A comparison of sampling methods in MCTS

1st Graham Innocent
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK

2nd Malo Hamon
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK

3rd Pranav Gopalkrishna
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK

***Abstract*—We improve upon the basic MCTS player provided by the TAG framework by experimenting with alternatives to the UCB1 sampling algorithm, using the framework's implementation of the game Sushi Go! We present our results from experimental exploration of the parameter space of rollout depth for each of these sampling algorithms. To maximise our effectiveness in an expected tournament of agents we also introduce additional techniques such as the All Moves As First heuristic (AMAF), hard pruning and multiple-root averaging and show their effect.**

***Index Terms*—MCTS, UCB1, Thompson Sampling, Bayes-UCB**

I. Introduction

Games with a discrete selection of options, such as chess, can be modelled as a branching tree of options, with certain nodes representing an opponent's moves. An AI controller can therefore be constructed to find an optimal path through this hypothetical tree to try to select the action at each turn most likely to lead to game victory.

With some games, such as Tic Tac Toe, the game is simple enough to model the complete game as a shallow tree with sparse branches. Such a tree can be readily explored in full by brute force at each turn using a breadth-first search. However, other games such as Sushi Go! or Chess have a much higher branching factor and more depth to the tree representing a complete game. Exploring such trees by 'brute force' methods each turn is not computationally feasible [1], so moves are chosen either by heuristics based on domain knowledge of the game, or stochastic methods. In this study, we have focused on one such method, Monte Carlo Tree Search (MCTS), and have sought to improve upon the performance of the basic implementation provided.

A key factor in the performance of MCTS is the way we solve the exploration-exploitation dilemma, which can be best handled using bandit methods (both of these concepts are explained later). In this study, we implemented new players which use two different sampling methods: Bayes-UCB and Thompson sampling. We then describe the experiments we ran to compare their performance with UCB1 to select a final algorithm for tournament submission and choose optimal parameters. This report goes on to detail additional potential improvements implemented and tested in the construction of our submitted AI player (AMAF, multiple root averaging and hard pruning).

II. TAG

The experiments described in this report were all implemented using the Tabletop Games Framework [2], a set of components implemented in Java which allow researchers to define and run simulated board and card games. This framework allows tournaments of large numbers of matches to be run, thus allowing the study of the properties of new algorithmic game-playing approaches at scale.

The framework provides several predefined algorithmic agents as described below:-

1) Random - This player chooses moves at random with uniform distribution.
2) FirstAction - The 1st element (i.e. the element at index 0) in the list of available actions presented by the Forward Model is always chosen.
3) OSLA - Calculates the reward of every available action followed by random moves for each opponent, then selects the action with the highest reward.
4) RHEA - Implements a Rolling Horizon Evolutionary Agent strategy; a population of action sequences compete and are 'bred' to form the next population.
5) RMHC - A player based on the Random Mutation Hill Climb.
6) BasicMCTS - A rudimentary implementation of Monte Carlo Tree Search - identifying the best action at each step by sampling many random-walk depth-first searches through the action space. Uses the UCB1 algorithm for sampling.
7) MCTS - A fully developed, highly configurable implementation of MCTS, is excluded from further study in this report.

The framework also includes several pre-made games ready for experimentation, and one of these, Sushi Go! was used as the environment in this study. Sushi Go! is a card game for two to five players, which is stochastic due to cards being dealt from a shuffled deck, and features imperfect information due to other players' hands being hidden.

III. Background

A. Introduction to MCTS

MCTS is a class of game tree search algorithms that make use of simulated games to evaluate non terminal states. Simulated games select random actions until a terminal state is reached and the reward is averaged over multiple simulations to estimate the strength of each action [3] This approach is justified by the law of large numbers, a statistical law stating that the sample mean asymptotically approaches the true mean as the number of samples tends to infinity [4].

B. Explaining the MCTS algorithm

The core MCTS search loop has four key steps: selection, expansion, rollout/simulation and backpropagation.

- Selection decides which state to expand according to the tree policy. Selection stops when an unexpanded node is encountered, after which an action is selected at random.
- Expansion generates a child node (usually randomly) of the selected node via actions possible from this node
- Rollout/simulation generates a random sequence of actions originating from the child node of the selected node
- Backpropagation updates the statistics of the above node as well as its parents (up to the root node). The reward is either a terminal value (ex. win, lose or final score) or a heuristic

C. Default Tree Policy: Upper Confidence Bound (UCB1)

The UCB1 algorithm is a common tree policy used in MCTS. It adds an exploration term weighted by c to $\bar{X}_j$, i.e. the average reward of simulations passing through node j. n is the number of times the parent of node j has been visited, whereas n_j is the number of times node j has been selected from its parent [3]. The policy selects the node that maximises:

$$UCB1(j) = \bar{X}_j + c\sqrt{\frac{\ln n}{n_j}} \tag{1}$$

IV. Method

A. Handling the exploration-exploitation dilemma

Exploitation refers to making the best decision based on current information, whereas exploration means exploring more uncertain (less sampled) options to gather more information about the environment.

To resolve this, we can use bandit methods, i.e. methods in the multi-armed bandit problem framework, which are methods that aim to achieve the highest cumulative reward in the long run given that the rewards are initially unknown (i.e. unknown before exploration). More specifically, we considered sampling methods, i.e. bandit methods that utilise distribution sampling:

- UCB1 (default)
- Thompson sampling
- Bayes-UCB

We compared these to each other and to the default bandit method: upper confidence bound (UCB1).

1) Thompson Sampling: As an alternative to the UCB1 tree policy, we implemented a version of Thompson sampling [11]. Here, we model each action's value as a probability distribution; this distribution is sampled to obtain the action's value when required [12]. The algorithm starts by assuming prior distributions for each action; the parameters of these distributions are updated using Bayes' rule. By choosing prior and posterior distributions of the same family (conjugate priors), the computation of the posterior distribution becomes trivial.

Thompson sampling converges to an optimal minimax tree in certain settings [8], and is an efficient way to explore due to the randomization of samples [8]. In nodes with few visits (corresponding to actions that lead to these nodes), the variance is large and therefore samples may vary widely, leading to this node being explored further. As the node is explored more, the variance will shrink and the sampled values will get closer to the true value of the node. This should provide a better estimation of the node than UCB1.

We are modelling each node as a Normal distribution with unknown mean and variance [13]:

The variance is modelled as a Gamma Distribution:

$$\tau \sim Gam(\alpha, \beta) \tag{2}$$

The value of the node is modelled as a Normal distribution:

$$Q(s,a) \sim Normal(\overline{x}, \sqrt{\frac{1}{\tau}}) \tag{3}$$

To efficiently sample from the Gamma and Normal distributions, we utilised the inverse transform sampling trick from [15], which allows us to quickly sample from a distribution using the inverse CDF function of the distribution. Our code implementation uses the SSJ package from the University of Montreal [14].

2) Bayes-UCB: Another alternative to UCB1 we implemented was Bayes-UCB. Bayes-UCB applies Bayesian inference principles to calculate upper confidence bounds for each action's value estimate. [9]

The Bayes-UCB is conceptually similar to Thompson sampling, with a slightly different underlying distribution: the rewards are treated as a binary state; winning the game or not, rather than a continuous score. Therefore, in our implementation, we modelled the reward distribution of each node using a Beta distribution.

$$Q(s,a) \sim Beta(\alpha, \beta) \tag{4}$$

Where α and β are initially 1 to give a uniform distribution, then incremented with wins and losses respectively as observations are collected.

B. Hard Pruning

One way of improving the performance of a time-limited tree search algorithm is to selectively skip over unpromising branches. We implemented a variant of the hard-pruning approach described by Hsu & Perez-Liebana, 2020 [10]

$$remainingNodes = max(\alpha logn, T) \quad (5)$$

α and T control the minimum number of branches to retain. We sort nodes according to a heuristic based on expected reward, check a node has been visited 20 or more times and mark nodes for skipping until we have $remainingNodes$ left. For large enough values of α log n or T, no pruning may occur at a particular node.

C. All Moves As First Heuristic (AMAF)

The AMAF Heuristic was first introduced in the game of Go [5], it changes the rollout propagation to include all sibling nodes of the selected node if it matches an action visited during the rollout [6]. The idea behind this is that it assumes certain moves are strictly good or bad for a particular player [7], and therefore the nodes representing these actions should be updated regardless of if they were visited in the particular selection process or not. We implemented the RAVE implementation of AMAF [6], where the AMAF backups and normal backups are computed separately, and merge them using a coefficient:

$$Q_{RAVE} = \alpha Q_{AMAF}(s,a) + (1-\alpha)Q(s,a) \quad (6)$$

$$\alpha = \max\{0, \frac{V - N(s)}{V}\} \quad (7)$$

The AMAF heuristic can multiply the information gained by a Monte-Carlo rollout but (maybe dangerously) ignores the order of actions [6]. In Sushi Go!, each action represents selecting a card, so there is no strict requirement for actions to be taken in order, which makes us think implementing RAVE will have a strong impact on MCTS.

D. Handling Partial Observability

The basic MCTS algorithm assumes perfect information, thus the term *perfect information* MCTS (PIMCTS). Under this assumption, it fixes a root state from which the rest of the search tree is generated. However, it is not possible to fix a root state accurately if we have either (1) partial information about the game state or (2) stochasticity in the initialisation or progression of the game state. In such cases, all we can do is initialise its root with a determinisation, i.e. a possible game state compatible with what we know. To correct for this, we have implemented multi-root MCTS. The core steps of the algorithm are as follows:

function play(player, infoset, n):
// infoset $\implies$ information set for player
// n $\implies$ number of roots to generate
for n iterations do:
randomly choose a determinisation d from infoset
generate MCTS search tree for d
save root, actions taken, children and reward statistics
get the average rewards for each first-level action
return action with highest average reward

We do not modify the MCTS search itself, instead averaging over the results of multiple searches. Importantly, since we use multiple MCTS calls, the budget given for the MCTS search algorithm must be divided by the number of roots we explore.

V. Experimental Study

A. Thompson Sampling

We compared UCB1 to Thompson Sampling in Sushi Go! with 500 games and our standard parameters. The table below shows the win rate and Mean Ordinal from both agents. In the appendix, a plot of the distribution of each child action at the root node can be found in the appendix, showing how Thompson Sampling works, at first the actions have a high variance (very wide distributions) and as the number of iterations increases, the variance decreases as the value of the action becomes clearer.

Agent	Win Rate	Mean Ordinal
UCB1	0.19 +/- 0.01	1.80 +/- 0.01
Thompson	**0.81 +/- 0.01**	**1.19 +/- 0.01**

B. Bayes-UCB

We compared the Bayes-UCB sampling method to UCB1 in Sushi Go! with 1000 games, and our standard parameters. The table below shows the win rate and Mean Ordinal from both agents.

Agent	Win Rate	Mean Ordinal
UCB1	0.22 +/- 0.01	1.78 +/- 0.01
Bayes-UCB	**0.78 +/- 0.01**	**1.22+/- 0.01**

As this result is close to the performance achieved by Thompson sampling above, a further 1000-game trial was run to establish which sampling algorithm would be the best tournament contender for this assignment:

Agent	Win Rate	Mean Ordinal
Bayes-UCB	0.46 +/- 0.01	1.54 +/- 0.01
Thompson	**0.54 +/- 0.01**	**1.46+/- 0.01**

C. Hard Pruning

Early exploration showed that pruning effects were small, so we ran 3000-game tournaments with pruned versus base versions of each algorithm and still saw no statistically significant impact at 95% confidence.

D. AMAF (RAVE)

We compared the MCTS Search with and without the RAVE heuristic in Sushi Go! with 500 games and standard parameters of 100ms of time budget, a maximum tree depth of 30 and a rollout length of 20. We ran this experiment with multiple different values of $amafV$, the hyperparameter which controls how much the AMAF values are used.

amafV	Normal Win rate	RAVE Win rate	Normal M.O.	RAVE M.O
50	0.48 +/- 0.01	0.52 +/- 0.01	1.51 +/- 0.01	1.48 +/- 0.01
100	0.49 +/- 0.01	0.51 +/- 0.01	0.51 +/- 0.01	1.49 +/- 0.01
200	**0.45 +/- 0.01**	**0.55 +/- 0.01**	**1.55 +/- 0.01**	**1.45 +/- 0.01**
300	0.51 +/- 0.01	0.49 +/- 0.01	1.48 +/- 0.01	1.50 +/- 0.01

E. Multi Root

We ran our multi-root agent against the 1 root version of itself for both the UCB1 tree policy. We ran experiments for 500 games and our standard parameters.

nRoots	Win rate	1 root Win Rate
5	0.67 +/- 0.02	0.34 +/- 0.02
10	0.74 +/- 0.02	0.26 +/- 0.02
20	**0.78 +/- 0.02**	**0.23 +/- 0.02**

F. Comparison

We then ran all these improvements to the BasicMCTS agent in a round-robin tournament to see how they compare to each other. We ran 500 games and our standard parameters. We also included the Random and RHEA agents from the framework. The results are shown below:

Agent	Win Rate	Mean Ordinal
Thompson	0.82 +/- 0.00	1.18 +/- 0.00
Bayes-UCB	0.76 +/- 0.00	1.23 +/- 0.00
Multi-root	0.63 +/- 0.00	1.36 +/- 0.00
BasicMCTS	0.48 +/- 0.00	1.51 +/- 0.00
AMAF	0.48 +/- 0.00	1.52 +/- 0.00
RHEA	0.32 +/- 0.00	1.68 +/- 0.00
Random	0.00 +/- 0.00	2.00 +/- 0.00

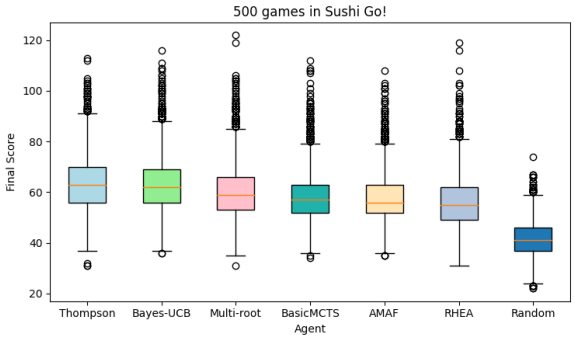

Fig. 1. Round robin tournament results comparing all agents.

VI. Discussion

A. Thompson Sampling

Our experiments show Thompson Sampling is a clear improvement over UCB1 as a tree policy, beating it over 80% of the time. Our Thompson sampling agent beat all other agents in the round-robin tournament. We believe this result is due to a more efficient exploration of the action space due to the sampling from distributions that Thompson sampling provides. Thompson Sampling is known to converge faster in cumulative regret than UCB1 [16], which means that it wastes less time visiting bad nodes.

B. UCB-Bayes

In the context of Sushi Go!, Bayes-UCB was also a significant improvement upon UCB1 but a 1000-game trial noted above demonstrated that it loses more often than it wins against Thompson sampling playing Sushi Go! under our standard parameters.

We hypothesise that Thompson sampling learns more over time as it is based on a continuous reward (score) rather than the simple binary outcome we used in Bayes-UCB; whether the player wins the game at the end of the rollout or not.

C. AMAF

The RAVE heuristic we implemented seemed to have a small effect, having a win rate of 55% over the normal UCB1 algorithm. However, in the full round-robin tournament, the BasicMCTS agent beats the AMAF agent. We therefore conclude that AMAF is not that effective for Sushi Go!

D. Mutli-Root

As expected, running multiple determinisations of the root MCTS node yields significantly better results than the BasicMCTS agent, beating it more and more as the number of roots gets larger. However, the multi-root agent was beaten by both Thompson and Bayes-UCB, indicating that these improvements are more significant than having multiple roots. The performance of multi-root MCTS demonstrates that agents in partially observable domains benefit from exploring multiple different game state possibilities (from the agent's perspective) rather than a single fixed determinisation. However, since the budget for each MCTS search is a fraction of the full budget, there is a lesser evaluation of long-term effects; this may have held back the algorithm's performance.

E. Comparison

In our final experiment, Thompson sampling and UCB Bayes come out as the clear winners, indicating that they clearly improve on the exploration of the action space compared to UCB1. Multi-root also beat out the Basic MCTS implementation, although by a less significant margin. We observe that the interquartile range is similar for all agents, indicating that they all perform pretty consistently, with some agents being better than others.

VII. Conclusions and Future Work

We have found that we can improve on the UCB1 sampling algorithm as an MCTS tree policy by utilising Bayesian inference. The parameterisation of nodes as a distribution yielded promising results.

We also found that we can significantly improve the performance of MCTS in the domain of Sushi Go!, by utilising multiple determinisations of the root node. We also found that certain methods do not work as well in Sushi Go! as they do in other domains such as pruning and the AMAF heuristic.

A key area that needs further work is the combination of multiple strategies outlined in this paper. In particular, the combination of multiple roots with Thompson sampling or Bayes-UCB as a tree policy should yield a significant improvement over the single determinisation version. However, in our experiments, the multiple root implementations of both of these tree policies underperformed with respect to their single root variant.

REFERENCES

[1] D. Feng, C. Gomes, and B. Selman, ”Graph Value Iteration” https://arxiv.org/pdf/2209.09608.pdf, September 2020.

[2] R.D. Gaina, M. Balla, A Dockhorn, R. Montoliu, D. Perez-Liebana ”TAG: A Tabletop Games Framework” http://www.diego-perez.net/papers/TAG_Tabletop_Games_Framework.pdf

[3] Cowling, Powley, and Whitehouse 2012 “Information Set Monte Carlo Tree Search.” IEEE 4, no. ISSN 1943-068X (June). https://eprints.whiterose.ac.uk/75048/1/CowlingPowleyWhitehouse2012.pdf

[4] E. Steinmetz and M. Gini, “More trees or larger trees: Parallelizing Monte Carlo Tree Search,” IEEE Transactions on Games, vol. 13, no. 3, pp. 315–320, 2021. doi:10.1109/tg.2020.3048331

[5] S. Gelly, D. Silver ”Combining online and offline knowledge in UCT” ICML 2007: Proceedings of the 24th international conference on Machine learning, June 2007, Pages 273–280, https://doi.org/10.1145/1273496.1273531

[6] D. P. Helmbold, A. Parker-Wood ”All-Moves-As-First Heuristics in Monte-Carlo Go” https://users.soe.ucsc.edu/ dph/mypubs/AMAFpaperWithRef.pdf

[7] T. Ager ”Monte Carlo Tree Search for Go” https://pats.cs.cf.ac.uk/?p=384&n=final&f=1-%20MCTS%20for%20Go%20Final.pdf&SIG=aaa311815c146f65f167a7318

[8] B. Lindenberg, K Lindahl March 2023 https://arxiv.org/pdf/2303.03348.pdf

[9] E. Kaufmann, O. Cappe and A. Gariver ”On Bayesian Upper Confidence Bounds for Bandit Problems” Proceedings of Machine Learning Research http://proceedings.mlr.press/v22/kaufmann12/kaufmann12.pdf

[10] Y. Hsu and D. Perez-Leibana. n.d. “MCTS pruning in Turn-Based Strategy Games.” Queen Mary University of London, London, UK. http://www.diego-perez.net/papers/MCTSPruningTribesAIIDE2020.pdf.

[11] “Thompson Sampling: The Algorithm for Efficient Decision Making.” 2019. Learn Statistics. http://www.learn-stat.com/parallel-distributed-thompson-sampling/.

[12] “What is Thompson sampling?” n.d. Autoblocks. https://www.autoblocks.ai/glossary/thompson-sampling

[13] S. Robers, “Thompson Sampling using Conjugate Priors,” Towards Data Science,

[14] P. L’Ecuyer, L. Meliani, and J. Vaucher, ‘SSJ: A Framework for Stochastic Simulation in Java’, in Proceedings of the 2002 Winter Simulation Conference, 2002, pp. 234–242.

[15] S. Olver and A. Townsend, ‘Fast inverse transform sampling in one and two dimensions’, arXiv [math.NA]. 2013.

[16] A. Bai, F. Wu, Z. Zhang, and X. Chen, “Thompson sampling based Monte-Carlo Planning in pomdps,” Proceedings of the International Conference on Automated Planning and Scheduling, vol. 24, pp. 29–37, 2014. doi:10.1609/icaps.v24i1.13616

APPENDIX

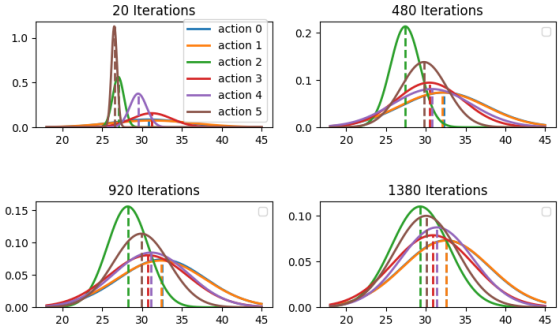

Fig. 2. Thompson Sampling distribution of actions at different timesteps

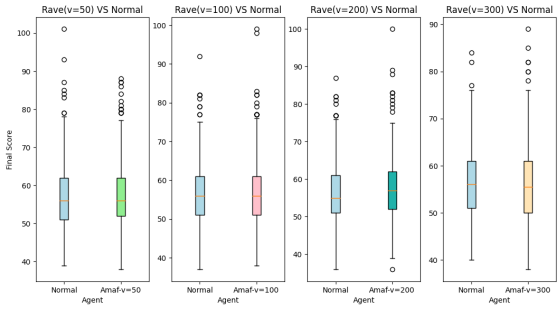

Fig. 3. Effect of amafV parameter in AMAF

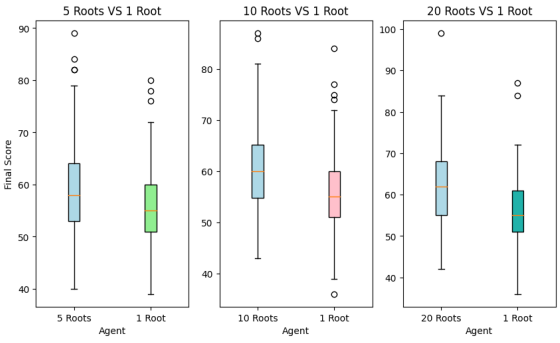

Fig. 4. Effect of nRoots in Multi Root