Enhancing MCTS Performance in SushiGo! with Domain-specific Heuristics

Boran Sac
Queen Mary University of London
School of EECS
London, United Kingdom
b.sac@se25.qmul.ac.uk

Deniz Genco Atilla
Queen Mary University of London
School of EECS
London, United Kingdom
d.atilla@se25.qmul.ac.uk

Harishkumar Kishorkumar Prajapati
Queen Mary University of London
School of EECS
London, United Kingdom
p.harishkumarkishorkumar@se25.qmul.ac.uk

***Abstract*—Imperfect information card games pose significant challenges for Artificial Intelligence design, requiring agents to make strategic decisions with incomplete knowledge of game state. Monte Carlo Tree Search, an AI technique that uses random sampling of game playouts to build a search tree, has been successfully applied to various game domains. While MCTS traditionally relies on domain-independent evaluation through random rollouts, it can be enhanced with domain-specific heuristics. In this paper we firstly provided an overview of the TAG Framework, the game Sushi Go!, and MCTS. We then detail the methodology that used for developing our heuristic, including the processes for parameter tuning and the functionality of the basic MCTS agent. Finally, the experimental results are presented, followed by a discussion of potential future work in this area.**

***Index Terms*—TAG Framework, MCTS, SushiGo!, heuristic**

I. Introduction

AI agents playing games have long been used as valuable testing grounds for research in Artificial Intelligence, providing structured environments to explore decision making under uncertain scenarios. Among these, MCTS has been one of the most successful general algorithms so far, achieving strong results in many complex environments such as the game of Go and general frameworks like TAG [1].

In this report, we use Sushi Go! as a case study due to the challenges it gives to agents, like imperfect information and a scoring system for card sets. The agent must balance short term gains with long term goals, often without knowing the opponents' future actions. Therefore, a base agent without domain knowledge can struggle to identify the optimal cards choices since it fails to understand the game's scoring structure where sets give bonus points for collecting certain cards.

To address this limitation, our work proposes an enhanced MCTS that integrates a custom heuristic evaluation to improve decision making in non-terminal states. It evaluates the state desirability based on the played and held cards while factoring in the card synergies and round based multipliers. By using these heuristics with the MCTS agent, the agent gains a more informed estimate state quality, expected to improve the agent's overall performance.

The remainder of this paper is structured as follows: Section 2 - The background information on MCTS, the game of Sushi Go! and the TAG framework. Section 3 presents the design and implementation of our heuristic as well as the changes we made to the agent itself. Section 4 discusses the experiments we conducted as well as the results gained, and Section 5 concludes with future improvements.

II. Background

A. Tabletop Games Framework (TAG)

The Tabletop Games Framework is a research platform developed to support experimentation of different AI agents across many tabletop and card-based environments including but not limited to: Tic Tac Toe, Connect 4, Poker, and Sushi Go!. TAG provides an interface that for defining game logic, player agents and simulation conditions, giving them the freedom to design, test and evaluate search or evolutionary algorithms under consistent and controlled settings.

A key advantage of TAG is its ability to handle games with both perfect and imperfect information, allowing agents to reason under uncertain conditions while maintaining reproducible behaviour. The agents must act under strict constraints, meaning that the effectiveness of an algorithm depends on both decision quality and computational efficiency.

In this study, we use TAG provides the environment for our Sushi Go! heuristic applied to the basic MCTS agent where it's compared against non-heuristic basic MCTS agents for fairness.

B. Sushi Go!

Sushi Go! is a competitive multiplayer card-drafting game where players aim to maximise their individual score by selecting cards that form the high scoring combinations over 3 rounds [2]. The scoring system relies on card synergies and set collection, encouraging long-term strategies than short-term. For example, Wasabi triples the value of a Nigiri card played after it, and two Tempura cards give a score of 5, individually providing none. Sashimi works like Tempura, except 3 cards for the set. Puddings are not scored until the end of the Round 3, rewarding the player with the most Pudding and penalising the last player, becoming the deciding factor for winning in close games.

Therefore, in such a imperfect and uncertain domain, without heuristic guidance, MCTS's decisions heavily rely on random rollouts, often failing to recognise card synergy

patterns. This limitation is a significant motivator to develop our domain-specific heuristic, allowing MCTS to make more informed, optimal decisions during the search.

C. Monte Carlo Tree Search (MCTS)

The Monte Carlo Tree Search is a stochastic best first search algorithm used to find optimal decisions in domains by "taking random samples in the decision space and building a search tree based on the results" [1]. The method was first formalized by Coulom, who demonstrated the effectiveness of MCTS in stochastic search spaces like Go [3].

The search tree is built incrementally through repeated simulations with four main stages – Selection: Navigating the tree with a UCT formula; Expansion: Selecting an unvisited node; Simulation/Rollout: Estimating the outcome of the game through random actions; Backpropagation: The result of the game is passed back up the tree. The UCT formula balances the exploitation of strong actions with the exploration of less visited actions, enabling it to algorithm to discover better strategies over time. This approach allows MCTS agents to concentrate their search on promising actions and converge to the optimal strategy. The potential of MCTS has been presented throughout many studies, most famously in the work of Silver and his collauges, where a new algorithm that combines MCTS with deep neural networks enabled AlphaGo to achieve a "99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0", an unprecedented feat in AI research [4]. Despite these achievements, MCTS's performance can degrade in imperfect domains with delayed rewards and high branching factor.

To address these challenges, many extensions to MCTS have incorporated heuristics which have shown significant improvements in convergence speed and accuracy of decisions made.

D. Heuristics in Game Playing AI Agents

Heuristics are evaluation methods that estimate the potential of a game state. They allow agents to make quicker, more informed decisions by estimating how good a position is based on domain information, rather than a simulation [5]. In games with large branching factors or delayed rewards, heuristics are often vital to improving performance since they guide the agent through more promising strategies.

Within MCTS, heuristics can be used at multiple stages. During the rollout stage, random plays can be adjusted or replaced using heuristic evaluation to produce better outcomes, significantly improving the algorithm's performance [6].

Research has consistently shown that incorporating heuristics into MCTS improves its convergence speed and decision making. Finnsson and Björnsson applied learned heuristic control in GGP, biasing rollouts towards moves that have a history of providing better results [7]. Cowling and his colleagues proposed Information Set MCTS which is an extension of MCTS designed for imperfect information domains [8]. Its reasoning is based on information sets which are collections of game states that seem identical from the player's perspective and are used to perform multiple simulations, allowing the agent to make decisions under uncertain conditions.

These studies collectively highlight the benefits of heuristics and how it improves MCTS performance, allowing agents to make more strategic decisions under strict constraints. The following section details our implementation of domain-specific heuristics to the MCTS agent provided and our findings.

III. Method

A. Agent Architecture and Modifications

Our Custom MCTS agent extends standard MCTS by incorporating domain-specific heuristics into state evaluation. While basic MCTS handles tree traversal, UCB1 selection, and statistical updates, our modifications focus on evaluation of heuristic.

1) Heuristic Evaluation Function: We enhance standard MCTS win rate statistics with domain-specific evaluation:

$$\text{evaluation} = \frac{\text{base\_score} + \sum \text{heuristic\_components}}{50.0} \quad (1)$$

bounded between -0.5 and 3.0. Our heuristic has three components: The heuristic evaluation considers three components: **(1) Base Game Score** – the actual score following TAG's Sushi Go! rules; **(2) Played Cards Evaluation** – assessing current and potential points through set completion (Tempura pairs: 5 pts, singles 50%; Sashimi triples: 10 pts, partials 50%/20%; Dumplings: 1–15 pts), Nigiri (Squid 3 pts, Salmon 2 pts, Egg 1 pt, Wasabi $\times 3$), majorities (Maki 1st = $1.0\times$, 2nd = $0.8\times$, others = $0.5\times$; Pudding = $1.0\times$ with round multipliers), and utilities such as unused Wasabi (2.0 pt potential); and **(3) Hand Evaluation** – estimating future scoring at 50–60% of completion value, including set potentials (Tempura/Sashimi 50%, Dumpling 60%), Wasabi–Nigiri combos (70% played, 60% hand), and Chopsticks utility ($2.5\times$ if hand > 5 cards, else $1.5\times$).

2) Temporal Awareness: Round-based multipliers dynamically prioritize card types:

TABLE I
ROUND-BASED CARD TYPE MULTIPLIERS

Card Type	R0	R1	R2
Tempura/Sashimi/Dumpling/Nigiri	1.0	1.0	1.0
Maki	1.0	1.0	1.2
Pudding	0.6	0.8	**1.5**
Wasabi	1.3	1.0	0.9
Chopsticks	1.4	1.0	0.7

This recognizes pudding's critical late-game value (1.5×) and wasabi/chopsticks' early-game utility (1.3-1.4×).

3) Opponent Modeling Strategy: Two weighted components incorporate opponent information:

1. Opponent Played Cards: Dynamic weighting increases with game progress:

$$w_{\text{played}} = 0.15 + (\text{round} \times 0.05) \quad (2)$$

2. Opponent Hand Evaluation: Applied only for known hands (passed hands):

$$w_{\text{hand}} = 0.1 + \min(\text{knownHands} \times 0.025, 0.05) \quad (3)$$

Conservative weighting (0.1-0.15) avoids over-committing to uncertain predictions. The `isHandKnown()` function ensures evaluation of observed hands only.

4) Parameter Optimization: We modified exploration constant K for agent and $raveK$ and heuristic weight for UCB1, UCB1-Tuned and RAVE with progressive bias:

$$\text{UCB1-Tuned-PB} = \frac{w_i}{n_i} + K\sqrt{\frac{\ln N}{n_j}\min\left(\frac{1}{4}, V_j(n_j)\right)} + \frac{h_i}{1+n_i} \quad (4)$$

Testing K from 0.5 to 1.4, we found K=0.7 optimal, reducing exploration when strong heuristics provide reliable guidance.

B. Experimental Design

We conducted 23 experiments in five groups using systematic incremental testing.

1) Tournament Configuration: Consistent 3-player tournaments: Players 1-2 use Basic MCTS (K=1.0, 200 iterations), Player 3 uses Custom MCTS (variable). Each configuration tested over 1000 games.

2) Experimental Groups: **Group 1 (Exp 1-7)** identified optimal configuration by incrementally adding components from K-tuned baseline (K=1.4): (1) played cards, (2) hand evaluation, (3) weight adjustments, (4) min/max functions, (5) late-game pudding focus, (6) opponent scoring, determining which components provide gains, when additions hurt performance, and whether components synergize or interfere. **Group 2 (Exp 8-11)** validated saturation by testing beyond peak performance, adding a 7th component (round-based weighting) then systematically removing components to quantify oversaturation loss, identify interfering components, and test performance restoration reproducibility. **Group 3 (Exp 12-17)** tested opponent modeling weights and strategies using the 5-component base with varied opponent hand weighting (0.3-0.6) and specialized strategies to determine if opponent modeling helps, whether weighting matters more than information type, and what balance between self-optimization and opponent awareness is optimal. **Group 4 (Exp 18-21)** determined optimal exploration-exploitation balance by testing K-values 0.5-1.4 with the 5-component heuristic, assessing whether strong heuristics need less exploration, finding optimal K, and evaluating robustness across values. **Group 5 (Exp 22-23)** tested novel enhancements and validated final configuration through heuristic-guided rollouts (Exp 22: 70% heuristic/30% random) and final optimization (Exp 23: 5 components + K=0.7 + light opponent modeling), determining if heuristic rollouts are cost-effective, whether the final configuration achieves peak performance, and if light opponent modeling complements strong heuristics.

3) Evaluation Metrics: Configurations evaluated on win rate (primary), average score (secondary), statistical significance (±0.015-0.016), and computational efficiency. This approach isolates design decisions' effects, quantifies component interference, and identifies optimal configuration empirically.

IV. Results

A. Experimental Results

We present 23 experiments across five groups, testing heuristic configurations in 3-player tournaments (1000 games each).

1) Group 1: Cumulative Component Building: Table II shows systematic component addition.

TABLE II
COMPONENT ADDITION PATTERN

Exp	Components Added	Comp.	Win	Score
1	Base (K=1.4)	0	0.34±0.015	45.51
2	+ Played Cards	1	0.40±0.015	47.07
3	+ Hand Eval	2	**0.51±0.016**	48.41
4	+ Weights	3	0.47±0.016	47.57
5	+ Min/Max	4	0.52±0.015	48.67
6	+ Late Pudding	5	**0.53±0.016**	48.18
7	+ Opponent	6	0.47±0.016	47.14

Note: "Comp." shows cumulative active components.

Performance increases from 0.34 to 0.53, then declines to 0.47 with opponent scoring added. Hand evaluation (Exp 3) provides the largest gain.

2) Group 2: Component Saturation Testing: Table III examines performance beyond saturation.

TABLE III
COMPONENT SATURATION TESTING

Exp	Configuration	Comp.	Win	Score
8	Base(6) + Round Weight	7	0.44±0.016	46.94
9	Base(5) + Round (no Opp)	6	0.45±0.016	47.37
10-11	Base(5) + Round Weight Variations	6	0.48-0.52	47-48

A 7th component drops performance to 0.44 . Removing opponent scoring and round weighting (Exp 10-11) restores performance to 0.48-0.52.

3) Group 3: Opponent Modeling Variations: Table IV tests opponent modeling strategies.

TABLE IV
OPPONENT MODELING VARIATIONS

Exp	Strategy	Win	Score
12-14	Opp. Hand (w=0.3-0.6)	0.42-0.52±0.015	46-48.39
15	Opp. Score Weight	0.47±0.016	47.1
16	Pudding Specialized	0.48±0.016	47.99
17	Opp. Played Cards	0.45±0.016	47.16

Note: "Base" = 5-component optimal (Exp 6: 0.53).

Opponent hand scoring with weight 0.6 (Exp 14) achieves 0.52, nearly matching peak. Light opponent modeling proves effective when properly weighted.

4) Group 4: K-Value Parameter Optimization: Table V shows exploration parameter sensitivity.

TABLE V
K-VALUE SENSITIVITY ANALYSIS

Exp	K	Exploration	Win	Score
18	1.1	Moderate	0.48±0.016	47.81
19	0.9	Conservative	0.51±0.016	48.61
20	**0.7**	**Low**	**0.52±0.016**	**48.07**
21	0.5	Minimal	0.50±0.016	48.35

Reducing K from 1.4 to 0.7 improves performance to 0.52. Below K=0.5, performance starts reducing.

5) Group 5: Advanced Implementation Tests: Table VI presents novel techniques and final configuration.

TABLE VI
ADVANCED TECHNIQUES AND FINAL MODEL

Exp	Configuration	K	Win	Score
22	Rollout Heuristic	0.7	0.48±0.016	48.04
23	**Final Model**	0.7	**0.52±0.016**	**47.85**

Note: Both use 5-component optimal (Exp 6) + K=0.7 (Exp 20).

Heuristic-guided rollouts achieve 0.48 but cost 2-3x more computationally. Final model (Exp 23) combines 5 components, K=0.7, and light opponent modeling (weight 0.3) for 0.52 win rate.

TABLE VII
PERFORMANCE TIER CLASSIFICATION

Tier	Win Rate	# Exp	Representative
Peak	0.51-0.53	5	3, 6, 14, 20, 23
Excellent	0.48-0.50	6	5, 10-11, 19, 21-22
Good	0.45-0.47	6	4, 7, 9, 15-17
Baseline	0.40-0.44	6	1-2, 8, 12-13, 18

6) Performance Tier Classification:

B. Analysis and Interpretation

1) Component Saturation Principle: Performance increases with components (0.34 $\rightarrow$ 0.53, Exp 1-6) but degrades beyond 5 (0.53 $\rightarrow$ 0.44, Exp 6-8) with variance between ±0.015-0.016

Multiple evaluation strategies create conflicting signals confusing MCTS tree search. Opponent scoring (component 6) and round weighting (component 7) introduce prediction uncertainty that outweighs informational benefits, causing suboptimal move selection.

2) Opponent Modeling Nuance: Results show opponent modeling effectiveness depends on weighting:

- **Light weighting (0.3-0.6):** Exp 14, 23 achieve 0.52, matching peak with ±0.015-0.016
- **Heavy weighting (>0.7):** Poor performance (0.42-0.47 ±0.015)

Light opponent information *complements* strong self-optimization heuristics rather than replacing them. Weight 0.3-0.6 acknowledges opponent presence without over-committing to uncertain predictions.

3) Parameter-Heuristic Interaction: K-value optimization shows strong heuristics enable reduced exploration: K=0.7 achieves 0.52 vs 0.48 at K=1.1. Strong domain knowledge provides reliable evaluation, reducing random exploration needs. The agent exploits known strategies rather than exploring alternatives, improving both performance and efficiency.

4) Rollout Heuristics Evaluation: Experiment 22 shows heuristic rollouts are viable but not beneficial: 2-3× slower with worse performance than tree-level heuristics.

5) Final Performance: Our optimized agent (Exp 23) achieves 0.52 $\pm$ 0.016 win rate with 95% CI: [0.504, 0.536] using Wilson score intervals for binomial data ($n = 1000$ games). We are 95% confident the true win rate lies between 50.4% and 53.6%, representing 56% improvement over random baseline (0.333 expected in 3-player games). The non-overlapping confidence intervals confirm this improvement is statistically significant ($p < 0.001$).

V. CONCLUSIONS

A. Critical Summary

This research systematically investigated MCTS heuristic design for Sushi Go through 23 experimental configurations, establishing three fundamental principles. First, the component saturation principle: performance peaks at an optimal number of heuristic components before degrading with over-saturation, demonstrating more complexity does not guarantee better performance. Second, opponent modeling effectiveness depends critically on weighting—light incorporation complements self-optimization achieving competitive performance, while heavy focus fails due to prediction uncertainty in imperfect information games. Finally strong heuristics enable exploration parameter reduction, improving both performance and computational efficiency. Our systematic methodology successfully identified optimal component selection, parameter tuning, and opponent modeling strategy, achieving dominant tournament performance with statistically significant improvements over baseline approaches.

B. Limitations and Future Directions

Heuristic-guided rollouts proved computationally expensive without performance benefits, suggesting future work should investigate more efficient implementations using lightweight evaluation functions. Static exploration parameters could benefit from dynamic adjustment mechanisms based on game phase or uncertainty levels, potentially using reinforcement learning. Opponent prediction remains limited even with optimal weighting, motivating future research into multi-opponent modeling that differentiates opponents based on observed playing styles. This research provides validated methodology and quantified guidelines for MCTS heuristic design in card games, establishing foundations for future AI development in imperfect information domains.

REFERENCES

[1] C. B. Browne *et al.*, "A survey of monte carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1-43, 2012.

[2] Gamewright, *Sushi Go! Rulebook*, 2013. [Online]. Available: http://cdn.1j1ju.com/medias/e5/92/74-sushi-go-rulebook.pdf [Accessed: Oct. 29, 2025].

[3] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Computers and Games (CG 2006)*, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., ser. Lecture Notes in Computer Science, vol. 4630. Berlin, Heidelberg: Springer, 2007. doi: 10.1007/978-3-540-75538-8_7

[4] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. doi: 10.1038/nature16961.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (4th ed.)*. Hoboken, NJ: Pearson, 2020.

[6] R. J. Lorentz, "Improving Monte–Carlo tree search in Havannah," in *Computers and Games (CG 2010)*, H. J. van den Herik, H. Iida, and A. Plaat, Eds., ser. Lecture Notes in Computer Science, vol. 6515. Berlin, Heidelberg: Springer, 2011. doi: 10.1007/978-3-642-17928-0_10

[7] H. Finnsson and Y. Björnsson, "Learning simulation control in general game-playing agents," in *Proc. 24th AAAI Conf. Artificial Intelligence (AAAI 2010)*, Atlanta, GA, USA, Jul. 11–15, 2010.

[8] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set Monte Carlo tree search," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 120-143, 2012. doi: 10.1109/TCIAIG.2012.2200894

[9] G. M. J-B. Chaslot, M. H. M. Winands, H. J. van der Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 343–357, 2008. doi: 10.1142/s1793005708001094

[10] B. Sac, D. G. Atilla, and H. K. Prajapati, "Experimental data: MCTS heuristic design for Sushi Go!," Google Sheets, 2025. [Online]. Available: https://docs.google.com/spreadsheets/d/1yNXryg3YCy178fptvJnbpGUn3Obq5cx3dvEZCVpksvo. [Accessed: Nov. 3, 2025].

VI. AI USAGE DECLARATION

AI assistance (Claude 3.5 Sonnet via GitHub Copilot) was used for:

- **Report writing:** Improving clarity, structure, and academic tone of technical descriptions
- **Content optimization:** Condensing sections to meet 4-page limit while preserving technical accuracy
- **Code debugging:** Identifying and resolving implementation errors
- **Heuristic development:** Getting ideas and validating approaches for heuristic design