Fingerprinting Tabletop Games

James Goodman
Game AI Research Group
Queen Mary University of London
james.goodman@qmul.ac.uk

Diego Perez-Liebana
Game AI Research Group
Queen Mary University of London
diego.perez@qmul.ac.uk

Simon Lucas
Game AI Research Group
Queen Mary University of London
simon.lucas@qmul.ac.uk

***Abstract*—We present some initial work on characterizing games using a visual 'fingerprint' generated from several independent optimisation runs over the parameters used in Monte Carlo Tree Search (MCTS). This 'fingerprint' provides a useful tool to compare games, as well as highlighting the relative sensitivity of a specific game to algorithmic variants of MCTS. The exploratory work presented here shows that in some games there is a major change in the optimal MCTS parameters when we move from 2-players to 3 or 4-players.**

***Index Terms*—Optimization, MCTS**

I. Introduction

There are many approaches to defining a set of useful characteristics to compare games, from quantitative measurement of branching factors and states in a game tree, to qualitative assessments of positional asymmetry, sensory feedback and accessibility to spectators [1]. Here we present some initial work on using the results of optimisation of Monte Carlo Tree Search (MCTS) parameters to characterise a set of multi-player tabletop games in the TAG framework [2].

The central idea is to run multiple independent optimisations of MCTS parameters over a large search-space, and use the marginal distribution over all optimisations of each parameter as a means of characterising the game. This develops from two common practices; one of reporting performance for parameter settings across multiple games to show which games benefit from the specific algorithmic variant being investigated; a second of reporting a single set of optimised parameters for a game. The approach generates a more 'distributional' output visualised as bar charts that can be compared easily by the human eye. It makes clear the relative importance of a parameter to performance in a game in a way that reporting a single optimal setting does not. If a setting is critical to performance then there will be a sharp peak in the marginal distribution as all optimization runs settle on that value; if a setting is superfluous then the distribution will be flat as each optimization run picks a value effectively at random.

The results are still undergoing detailed analysis, but one immediately clear conclusion is that the fingerprint and best parameter settings of a single game can sometimes change radically as the number of players varies.

II. Background

A. Tabletop Games Framework (TAG)

TAG is a recently developed framework for the implementation of modern Euro-style board and card-games [2]. These games are of interest for research both because of their high popularity, and because they often have high levels of hidden information, stochasticity and interaction with more than two players [3]. They contrast with 2-player, perfect information classical games such as Chess or Go, which are well covered in the Ludii project [4]. The games used are listed below, for more detailed summaries see the TAG wiki[1]:

- Tic Tac Toe (c. 1850)
- Dots and Boxes (1889)
- Uno (1971)
- Diamant (2005)
- Dominion (2008)
- Love Letter (2012)
- Colt Express (2014)
- Virus (2015)
- Exploding Kittens (2015)

B. MCTS

Monte Carlo Tree Search (MCTS) [5]–[7] has been used in many games. It is an anytime algorithm that uses a time budget to search the forward game tree in four steps:

1) Selection. Select an action to take from the current state. If all actions have been selected at least once then the best one is picked using the Upper Confidence for Trees equation [8]: $J(a) = Q(a) + K\sqrt{\frac{\log(N)}{n(a)}}$ The action a with largest $J(a)$ is selected. N is the total number of visits to (iterations through) the state; $n(a)$ is the number of those visits that then took action a; $Q(a)$ is the mean score for all visits to the state that took action a; K controls the trade-off between exploitation (choosing high-valued actions) and exploration (actions with few visits). This step is repeated down the tree of game states until a state is reached with previously untried actions.
2) Expansion. Pick one of the untried actions uniformly at random and add a new child state to the game tree.
3) Rollout. From the expanded state, take actions uniformly at random for a number of steps (or the end of the game) to obtain a final score.

[1]https://github.com/GAIGResearch/TabletopGames/wiki/Implemented-Games

Parameter	Values	Description
Tree Policy	{UCB Alpha EXP3 RM}	Standard UCT formula The AlphaGo formula Softmax Selection Regret Matching UCT
Opponent Tree Policy	{MaxN Paranoid SelfOnly}	Maximise their score Minimise our score Act randomly
Final Policy	{Robust Simple}	Most visited action Most valuable action
TreeDepth	{1, 3, 10, 30, 100}	Max Depth of Tree
Rollout	{0, 3, 10, 30, 100}	Max actions per rollout
Redeterminise	{false, true}	Information Set MCTS
Open Loop	{false, true}	Use Forward model in tree
K	{0.01, 0.1, 1, 10, 100}	Exploration coefficient for UCB and Alpha
explore ϵ	{0.01, 0.03, 0.1, 0.3}	Exploration chance for EXP3 and Regret Matching

TABLE I: MCTS Parameters included in each Optimization run and fingerprint. See main text for details on each.

4) Back-propagation. Propagate this score up the nodes visited in this iteration. Each state records the mean score of all iterations that take a given action from that state ($Q(a)$), which will affect future Selection steps. Once the time budget has been used the action at the root state with either the highest score or most visits is executed.

The TAG games are multi-player games so we include a number of variations to the core MCTS algorithm that may help in this environment. The nine parameters we configure to generate a fingerprint are summarized in Table I.

- Tree Policy. As well as the classic selection policy of the UCT algorithm [8] we consider the variant used by [9], which modifies the exploration term to $\sqrt{N/(1+n(a))}$. We also consider the EXP3 and Regret Matching (RM) UCT policies described in [10] that have better theoretical behaviour in adversarial environments.
- Opponent Tree Policy. We consider three variants to model the opponent in the tree. 'MaxN' assumes that each other player maximises their score and uses a single tree to model all players; at each node the acting player's score is used to make a decision. This is the Multiplayer-UCT of [11]. 'Paranoid' adapts this with all other players modeled as reducing our score (in a 2-player game with a Win/Lose reward this is identical to MaxN). 'SelfOnly' just models the agent's own actions in the tree; it assumes that the actions of other players do not matter, and makes random decisions for them to advance the game state.
- Final Policy. This controls how a decision is made at the root node after the search process is completed. 'Robust' picks the action with the most visits. 'Simple' picks the action with the highest average value.
- Max Tree Depth. The maximum depth to which the tree will be constructed before a rollout starts.
- Rollout Length. The maximum length of a rollout after the tree phase. This many actions will be taken randomly before the final state is evaluated (or the game ends).
- Redeterminise. If `true` then Information Set MCTS is used [12]. This redeterminises the game state at the start of each iteration of MCTS, reshuffling across all possible sets of the hidden information. If `false` then this defaults to Perfect Information MCTS, with a random (valid) shuffling of the hidden information used for all MCTS iterations. In the case of a genuinely perfect information game, such as Dots and Boxes in our sample, both settings should give identical behaviour apart from any performance hit of redeterminisation.
- Open Loop. When `true` the forward model advances the game state from the root through the tree on every iteration. This allows the underlying state to be different due on each node visit, due to a stochastic environment or actions of other players outside in the tree [13]. When `false` the forward model is not applied in the tree, and each leaf stores the state that caused its expansion. This approach reduces the number of forward model calls but will perform less well if the game is not deterministic.
- K. The exploration weighting in the UCT formula.
- ϵ. K is not used in EXP3 and RM selection tree policies. Instead a random action is taken with probability ϵ

C. NTBEA

The N-Tuple Bandit Evolutionary Algorithm (NTBEA) is described in [14]. It has been benchmarked against other optimisation algorithms in stochastic game environments and proven to be more effective at finding a good set of parameter settings than other algorithms within a fixed computational budget [15]. Similarly [16] find NTBEA is the best optimiser of MCTS parameters during algorithm execution for a number of games. Given a parameter setting $\boldsymbol{\theta}$, NTBEA uses an N-Tuple model to predict $f(\boldsymbol{\theta})$, the expected score if a game is played using $\boldsymbol{\theta}$. In each iteration of NTBEA we:

1) Run one game using the current test setting $\boldsymbol{\theta}_t$. $\boldsymbol{\theta}_0$ is selected at random.
2) Update the N-Tuple Model with the evaluation result [14].
3) Generate X points by applying a mutation operator to $\boldsymbol{\theta}_t$.
4) Evaluate the Upper Confidence Bound (UCB) for each of the X points using the N-Tuple Model. Select the one with the highest UCB as $\boldsymbol{\theta}_{t+1}$, and repeat from 1.

In this study we set X=50, and the mutation operator randomly mutates each θ_i to a random setting with probability $\frac{1}{d}$.

After running a specified number of iterations, NTBEA returns the $\boldsymbol{\theta}$ with the best predicted score according to the model. Previous work has shown that an NTBEA run is effective at finding a good $\boldsymbol{\theta}$ in a small number of trials with a noisy utility function, but any individual NTBEA run may get stuck in a local optimum [17]. This analysis recommends splitting the computational budget into a number of separate runs, and then using the single best result.

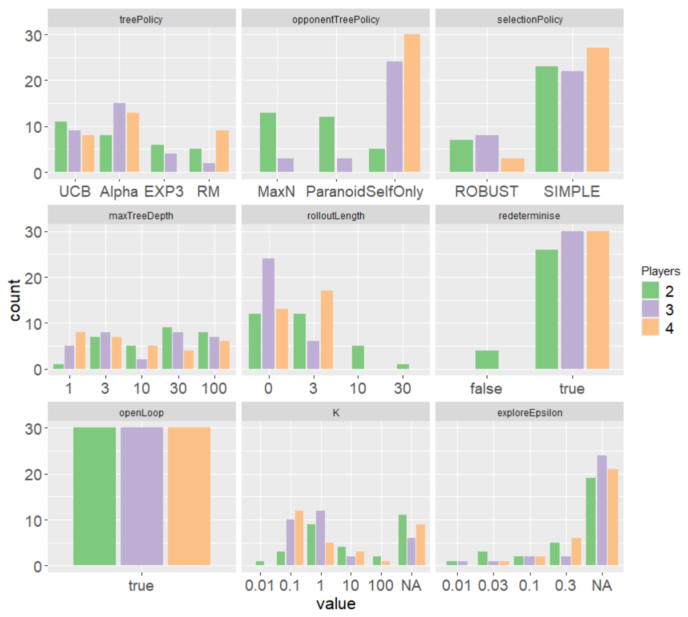

Fig. 1: Fingerprint for Love Letter against OSLA opponents. Each box shows the marginal distribution of one parameter over 30 NTBEA runs. Every run uses open loop, and all 4-player runs recommend 'SelfOnly' as an opponent tree policy.

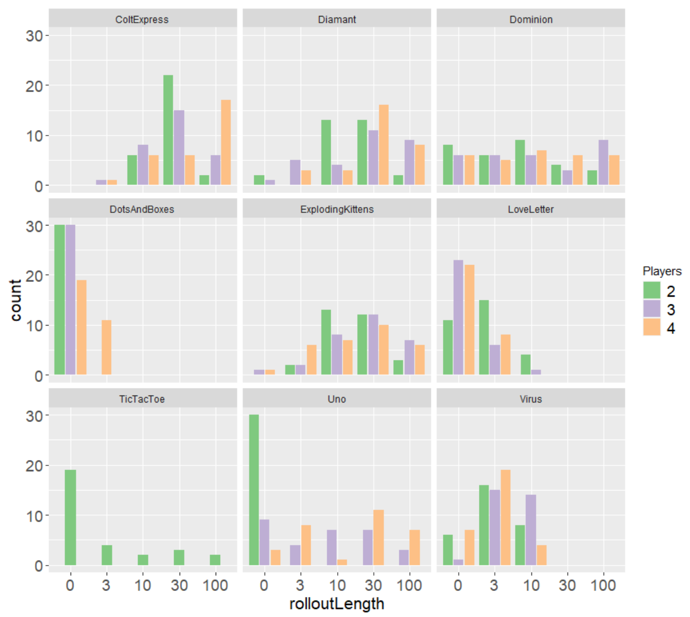

Fig. 2: Fingerprint of Rollout length for all games and player counts against simple MCTS opponents.

III. Method

We optimise MCTS agents to win against a fixed opponent with a 40ms budget per decision. We repeat this for three different opponents: random (RND), a one-step lookahead (OSLA), and a simple MCTS agent with a 20ms time budget. We use wall-time instead of the number of iterations or forward model calls to avoid tilting the field towards short-rollouts (favoured if we budget on forward model calls), or towards open loop MCTS and long rollouts and tree-depth (favoured if we budget on iterations, allowing any number of forward model calls on each). Using a wall-time budget means the optimisation process will automatically trade-off the benefits of each under a common constraint. Experiments are run on a 2.6 GHz Intel Xeon Gold 6240 CPU.

For each game we run 30 independent NTBEA runs, each with 4,500 trials. The total search-space across the nine parameters defined in Table I has 48,000 possible settings, so only a small fraction of these will be tried in any run. When optimising an N-player game, a single MCTS player using the optimising parameters is placed at a random position in each trial, and the remaining N-1 players use the same fixed opponent (RND, OSLA or MCTS). For each of the 73 combinations of game, player count and opponent this gives 30 independently optimized parameters. We reduce this to a fingerprint of nine marginal distributions, one per parameter, displayed as a grid of barcharts, as shown in Figure 1.

IV. Results

For reasons of space we show just a sample of the full results here. The full results for all games, player counts and opponents can be browsed at https://gaigresearch.github.io/TabletopGames/site/GameFingerprints/NTBEAFingerprints.html.

Figure 1 shows the fingerprint of Love Letter against an OSLA opponent for 2-4 players. This is interesting because it shows a major change as we move to 3/4 players, for which it is much more advantageous to allocate computational budget to exploring ones own options while ignoring what the other players might so in response (opponentTree=SelfOnly). This pattern is also visible when facing a random or MCTS opponent. Love Letter is a role-deduction game, in which the hidden information is the role-card that each player has, and the strong tilt towards open loop and information set MCTS (redeterminisation=true) is hence as expected.

Figure 2 then shows the different patterns of a single MCTS parameter across all games and player counts (in this case against the simple MCTS opponent). We see that Dots and Boxes favours short (or no) rollouts; and Colt Express favours long rollouts. Colt Express is a 'programming' game, with players first taking turns to play cards that will only later be executed in order. This means that there is often a long gap between a move and any reward from that more, and in this situation longer rollouts are needed to ensure this reward is back-propagated. Dots and Boxes in contrast has a much more rapid feedback with a point gained as soon as a player encloses a box, and long rollouts are not necessary. The optimisation results show that here it is better to truncate rollouts and use the available computational budget to run more iterations.

Like Love Letter in Figure 1, 2-player Uno is startlingly different to Uno with more players, and this is a pattern evident in Uno across other parameters too. Of all the games used, Uno has the largest change as we move from 2 to 3 players.

Figure 3 shows all nine games with 2-players and a random

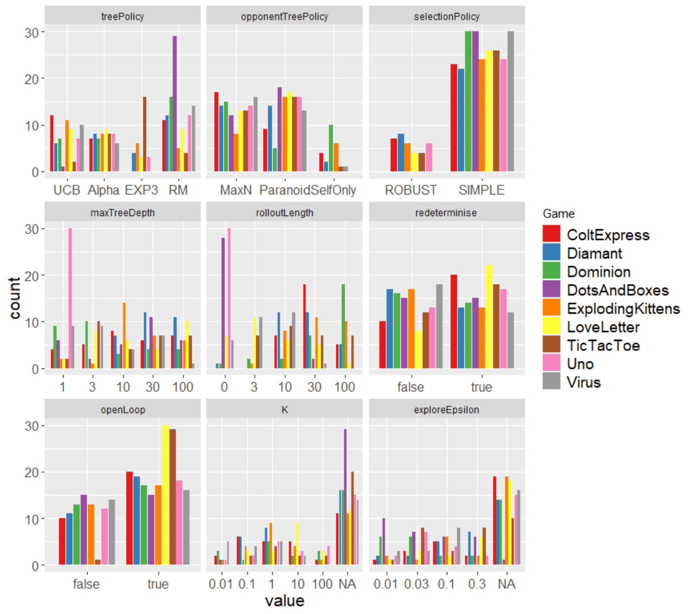

Fig. 3: All parameters for all 2-player games against random opponents.

opponent. A general observation is that using a Simple final action selection policy seems a better general choice than the more usual Robust method. Otherwise this slicing of the data highlights a few outliers; in particular Dots and Boxes strongly favours a Regret Matching tree policy, while it and Uno are the only games that do well with a zero rollout length. Dots and Boxes is the only perfect information game, and the 2-player version is a direct zero-sum adversarial contest for which Regret Matching has theoretical benefits.

V. Discussion

This paper does not exhaustively analyse the results, but picks out some interesting examples. This is work-in-progress, and we need to evaluate how useful these game characterizations are in applications. Can these fingerprints usefully predict external game attributes (in terms of strategic depth for example), or as the basis for an ensemble AI agent? The method does not highlight interactions between parameters given the 1-dimensional marginalisation, and 73 different game settings (giving 657 combinations across 9 parameters) means we must be careful of asserting statistical significance of any one result. Despite this caveat the Love Letter shift to a 'Self-Only' tree at 4-players in Figure 1, or the shift to zero rollout length for 2-player Uno in Figure 2 are robust under any correction for multiple tests.

We also need to check robustness to varying the optimisation approach, and computational budget. Using 40ms for each decision allows 300-800 MCTS iterations per move, depending on the game. We would expect the results to change as this is increased, and investigating the relative sensitivity of different games to computational budgets is future work.

VI. Conclusions

We have introduced a method for graphically 'fingerprinting' games from multiple independent optimisations of algorithmic parameters (here, MCTS), and applied this to nine games for player counts between 2 and 4. This has shown some cases where the fingerprint of a game changes quite dramatically as we change the number of players, and also in some cases where the opponent changes. Future work is to extend this method to other algorithms, computational budgets, and games, and construct a landscape of multi-player games.

Acknowledgment

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/S022325/1. Thanks are due to Martin Balla, Alex Dockhorn, Raluca Gaina and Raul Montoliu for contributing games to TAG.

References

[1] G. S. Elias, R. Garfield, and K. R. Gutschera, *Characteristics of games*. MIT Press, 2012.

[2] R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, "TAG: A tabletop games framework," in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2020.

[3] S. Woods, *Eurogames: The design, culture and play of modern European board games*. McFarland, 2012.

[4] M. Stephenson, E. Piette, D. J. Soemers, and C. Browne, "An overview of the ludii general game system," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–2.

[5] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*. Springer, 2006, p. 72–83.

[6] G. Chaslot, S. De Jong, J.-T. Saito, and J. Uiterwijk, "Monte-carlo tree search in production management problems," in *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, 2006, p. 91–98.

[7] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, p. 1–43, Mar 2012.

[8] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, p. 282–293.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, and et al., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354–359, Oct 2017.

[10] V. Lisy, "Alternative selection functions for information set monte carlo tree search," *Acta Polytechnica*, vol. 54, no. 5, p. 333–340, 2014.

[11] N. Sturtevant, "An analysis of UCT in multi-player games," *ICGA Journal*, p. 14, 2008.

[12] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, p. 120–143, Jun 2012.

[13] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open loop search for general video game playing," in *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*. ACM Press, 2015, p. 337–344. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2739480.2754811

[14] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The n-tuple bandit evolutionary algorithm for game agent optimisation," *arXiv:1802.05991 [cs]*, Feb 2018, arXiv: 1802.05991. [Online]. Available: http://arxiv.org/abs/1802.05991

[15] S. Lucas, J. Liu, I. Bravi, R. Gaina, J. Woodward, V. Volz, and D. Perez-Liebana, "Efficient evolutionary methods for game agent optimisation: Model-based is best," *arXiv preprint arXiv:1901.00723*, 2019.

[16] C. F. Sironi and M. H. Winands, "Comparing randomization strategies for search-control parameters in monte-carlo tree search," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, p. 1–8.

[17] J. Goodman and S. Lucas, "Weighting NTBEA for game AI optimisation," *arXiv preprint arXiv:2003.10378*, 2020.