewer nodes are evaluated with groups.

First, creating the best group/bias pair was tried where the top three arms were split among groups and the bias term scaled to higher values with more simulations. Overall, the results were poor with all methods performing 7-12% worse than basic UCB at 8192 simulations which selected the best node 86.5% of the time. Next, the grouping where the best three nodes are grouped together with a bias term of 0.5 was tested. In this case, the performance of creating groups once after 512 simulations was within error bounds of basic UCB. Other methods did not perform as well with performance hits of 6-10% compared to basic UCB at 8192 simulations.

To test the effect of prior knowledge, groups were created initially given that the best three nodes are known. The bias term was varied from 0.1 to 10.0. Two grouping methods were tested – separating the best three nodes and grouping them together.

When separating the best nodes, the best performing bias terms were 2.0, 5.0, and 10.0. Initially, they performed worse than basic UCB. After 64 simulations, they started performing significantly better. However, 2.0 and 5.0 often switched to selecting the second best node rather than the best node at higher simulations. The group using a bias term of 10.0 performed significantly better than basic UCB after 64 simulations and reached 99.8% at 2048 simulations. Putting the best nodes in the same group had more consistent performance, but never better than basic UCB. Bias terms of 0.5, 1.0, and 2.0 all performed within the error bounds of basic UCB.

## 6 Conclusions and Potential Applications

There are several conclusions that can be drawn from the experimental results. The first is that simply introducing arbitrary or random move groups into the search only speeds up the search and does not increase the simulation efficiency. This is compounded when there are additional savings in move generation as shown by Lorentz's Amazons program [7]. Additionally, with very high bias terms, groups on average performed better than basic UCB in the experiments as seen in Table 2.

Since random move groups cannot always be used, we need some way to differentiate between the nodes so that they can be grouped properly. The experiments showed that splitting up the best nodes into a few groups had good results regardless of their underlying payoff. This was tested using simulation values and prior knowledge. Without prior knowledge, the best results were when the top three nodes were grouped together after a large number of simulations.

They performed as well as basic UCB in terms of number of simulations with a 25% increase in speed. Given that only nine nodes is a tough case and the branching factor would typically be decreased much more, this speed increase is expected to persist even with higher node counts. With prior knowledge, grouping the best nodes together at the start performed as well as basic UCB and had the speed increase. Splitting up the best nodes performed better with more simulations and higher bias terms. To exploit this in other games, the introduction of move groups during simulations could be used in general. That may prove to be challenging to implement due to the fact that the tree is changed during simulations. If prior knowledge is available, such as using a pattern database to rank moves, then groups could be created based on that ranking during node expansion.

The results described in this paper allow for several interesting avenues of research, such as introducing move groups during simulations for any game, using pattern databases in Go to rank moves and create groups, and using a heuristic function in chess to rank moves and create groups. Applying approaches to such games remains a topic for future work.

## References

1. Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. Best Arm Identification in Multi-Armed Bandits. In *COLT'10*, pages 41–53, 2010.
2. Peter Auer, Nicoló Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, pages 235–256, 2002.
3. Guillaume M. J-B. Chaslot, Mark H. M. Winands, H. Jaap van den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation (NMNC)*, 4:343–357, 2008.
4. Benjamin E. Childs, James H. Brodeur, and Levente Kocsis. Transpositions and Move Groups in Monte Carlo Tree Search. In Philip Hingston and Luigi Barone, editors, *IEEE Symposium on Computational Intelligence and Games*, pages 389–395. IEEE, December 2008.
5. Sylvain Gelly and David Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence*, 175:1856–1875, July 2011.
6. Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In *ECML'06*, pages 282–293, 2006.
7. Richard J. Lorentz. Amazons Discover Monte-Carlo. In *Proceedings of the 6th international conference on Computers and Games*, CG '08, pages 13–24, Berlin, Heidelberg, 2008. Springer-Verlag.
8. Martin Müller. Fuego at the Computer Olympiad in Pamplona 2009: a Tournament Report. Technical report, University of Alberta, Dept. of Computing Science, TR09-09, May 2009.
9. Jahn-Takeshi Saito, Mark H.M. Winands, Jos W.H.M. Uiterwijk, and H. Jaap van den Herik. Grouping Nodes for Monte-Carlo Tree Search. In *Computer Games Workshop 2007*, pages 276–283, 2007.
10. Yizao Wang and Sylvain Gelly. Modifications of UCT and Sequence-Like Simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games*, pages 175–182. IEEE, April 2007.