Enhancing Monte Carlo Tree Search for Sushi Go!: Integrating Determinization, Heuristic Rollouts, and Parameter Optimization in a Partially Observable and Stochastic Domain

(Ibrahim Hussain: ec25271@qmul.ac.uk, Weijun Zhong: ec251101@qmul.ac.uk, Oğulcan Özen: ec25061@qmul.ac.uk**)**

1. Abstract

Sushi Go presents a challenging partially observable and stochastic environment due to hidden hands, random card shuffles, and a large combinatorial state space. This report shows our enhancement of the Basic Monte Carlo Tree Search agent within the TAG Framework. This improved agent integrates determinization, to deal with hidden information, domain specific heuristic rollouts, replacing random rollouts, and tuned parameters, in particular its rollout length and max tree depth. These three alterations are designed to improve the agent's decision-making efficiency and stability under limited computation time. Experiments demonstrate significant performance increases from its baseline BasicMCTS agent, showing the benefits of applying domain aware heuristics driving rollouts and information awareness to algorithms within environments like Sushi go.

2. Introduction

Sushi Go is a partially observable and stochastic card game that poses unique challenges for artificial intelligence agents. Each player simultaneously selects a card from their hidden hand (only they know at this moment, what cards are in their hand) before passing the remainder to the player next to them, which creates uncertainty over both current and future game states. The shuffling of cards and simultaneous card selection contributes to the gigantic combinatorial state space (Klang et al., n.d.), which makes Sushi go a good place to test algorithms like Monte Carlo Tree Search under imperfect information and time constrictions.

Monte Carlo Tree Search is one of the most used approaches for sequential decision-making in games, due to its balance of exploration and exploitation through repeated sampling (simulations) of possible outcomes of the game state (Browne et al., 2012). However, in domains like Sushi Go, baseline or "vanilla" implementations such as BasicMCTS perform too inconsistently. Its uniformly random simulations disregard the ideas of scoring based upon sets of cards from one turn to the next - for example, collecting two Tempura for a 5-point increase or sets of Sashimi, but also failing to take into consideration what an opponent's hand could look like (Soen, 2019). This games' quality of hidden information and stochasticity also violates the perfect information assumptions of classical search methods like Minimax and Alpha-Beta pruning, which is an extension of minimax that prunes branches that are less desirable based on heuristics (Liu and Bailey, 2019).

Embring Klan et al's paper on 'Assessing Simultaneous Action Selection and Complete Information in TAG with Sushi Go!' suggests that agents must use mechanisms to reason under uncertainty. Research on simultaneous-move search (Saffidine, Finnsson and Buro, 2021) and extensions of Minimax (Liu and Bailey, 2019) reinforce the need for information awareness in search methods that can manage stochasticity and simultaneous decision processes. These insights are mostly behind the motivation to the adaptations made to BasicMCTS that implement domain knowledge and probabilistic reasoning guiding rollouts.

This research aims to extend the BasicMCTS implementation in the TAG framework by making three enhancements designed to address the structural complexity of Sushi Go:

1) Determinization: sampling plausible hidden hands to approximate full game states, enabling more informed rollouts (OpenAI, 2025).
2) Heuristic guided rollouts: replacing the random simulations in BasicMCTS with a domain specific (Sushi Go) heuristic that prioritizes synergistic and high value card selections (Soen, 2019)
3) Parameter tuning: optimizing the rollout length and maximum tree depth to achieve efficient convergence within the time budget (1000ms) (OpenAI, 2025).

By combining these three improvements, this upgraded BasicMCTS agent aims to achieve more consistent and strategically informed play than the vanilla baseline. Our results find how basic tree search algorithms can be manipulated to achieve greater results in partially observable and stochastic domains.

3. Background Information

3.1. TAG Framework

The TableTop Games (TAG) framework is an environment designed for developing, testing and comparing AI agents across a range of turn-based and simultaneous-action board games (Klang et al., n.d) built in Java. It provides a modular architecture where agents can interact through the ForwardModel (essentially an API), to take actions and receive information about the particular state of the current game, allowing agents to query available actions and simulate outcomes. Within TAG, all game information is encoded and retrieved by the agent through the AbstractGameState (via the Forward model), which determines what each player can see. This allows modelling of perfect-information games, like TicTacToe, as well as partially observable games, like Sushi Go. By creating a modular single interface for agents, TAG enables reproducibility and fair comparison.

3.2. Sushi Go! Game

Sushi Go is a card drafting game for two to five players, played over three rounds. Each player begins a round with a hidden hand of cards; each player picks a card and simultaneously reveal which card they picked before passing their current hand clockwise to the next person. Due to hands being hidden until a full revolution has been made and the shuffling of cards between rounds, the game is considered as partially observable and stochastic, as players must reason about uncertain card distributions (Klang et al., n.d). The game's scoring system rewards synergies between cards rather than picking isolating cards. For example, by choosing two Tempura cards can return 5 points, three Sashimi cards return 10, whilst Dumplings score progressively more with each additional card. Other cards like Wasabi, multiply the value of Nigiri cards. These interdependencies describe the synergy rewarding system in Sushi Go, causing a non-linear payoff structure, which complicates the estimation of immediate versus delayed rewards. Since hands of cards are switched, each decision made can affect the next opponents' choices- introducing the chance to integrate strategic predictions of what strategies other plays may have (Soen, 2019). This combination of hidden information, card shuffling (stochasticity) and combinatorial depth, estimated at over 10^59 possible states for a 4-player game, making Sushi Go an ideal benchmark for search algorithms operating in similar environments.

3.3. Monte Carlo Tree Search

Monte Carlo Tree Search is a family of best-first search algorithms that estimate the value of actions through repeated random sampling of simulated game trajectories (Browne et al., 2012). It proceeds iteratively through four stages: **selection, expansion, simulation** and **backpropagation**, to incrementally build a search tree that concentrates its efforts on the most promising parts of the state spaces (the set of all possible states the game could become). During selection, the algorithm traverses the current tree using a tree policy that balances the exploration and exploitation. This is known as the Upper Confidence Bounds applied to Trees (UCT), which selects the action that maximizes the following:

where X_j is the average reward of child j, N is the visit count of the parent, n_j is the visit count of the child, and C_p is an exploration constant parameter, controlling the tradeoff between the two (Browne et al., 2012).

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

The simulation (rollout) phase plays the game to completion using a default policy, which can be altered to be random or guided and propagates the terminal reward upward during backpropagation. Therefore, MCTS is considered an anytime algorithm, capable of returning an improved decision the longer it runs, and a heuristic that requires little domain knowledge. Despite this flexibility MCTS assumes that game states are fully observable (no hidden information) and deterministic, which in games like Sushi Go, these random rollouts often fail to provide meaningful evaluations due to their ignorance of domain specific patterns and hidden information. Resulting in our enhanced version of this baseline MCTS to adapt it to such domains.

3.4. MCTS in Imperfect Information and Stochastic Games (determinization)

A common technique for dealing with partial observability is determinization, in which the hidden aspects of a game are sampled to form a set of fully observable “determinized” states. Each determinization is searched independently using MCTS, and the aggregated statistics guide final action selection (Browne et al., 2012). While simple, determinization can lead to “strategy fusion” errors where the agent assumes knowledge across incompatible states. Nevertheless, it remains a practical baseline for many stochastic games.

3.5. Heuristics and Rollout Policies

Replacing random simulations with **heuristic-guided rollouts** is a common way to improve MCTS performance. A heuristic uses knowledge about the game to guide the search toward more useful actions, helping the algorithm learn faster and make more consistent choices (Browne et al., 2012). In Sushi Go, this means the agent can use information about card types and scoring patterns instead of playing moves randomly.
A heuristic usually has the following features:

- Immediate score potential- how many points a card gives a player immediately.
- Set completion chance- how likely a card helps you finish a card set for valuable combos.
- (Less usual) Opponent blocking- how much a card will prevent an opponent from scoring well.

1.6. Summary

In summary, the TAG framework enables experimentation with search-based agents across diverse tabletop games. Sushi Go provides an ideal benchmark due to its mix of stochasticity, partial observability and simultaneous-action selection. Baseline MCTS offers a general foundation for such domains, but its performance depends heavily on the quality of information modelling and rollout s. The research literature shows that combining determinization, heuristic guidance and parameter optimisation can meaningful improve MCTS performance under uncertainty (Browne et al., 2012; Saffidine, Finnsson and Buro, 2012; Perez and Oommen, 2019; Soen, 2019; Embring Klang et al., 2021).

Method

4. Describing our approach

Based on the criteria of the coursework given, the approach we took was to enhance the base version of the basic vanilla MCTS model and build on its base functionality/features in order to improve its performance in playing SushiGo. We started off with doing a large amount of research looking into the areas of improvements and potential optimisations that we could implement for enhancing the basic vanilla MCTS model, for example going through the academic materials and sources, then we elicited a 'game' plan by listing and finalising the actual improvements to add. The enhancement is achieved through implementing heuristic-guided rollouts, adding determinisation, and adjusting preset values of parameters such as the 'rolloutLength' and 'maxTreeDepth' etc. This approach enables the basic vanilla MCTS agent to go from being a fully stochastic player that learns from random simulation with no game knowledge or strategic planning ability at all, to being a player that is able to think rationally about what the current best move at a given game state is and strategically 'plan ahead' using heuristics and determinizations to maximise scores. Our ideas were inspired by Gelly and Silver (2007) which showed that incorporating domain knowledge into rollout policies can dramatically improve MCTS performance, particularly in games where random play is weak Their work on combining Monte Carlo simulation with value function approximation demonstrated substantial improvements in Go.

4.1. How our agent works

As mentioned above, our agent model is an enhanced version of the basic vanilla MCTS model. The core function of our current model remains the same as the base line model (basic, vanilla MCTS) as it still plays the game through simulations and rollouts using the four core stages of MCTS algorithm, which is selection, expansion, simulation and back-propagation.

First, during our model's selection stage, our agent traverses down the existing tree using UCB1 with adversarial adjustment which adds adversarial awareness by assuming that opponents are minimising our score when it's their turn. The exploration rate is set as 'K = **√2**' which helps balancing out exploitation and exploration. A small amount of random noise is also added to the model’s rollouts to prevent deterministic bias.

For the expansion stage, our agent expands the tree upon reaching a node inside the tree with unexplored actions. It achieves this by uniformly sample unexplored actions and create a new child node for the unexplored actions then apply simulations and MCTS to it afterwards in order to learn and improve performance, in SushiGo this would be creating a node for an unplayed card choice. The agent will not create infinite trees due to the 'maxTreeDepth' value that is adjusted and budget management with using the forward tracking model that exists.

For simulation and rollout, our agent uses heuristic-guided rollouts instead of fully random rollouts to make a smarter choice in the decision of picking the best action to take. Our agent will evaluate each action using the SushiGo specific heuristics that we have implemented to score the best possible action so that the agent knows which would be the best available play at a given game state, it is like evaluating the situation strategically to know which move would yield the most points based on heuristics. However, our agent's play isn't fully based on heuristic as we added a little amount of epsilon to encourage some random plays in order to prevent being too predictable. Determinization is applied to handle hidden information in the game, our agent does this by creating a completing copy of the game states and randomly fills in opponent hands from the unseen cards which creates a “possible world” and then build MCTS tree and apply simulations on it as if it were the real game. This will accumulate statistics which then helps the agent to pick the best average move.

For the final stage, our agent uses backpropagation to propagate results up the tree and updates all ancestor nodes. It stores sum of all rollout results through one node and tracks how many times this node was visited then updates from leaf to root (not just immediate parent) that prompts agent to select the most suitable final action in the real game state. For selecting the final action, the agent will use the maximum visit count as the selection criterion because visit counts indicate robustness across different scenarios which is much more reliable as well than just picking action with the highest single-scenario value.

4.2. Our Modifications

For our modifications, we implemented 'Heuristic-Guided Rollouts' through creating a set of new heuristics specifically for SushiGo which is stored in the heuristics folder within the TAG framework, it works as a replacement of the original score heuristic that the basic vanilla MCTS model uses. It is then applied through adding a few new tunable parameters called 'useHeuristicRollouts' which is a boolean type that could be turned on when its value is set to 'TRUE' or off when it is 'FALSE', and 'heuristicRolloutProbability' with its value set to 0.9 (percentage of heuristic rollouts vs random rollouts). The purpose of the two new parameters is that to replace purely randomised simulations with domain-informed action selection during the rollout phase in order to enable our agent to make plausible strategic plays, value more accurate estimates as simulations could represent real game play-out and learn which moves lead to strong positions.

Later, we also implemented determinization through adding the new 'nDeterminizations' parameter with its value set to 5 which handles imperfect information in SushiGo, where opponent hands are hidden and cards are passed. Whitehouse et al. (2011) specifically studied determinization in trick-taking card games and found that 5-10 determinizations provide a good balance between computational cost and decision quality their empirical results showed diminishing returns beyond 10 determinizations in most card game scenarios.

As the next step, we implemented parameter tuning techniques which involves adjustments made on 'rolloutLength', 'maxTreeDepth', and 'K' value (UCB exploration constant) in order to optimise computational efficiency by reducing computational costs per simulation and balance exploration-exploitation trade-offs so to make the model more robust as the model gains better tree statistics.

4.3. Experiment and Expectation

Lastly, we performed experiments by setting up a tournament game in-between the baseline Basic Vanilla MCTS, Random Player, and our enhanced version of Basic MCTS to play against each other for 500 match ups with different parameters being used and measure against different conditions, for example using heuristic only, determinization only, tuned parameters only, heuristic plus determinization, heuristic plus parameters, determinization plus parameters, and all three combined. As for results, we would expect to see significant improvements in win rate when our model plays against the baseline Basic Vanilla MCTS model and the Random Player model.

5. Results

To evaluate the performance of the proposed agent, the experimental study was partitioned into two distinct phases. The first phase consisted of a parameter tuning sweep to identify the optimal configuration for the agent. The second phase employed an ablation study to deconstruct the agent's architecture and quantify the contribution of its primary components: determinization, heuristic rollouts, and parameter tuning.

All experiments were conducted as 3-player tournaments, with the Improved BasicMCTS agent competing against the standard BasicMCTS agent and a random baseline player. Each experimental configuration was run 3 times, with each run consisting of a 498-game tournament. The win rates and mean scores presented are the average of these 3 runs.

5.1. Ablation Study

The first phase was an ablation study to isolate the impact of the agent's three core modifications: Parameter Tuning (P), Heuristic Rollouts (H), and Determinization (D). This study was designed to test the agent under generous time constraints. The "Full Upgrade" agent was run with a 1000ms budget, while the ablated (component-removed) agents were all run at 500ms.

Agent Configuration	Budget	Win Rate +/- Std. Error	Mean Score
No Determinization	500ms	89.2% +/- 0.014	53.22
No Param. Tuning (Det. + Heuristic + Default)	500ms	69.1% +/- 0.021	44.78
No Heuristic (Det. + Random + Tuned)	500ms	85% +/- 0.018	51.66
Full Agent (Det. + Heuristics+ Default)	1000ms	65.3% +/- 0.021	45.00

Table 1

5.1.1. Analysis of Component Importance

By comparing the three ablated agents (A, B, and C), all running at 500ms, we can clearly rank the importance of each component. Parameter Tuning (P) proved to be the most critical; the "No Parameters" agent (Row C) saw a performance collapse to 69.1%, a massive drop of ~20 percentage points from the top-performing "No Determinization" agent (89.2%). This proves the tuned parameters are the most vital component. The Heuristic Rollout (H) was secondary, as the "No Heuristic" agent (Row B) dropped the win rate to 85.0%, a significant 4.2-point reduction, confirming its value. Finally, Determinization (D) appears computationally expensive or harmful at this budget. The "No Determinization" agent (Row A) was the top performer at 89.2%, strongly suggesting the agent performs better by not running this component and using the time for more search.

5.1.2. Analysis of Increased Time Budget

A critical insight is revealed by comparing Experiment C and Experiment D. Both ran the exact same agent (Heuristic +

Determinization with default parameters), but at different time budgets. At 500ms (Exp. C), the agent achieved a 69.1%-win rate, but at 1000ms (Exp. D), the same agent's performance dropped to 65.3%. This demonstrates that simply providing more time does not guarantee better performance. In this case, the BasicMCTS opponent (whose win rate rose from 27.5% to 30.1% with the extra time) benefited more from the increased budget than our H+D agent did, effectively reducing our agent's relative performance. This highlights the vital importance of the tuned parameters found in the next section.

5.2. Parameter Tuning

To find the best configuration, a series of experiments were conducted. All variations of the Improved BasicMCTS agent (which included Determinization and Heuristic Rollouts) were run in a 3-player tournament against the baseline BasicMCTS and a random player. To test the agent's computational efficiency, a highly constrained budget of 40ms per turn was enforced for this tuning sweep.

The results of the 3 averaged 498-game tournaments for the most salient parameter combinations are presented in Table 2.

Exp #	Rollout Length	Max Tree Depth	Win Rate +/- Std. Error	Mean Score
(1)	10	10	89.4% +/- 0.014	53.15
(2)	10	20	88.8% +/- 0.014	53.88
(3)	13	5	90.0% +/- 0.013	54.07
(4)	13	10	89.0% +/- 0.014	53.71
(5)	13	20	89.0% +/- 0.014	53.51
(6)	15	5	89.0% +/- 0.014	54.51
(7)	15	20	90.4% +/- 0.013	54.51

Table 2

The data in Table 2 reveals a clear optimal configuration. The agent configured with ROLLOUT_LENGTH = 15 and MAXTREEDEPTH = 20 achieved the highest win rate (90.4%).

The key insight from this experiment is that the interaction between rollout length and maximum tree depth is complex. While increasing RL did not always improve performance (e.g., at MD=5, increasing RL from 13 to 15 dropped the win rate from 90.0% to 89.0%), the optimal configuration of RL=15, MD=20 achieved the highest win rate. This indicates that the longest rollout (RL=15) is only maximally effective when paired with the deepest search (MD=20), suggesting the 'sprint' budget is best spent by giving the agent *both* a long simulation and sufficient depth to find promising paths.

Conclusions and Future Work

This work successfully developed and analyzed an improved MCTS agent for the game of Sushi Go!. The experimental results yielded several critical insights. First, the ablation study definitively proved that parameter tuning was the single most important factor in the agent's performance, far outweighing the benefits of the heuristic rollout or determinization. An agent with tuned parameters achieved an 89.2%-win rate, while an untuned agent collapsed to 69.1% under the same 500ms budget.

Second, the parameter tuning sweep revealed that the agent's peak performance (90.4%-win rate) was achieved on a highly constrained 40ms budget. This was not due to a simple "deep search" strategy, but rather a complex interaction between rollout length and maximum tree depth The optimal configuration (RL=15, MD=20) indicates that the longest rollouts are only maximally effective when paired with a deep search, suggesting the 'sprint' budget is best spent by giving the agent *both* a long simulation and sufficient depth to find promising paths. Finally, the experiments showed that components like determinization can be computationally harmful, and that simply increasing an agent's time budget does not guarantee improved performance, especially when its parameters are not optimized for that budget.

Future work may focus on expanding the successful heuristic rollout, which provided a significant 4.2-point boost. This may be achieved by incorporating a simple opponent model, such as simulating opponents greedily taking their best card rather than randomly. This would create a more realistic simulation and likely lead to stronger strategies.

Use of Generative AI:

The use of generative AI in this report has been used throughout the construction of this report, to verify theoretical knowledge, supply definitions and build our ‘readme’ file. All information we have used in this document that has been generated partly by AI has been critically evaluated to ensure its correctness. Primarily its use has been for definitions and theoretical knowledge.

References:

[1] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, [online] 4(1), pp.1–43. doi:https://doi.org/10.1109/tciaig.2012.2186810.

[2] Klang, C.-M., Enhörning, V., Alvarez, A. and Font, J. (n.d.). Assessing Simultaneous Action Selection and Complete Information in TAG with Sushi Go! [online] Available at: https://ieee-cog.org/2021/assets/papers/paper_311.pdf.

[3] Liu, J. and Bailey, J. (2019). AI 2019 : advances in artificial intelligence : 32nd Australasian Joint Conference, Adelaide, SA, Australia, December 2-5, 2019 : proceedings. Cham, Switzerland: Springer.

[4] Saffidine, A., Finnsson, H. and Buro, M. (2021). Alpha-Beta Pruning for Games with Simultaneous Moves. Proceedings of the AAAI Conference on Artificial Intelligence, 26(1), pp.556–562. doi: https://doi.org/10.1609/aaai.v26i1.8148.

[5] Soen, A. (2019) Making Tasty Sushi Using Reinforcement Learning and Genetic Algorithms. Available at: https://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2019/paper/ABCs2019_paper_v2_81.pdf

[6] OpenAI (2025). *ChatGPT*. [online] chatgpt.com. Available at: https://chatgpt.com

[7] Kocsis, L., and Szepesv´ari, C. (2006). ”Bandit Based Monte-Carlo Planning.” European Conference on Machine Learning, 282-293

[8] Gelly, S., and Silver, D. (2007). "Combining Online and Offline Knowledge in UCT." International Conference on Machine Learning, 273-280.

[9] Whitehouse, D., Powley, E. J., and Cowling, P. I. (2011). ” Determinization and Information Set Monte Carlo Tree Search for the Card Game Dou Di Zhu.” IEEE Conference on Computational Intelligence and Games, 87-94.