MultiTree MCTS in 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 introduce MultiTree Monte Carlo Tree Search (MT-MCTS), in which a tree is constructed independently for each player. This permits deeper search for the acting agent's own move, at the cost of a poorer opponent model and the loss of conditioning a move on the specific action of another player.**

We test MT-MCTS in eleven different tabletop board and card games, with varying numbers of players. The main benefit occurs in simultaneous-move games, where independent trees better model the information structure. We find that in other games MT-MCTS can outperform vanilla MCTS, which incorporates all players in a single tree, but that this advantage usually decreases as the computational budget increases, and the cost of poor opponent modelling outweighs the gain from deeper search.

***Index Terms*—MCTS, Multiplayer, Tabletop Games, Opponent Modelling**

I. Introduction

Modern tabletop games have different characteristics to classic AI test-beds such as Chess, Go, or Checkers. Instead of being 2-player zero-sum games of perfect information, they are N-player (for $N \geq 2$) games, often with imperfect information. Formally they are zero-sum as only one player can win, but there are elements of general-sum nature in that players usually have a score, which many players will seek to maximise as a secondary objective to winning, and it is often possible for players to gain points without taking them from others. Further, some of these games, especially those that fall into the genre of 'Eurogames' [1], are less directly adversarial, in that the action one player takes has little direct impact on the position of others. We use the term 'adversarial' for games in which the action of one player has a direct impact on the position of another, either in material held or availability of moves. For example in the game of Dominion, each player buys new cards to build their private deck, which affects their options on later turns. A few of these options can directly attack opponents but most are 'internal' to the player's own position, as they build an engine used to buy victory point cards later and are not 'adversarial' in our sense. The game usually ends when the last 'Province' victory point card is bought, and the game can be seen as a race between the players to get their engine firing as fast as possible. In this respect, eurogames can have single-player aspects [2].

In perfect-information 2-player games, minimax search, or Monte Carlo Tree Search have been very successful [3]–[6]. These construct a game-tree from the current state to decide on the best move to take, either by full enumeration (with pruning), or by stochastic sampling. A common factor is that opponent nodes are included in the tree, with the assumption that in a zero-sum environment opponents act to minimise our score or win chance.

The insight we build on is that the inclusion of every opponent move in the tree gives a large branching factor and a shallow search, with many of the leaves identical with respect to the moves of the acting agent. In the context of the lower adversariality of some eurogames this may be counter-productive. This is not a new insight, and is behind several techniques in the literature, such as Best Response Search (BRS), Opponent Move Abstraction (OMA) and Opponent Pruning Paranoid Search (OPPS), to name three that are closest to this work [7]–[9].

We consider the impact of constructing separate trees for each player in the game during MCTS, which we term Multi-Tree MCTS (MT-MCTS). Instead of a single tree containing nodes for all player actions, an independent tree is constructed for each player to model their actions. Each MCTS iteration adds a new node to each tree using a single trajectory, with decisions made by a player using the information in their tree. This increases the depth of each tree by avoiding the combinatorial explosion at each ply as each opponent action is explored. If there are A actions at each node and a search depth of d, then the nodes in the tree, $N = O(A^d)$. If k players have equal numbers of turns, then $d = k\alpha$, where α is the depth of the tree in terms of self-moves; and $\alpha = O(\frac{1}{k})$.

This deeper search comes at the cost of an inability to make move-conditional responses. If Y is a good response to A, but not for actions B and C, then the opponent tree will not learn to take action Y after A as this decision is not conditioned on A being the previous action. Instead, the opponent model uses statistics averaged over trajectories including all A, B and C.

We hypothesize that in modern tabletop games the benefit of increased planning depth may outweigh the costs of not detecting such move-specific adversarial responses. This benefit is expected to increase as the player count increases, as this proportionally reduces the search depth in self-actions for vanilla MCTS. We test this using a set of 11 games implemented in the Tabletop Games Framework (TAG, www.tabletopgames.ai) for varying numbers of players.

We find that there are some games in which MT-MCTS does perform well, but that this not universal across all 11 games. In some games an extreme version of MCTS in which the moves of the opponent are ignored completely within tree search can perform best.

II. Background

A. MCTS

Monte Carlo Tree Search (MCTS) [5], [10], [11] has been used in many games. It searches the forward game tree by sampling. On each iteration four steps are followed:

1) Selection. Select an action to take from the current node. If all actions have been selected at least once then the best one is picked using the Upper Confidence for Trees equation [12]: $J(a) = Q(a) + K\sqrt{\frac{\log(N)}{n(a)}}$s The action a with largest $J(a)$ is selected. N is the total number of visits to the node; $n(a)$ is the number of those visits that took action a; $Q(a)$ is the mean score for all visits to the node that took action a; K controls the trade-off between exploitation, and exploration choosing actions with few visits so far. This step is repeated down the tree until a node is reached with previously untried actions.
2) Expansion. Pick one of the untried actions (using an expansion policy, which may be random), and expand this, creating a new node in the game tree.
3) Rollout. From the expanded node, take actions using a rollout policy (which may be random) for a number of steps (or the end of the game) to obtain a final score.
4) Back-propagation. Back-propagate this final score up the tree through all nodes visited this iteration. Each node records the mean score of all iterations that take a given action from that node as $Q(a)$ that will affect future Selection steps. Once the time budget has been used the action at the root node with either the highest score or most visits is executed.

B. Tabletop Games Framework (TAG)

TAG is a framework for the implementation of modern Euro-style board and card-games [13]. These games are of research interest both because of their high popularity, and because they often have high levels of hidden information, stochasticity and support more than two players. The 11 games used are listed below, for more detailed summaries see https://boardgamegeek.com/:

- TicTacToe. The trivial perfect information 2-player game used as an illustrative baseline.
- Poker (1810). Uses Texas Hold'em rules.
- Dots and Boxes (1889). A perfect information game in which players take turns to connect adjacent dots.
- Uno (1971). Players match suits and numbers to discard all cards from their hand before their opponents.
- Love Letter (2012). A game of role deduction. Players target opponents to gain information or knock them out.
- Diamant (2005). A simultaneous-move push-your-luck game. Players decide whether to continue exploring a mine. Treasure can be found, or the mine may collapse.
- Dominion (2008). A Deck-building card game in which a player first needs to build an 'engine', and then use this to gain victory point cards.
- Sushi GO! (2013). Simultaneous-move set collection.
- Colt Express (2014). Players plan a partially observable sequence of actions to rob a train.
- Virus (2015). Players play cards from their hand to construct a healthy body of four organs (cards), and use Virus cards to infect the organs of other players.
- Exploding Kittens (2015). Players are knocked out if they draw an Exploding Kitten. Cards can be played to peek at and manipulate cards in the draw deck.

III. Previous Work

Constructing a tree per player to determine their likely move is a form of opponent modelling. The key works we build on are covered below; excellent recent surveys of opponent modelling techniques are [14], [15].

A. Reduced Opponent Computation

Classic search algorithms, be they exhaustive (e.g. minimax) or sampled (e.g. MCTS), treat all players equally in the tree. For N players, assuming each moves in sequence, there are $N-1$ plies in the tree that consider opponent moves before returning to the acting agent. For large branching factors the tree does not reliably reach the agent's next move and spends the computational budget on the intervening opponent moves. Schadd et al. [7] took this insight in three multiplayer perfect information games to develop Best Response Search (BRS). Given a move ordering function this collapses all the intervening opponent moves to the single best opponent move, with the others taking no action. In BRS every odd ply is a self-move, and every even ply an opponent-move, converting the game into a 2-player zero sum game, which increases the search depth and allows $\alpha\beta$-pruning to be used.

BRS was found to beat Max^N [16] and Paranoid [17] search approaches for $N > 2$, and later extended to BRS+, in which the other opponents take random actions to ensure that the game state remains legal during search [18]. Max^N and Paranoid both construct a single tree with nodes for all players. In Max^N statistics at each node use the back-propagated value specific to the player at that node, so that other players are modelled as maximising their own win probability. Paranoid instead assumes that other players aim to minimise our score.

OPPS generalises BRS+ with three parameters; n, the number of opponents who are given an action space of size *l1* in the tree, with the remainder given a (smaller) action space of size *l2* [9]. As with BRS, a move ordering function is required to select the specific *l1, l2* actions.

Opponent Move Abstraction (OMA) [8] uses the same insight, that it may be beneficial to focus more of the computational budget on self-moves than opponent responses, in MCTS. OMA constructs a full tree containing all players, but aggregates statistics for self-moves across nodes which have the same sequence of self-actions from the root. This set of aggregated statistics is then interpolated with the single node statistics (i.e. just for the observed sequence of self- and opponent-actions from the root), so that asymptotically only the full tree statistics are used. OMA does not require a move ordering function.

MT-MCTS differs from OMA by constructing a single tree per player, only using the aggregated statistics, and in also applying these aggregated statistics to opponent decisions.

All of the above algorithms have been tested in perfect information environments with sequential turn-taking by players. Two of the games used here are perfect information (TicTacToe; Dots and Boxes). The rest are imperfect information with large amounts of hidden information; usually the draw decks and opponent hands. For most of these (see Section V) we use Information Set MCTS (IS-MCTS) with redeterminisation (shuffling) of unknown information at the root on each iteration, and each node corresponding to an information set from the perspective of the acting player [19]. Formally IS-MCTS has no guarantees of converging to an optimal (Nash Equilibrium) strategy due to the leakage of information known to the root player into the implicit opponent models at opponent decisions further down the tree [20]. However, this becomes a problem only asymptotically, and in practice with small computational budgets the algorithm can work well.

B. Multiple Trees in MCTS

Cowling et al. [19] construct one tree per player in Multiple Observer Information Set MCTS (MO-ISMCTS). Their purpose is to construct each tree using information sets from the perspective of a different player, so that the modelled opponent actions use information available to them, but not to the acting player. Like us, MO-ISMCTS updates all trees using a single trajectory, making decisions for each player using the statistics in their tree and adding one node to each tree per iteration. However, each tree in MO-ISMCTS contains the actions of all players where these are visible to the tree-player. In contrast we only include the actions of the tree-player.

IV. MultiTree MCTS

In MultiTree MCTS the overall shape of the MCTS algorithm described in Section II-A is unchanged. However, because each iteration descends multiple trees, one per player, it is possible for some of these to be in the Rollout phase, while others are in the Selection or Expansion phases. Hence the major change is to interleave these three phases on a single trajectory as outlined in Algorithm 1.

We require a transition function $T(s,a) : s_{t+1} = T(s_t, a_t)$, where s_t, a_t are the state at turn t and the action taken from that state; a player function $P(s)$, that returns the player whose turn it is in state s; $\pi_{roll}(s), \pi_{exp}(n), \pi_{tree}(n)$ policies to use in each of the rollout, expansion and selection phases, where n is the current node in the search tree. We are careful to distinguish the underlying state, s, from the tree node n, as there is no 1:1 correspondence between these in IS-MCTS due to different determinisations at the root on each iteration. This difference is exacerbated in MT-MCTS as different visits to a node may have included different actions by other players. Hence, π_{exp} and π_{tree} both return the action a to be taken, and the node n that this leads to in the current tree.

Starting from a root state on each iteration (line 1), we initialise a root node for each player and store this in an array of current nodes (line 2). For the current player (line 4) we determine what phase they are in (lines 6, 9, 13).

In the selection phase the tree policy (vanilla UCT is used) picks an action and descends the current tree (line 14). The new node reached is stored in the array of current nodes for when the trajectory returns to this player (line 15).

In the expansion phase an action is picked (a uniform random expansion policy is used) and a new node created and stored in an array of final nodes reached (line 11). The current node for the player is set to `null` to mark that the trajectory has left their tree and moved into the rollout phase (line 12).

In the rollout phase an action is picked using the rollout policy (a simple random policy over valid actions is used).

The action selected for the current player advances the game state, s (line 16). This whole loop is repeated until the game ends, or the budget of rollout actions is reached (line 3). MT-MCTS does not require domain-specific knowledge in the form of a move-ordering heuristic function. Its complexity in terms of forward model calls is unchanged from vanilla MCTS, as the single trajectory updates each tree.

The EndNode array stores the last node in each player's tree (the one created as a result of expansion). Where a player has not reached the Expansion phase, the EndNode is set to the last node reached in the tree (lines 17-19). Standard backpropagation then takes place independently on each player tree from the corresponding EndNode. The value that is backpropagated is the value of the final state, s to the player concerned (see further details in Section V).

Algorithm 1: One iteration of MultiTree MCTS

Input: Transition $T(s,a)$, Player $P(s)$, maxRoll, root state, player count Φ, $\pi_{roll}(s), \pi_{exp}(s), \pi_{tree}(s)$
Output: EndNode$[i]$ $\forall i \in 1..\Phi$, end state

1 Initialise roll $= 0, s =$root state,
2 $\forall i \in 1..\Phi$: Node$[i]$ = Node(), EndNode$[i]$ = NULL;
3 **while** *roll* $\leq$ *maxRoll* $\wedge$ *s.isNotTerminal()* **do**
4 $p = P(s)$
5 $n =$ Node$[p]$
6 **if** $n ==$ *NULL* **then**
7 roll = roll + 1
8 $a = \pi_{roll}(s)$
9 **else if** *n has untried actions* **then**
10 $(a, n) = \pi_{exp}(n)$
11 EndNode$[p] = n$
12 Node$[p]$ = NULL
13 **else**
14 $(a, n) = \pi_{tree}(n)$
15 Node$[p] = n$
16 $s = T(s, a)$
17 **foreach** $p \in 1..\Phi$ **do**
18 **if** *Node*$[p] \neq$ *NULL* **then**
19 EndNode$[p]$ = Node$[p]$

At any node we may not have the same set of actions available on each visit, as this can depend on actions taken by other players as well as the precise root determinisation. To avoid over-exploring rarely seen actions, we use the same technique as [19], and maintain separate statistics for $N(a)$, the number of visits for which action a *could* have been taken, as well as $n(a)$, the number of times the action *was* taken. $N(a)$ is used in place of N in the UCT equation.

V. Experiments

We investigate five different MCTS variants:

- MaxN. This constructs a single tree including nodes for all players. Statistics at each node use the back-propagated value (score) specific to the player at that node.
- Paranoid (PN). As MaxN, but the back-propagated value for all opponents is -1 times the value for the root player.
- Self-MCTS (Self). This constructs a single tree including only nodes for the root player. Other players act randomly on their turns. It can be considered an extreme version of MT-MCTS that treats the game as if it is single-player.
- MultiTree (MT). This uses the MT-MCTS algorithm described in Section IV. In each tree the back-propagated value is that of the player in that tree, analogous to MaxN.
- MultiTree Paranoid (MTP). As for MT, but assuming other players act to minimise our score.

In all cases the heuristic used to value a state at the end of an iteration is the current game score; with a bonus of 50% if the player has won, and a malus of 50% if they have lost. The only games not to have a game score are TicTacToe and Exploding Kittens, which have instant win/lose conditions instead of the winner being the player with the highest score at game-end. For TicTacToe a score of +1 for a win, 0.5 for a draw and -1 for a loss is used. For Exploding Kittens a heuristic of the number of other players knocked out of the game is used.

Using the underlying game score takes advantage of a common feature of modern board games without the need to construct and tune game-specific heuristic functions. The alternative approach of running all rollouts to game end is possible, but the length of the games leads to weaker and sparser reward signals and lower overall playing strength in initial experiments.

There is a risk that this game score may be 'deceptive' in the sense of [21], with short-term reward at the expense of long-term gain. This is evident in Dominion where an early pursuit of game score is easily beaten by a more strategic focus on acquiring useful action cards first, and only starting to buy victory point cards much later. This does not affect a comparison between the MCTS variants, but can mean they find a relatively poor optimum and can be beaten by a human player, or by MCTS with a game-specific heuristic function that takes these factors into account.

To pick the remaining MCTS parameters, we use previous work on MCTS tuning on these games in TAG that shows the two most important dimensions in MCTS-parameter space are whether IS-MCTS is used, and the length of rollouts [22]. Based on this tuning we use parameters from four quadrants:

- IS-MCTS, No rollout
- Perfect Information MCTS (PI-MCTS), No rollout
- IS-MCTS, Rollout of 30 actions
- PI-MCTS, Rollout of 30 actions

Quadrant	Games
No Rollout, IS-MCTS	Uno, Love Letter
Rollout = 30, IS-MCTS	Colt Express, Dominion, Exploding Kittens, Poker, Diamant
No Rollout, PI-MCTS	Dots and Boxes, Virus
Rollout = 30, PI-MCTS	TicTacToe, Sushi GO!

TABLE I: Tuned quadrants for each game

PI-MCTS uses a single random determinisation at the root node for all iterations. This is not the actual game state, which is unknown to the players. PI-MCTS assumes hidden information is perfectly known, but this can work surprisingly well in some games [23]. The standard UCT selection rule is used, with K normalised to the maximum game score, and a maximum tree depth of 30.

The TAG framework controls the main game state, and passes a copy of this to each agent when a decision is required, with all hidden information randomised. The agent returns its action selection, and the copy of the game state is discarded. This avoids information sharing between agents. When a set of games is run, the position of each agent is randomised so that the order of decision-making varies across games. This may be important if there is an advantage to going first for example. Three sets of experiments were run (all on a 2.6 GHz Intel Xeon Gold 6240 CPU):

1) **Fixed MaxN opponent**. An initial set of experiments comparing MT, MTP, Self and PN against fixed opponents using MaxN in each game and player count. One player uses the variant, and all others use MaxN. This was repeated in each of the four MCTS parameter quadrants with a 40ms budget and 1000 games.
2) **Full tournaments**. Round Robin tournaments were run for each game with all 20 agents (each of the five variants from each of the four quadrants), with a 40ms budget. 5000 games were run for each setting.
3) **Tuned Quadrant tournaments**. Round Robin tournaments for each game using the five agents from the best quadrant for that game. For each game the quadrant that gives the best average performance in that game is used. (Table I details the quadrants used.) Budgets of 40ms, 200ms and 1s were used. As only five agents are used, the player count is capped at five to avoid duplicating agents. Between 2k and 40k games were run for each tournament, depending on game and budget. Uno was not run at a 1s budget due to time and memory constraints.

VI. Results and Discussion

The first two sets of experiments have issues that affect their interpretation as explained in VI-A. The Tuned Quadrant

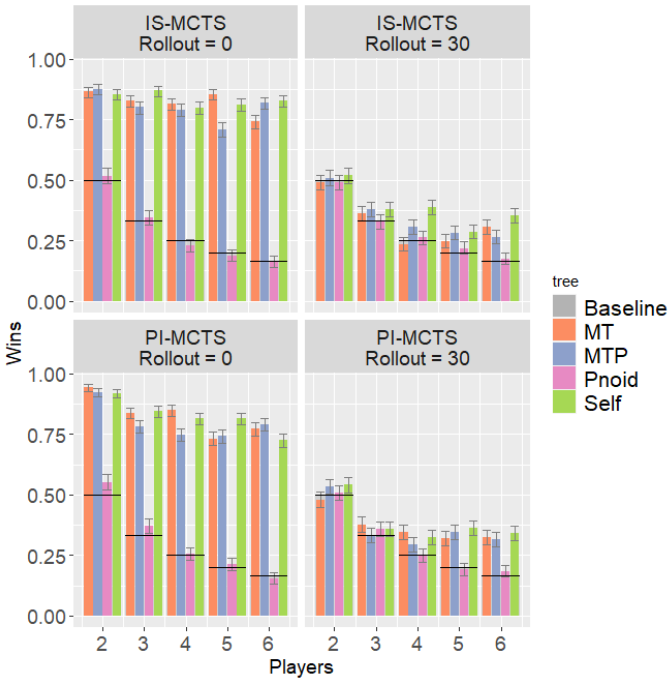

Fig. 1: Each MCTS variant plays individually against MaxN in Colt Express. The baseline in grey/black is the expected win rate at each player count the variant is equal in strength to MaxN. Error bars show 95% confidence intervals. This suggests that MultiTree MCTS is effective if we do not use any rollout.

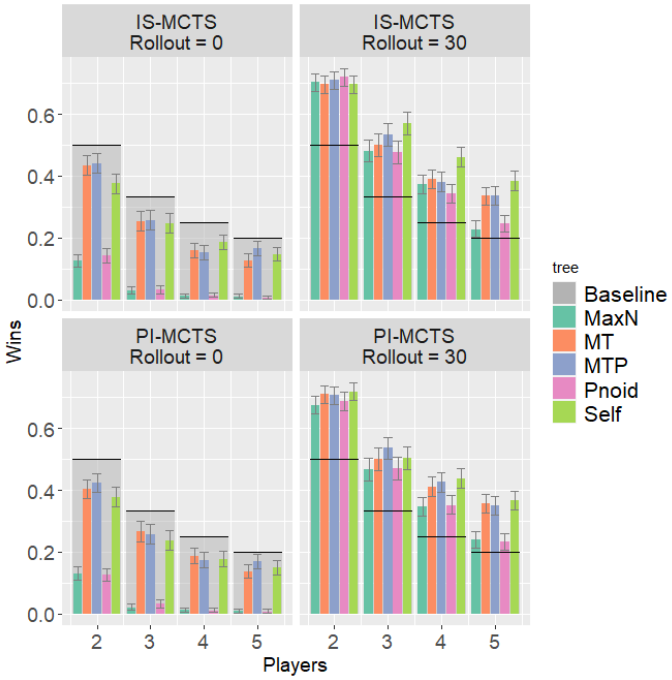

Fig. 2: Results of a Round Robin Tournament between all 20 agents in Colt Express. Zero rollout in the left two quadrants performs poorly, rendering the relative benefit of MultiTree MCTS in this case in Figure 1 moot.

tournaments are most informative, and we discuss these in Sections VI-B to VI-E.

A. Fixed MaxN opponent and full tournaments

Major problems with interpretation of these results arise because only performance against a default MaxN opponent is measured, and because we look at this separately in each of the four quadrants. This has the advantage of changing a single variable at a time, but can give different effects depending on the quadrant. For example, Figure 1 shows the results for Colt Express. In the two quadrants with no rollout, MT, MTP and Self all do very well against MaxN. However, in the other two quadrants (with rollout of 30) this effect disappears for 2/3-players, and is much reduced in magnitude for 4-6 players.

Figure 2 shows the results of the Round Robin tournament between all 20 agents for the same game. A zero rollout is poor in Colt Express, and the fact that using MT or Self policies gives a strong boost to performance over MaxN in this case in Figure 1 is less important than the relatively small change in performance seen when we use a longer rollout length, in which playing strength is stronger across the board.

A better view of the impact of the MCTS variants comes from considering their effect when the remaining MCTS parameters are fixed at values tuned for the game, and when we look at an average performance against all other variants.

B. Tuned Quadrant Tournaments

This is the rationale behind the third set of experiments. For each game the quadrant that gives the best average performance is used (Table I).

Figure 3 shows the best performing variants for each game, player count and budget. In most cases MT and MTP perform similarly, as do MaxN and Paranoid, so for clarity these have been combined. In most games MT-MCTS ('M') is at least competitive, with a win-rate equal to the winner within 95% confidence intervals. In Poker, SushiGo! and Diamant it is the best overall algorithm.

We can broadly partition the games into three groups. One in which MT-MCTS does not provide any benefit over vanilla MCTS ('V') for any computational budget (TicTacToe, Dots and Boxes, Dominion, Exploding Kittens, Uno); a group in which MT-MCTS works well at lower budgets only (Poker, Sushi GO!, Virus); and a group in which MT-MCTS and Self-MCTS are always dominant (Colt Express, Diamant, LoveLetter).

We now look at each of these groups in turn. The full data from one game in each is presented as an exemplar of the pattern in Figures 1, 2, 4. The full set of results, with figures for each game are available in supplemental material online[1].

C. MT-MCTS good at lower budgets only

MT-MCTS works well in Poker, Sushi GO! and Virus at lower budgets and is overtaken or matched by MaxN at larger budgets. Figure 4 shows the detailed results for Poker. This supports the initial hypothesis that the benefit of greater

[1]https://www.tabletopgames.ai/results/MT-MCTS-202204.html

Game	40ms 2P	40ms 3P	40ms 4P	40ms 5P	200ms 2P	200ms 3P	200ms 4P	200ms 5P	1000ms 2P	1000ms 3P	1000ms 4P	1000ms 5P
TicTacToe	V M				V M				V M			
Dots and Boxes	V	V	V	V	V	V	V	V	V	V	V	V
Uno	V	V	V M	V M	M S	V M S	V M S	V M S				
Poker	V M	M	M	M	M	V	M	M	M	V	V	V
Exploding Kittens	V M S	V M S	V M	V	V M S	V M S	V M	V M	V M S	V S	V M S	V M S
Virus	V M S	M S	M	M S	V	V M S	M S	S	V M S	V M S	V M S	M S
Dominion	V S	V S	S		V S	S	S		M S	V S	V	
Sushi GO!	S	M	M S	M S	M	M	M	M	V M	V M	M	M
Love Letter	S	M S	M S		S	M S	M S		S	M S	M	
Diamant	M	S	S	M S	M	M S	M S	M S	M	M	M S	M
Colt Express	S	S	S	S	M S	M	S	S	M S	M S	M S	M S

Fig. 3: The best performing variants for each game, player count and time budget. 'V' (for vanilla) is used for MaxN or Paranoid, 'M' for MT or MTP, and 'S' for Self. More than one coloured box show no significant difference in win rate at a 95% confidence level. Grey boxes are for non-supported player counts, or experiments not run (Uno). These are roughly sorted with games for which vanilla MCTS is best at the top, to those for which considering opponent moves is least useful.

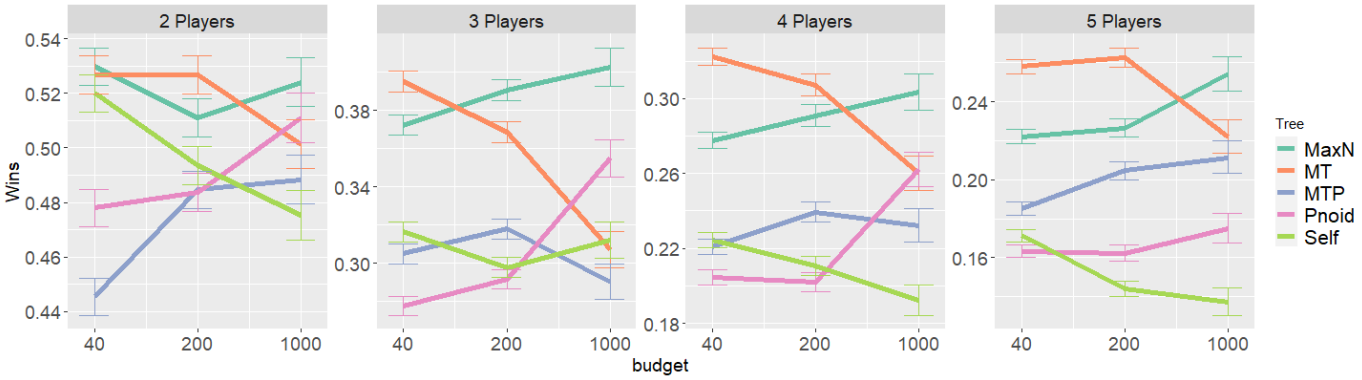

Fig. 4: Tuned Quadrant Tournament results for Poker with computational budgets of 40, 200 and 1000ms. As the budget increases the benefit of MT-MCTS declines with MaxN becoming dominant. This decline is less pronounced as the number of players increases. Self-MCTS declines in relative performance as the budget increases.

search depth can help even at the cost of a poor modelling of an adversarial opponent. As the budget increases MaxN can search to a useful depth while modelling the benefit of actions conditional on those of an opponent (and vice versa).

As the player count increases, the benefit of MT-MCTS takes longer to decline (compare the trajectory of the orange MT line in the plots of Figure 4). This is expected as the more players included in a single tree, the less deep an agent can search with respect to its own decisions.

Poker and Virus are 'adversarial' in that opponent actions directly affect one's own available actions (Poker) or material position (Virus, in which players play cards to infect others' organs and reduce their points). Sushi GO! is less expected as here players construct sets of cards in their own tableau, and cannot directly affect other players. However, it is one of two games in the set in which all players take their actions simultaneously, and the single tree of vanilla MCTS is incorrect with its innate assumption that other players will be able to take into account one's own card-play. This may explain why MT-MCTS is the best overall variant here.

D. MT-MCTS good at all budgets

A related pattern can be seen in Colt Express, Diamant, and to a lesser extent in Love Letter. At lower budgets Self-MCTS and MT-MCTS are the competitive variants with MaxN/PN performing poorly. Figure 5 shows the detailed results for Colt Express. At higher budgets Self-MCTS declines in relative performance, with MT-MCTS doing better. For all player counts above 2, Figure 5 also shows MaxN/PN increasing in relative performance at 1000ms, but not sufficiently to surpass the other variants.

This is consistent with the argument in VI-C. At smaller time budgets the benefit of searching deeper for an agent's own plan is greater than any loss from ignoring opponent counter-actions. As the budget increases this balance shifts, and the benefit of deeper search under the incorrect assumption of a single-player environment has diminishing returns.

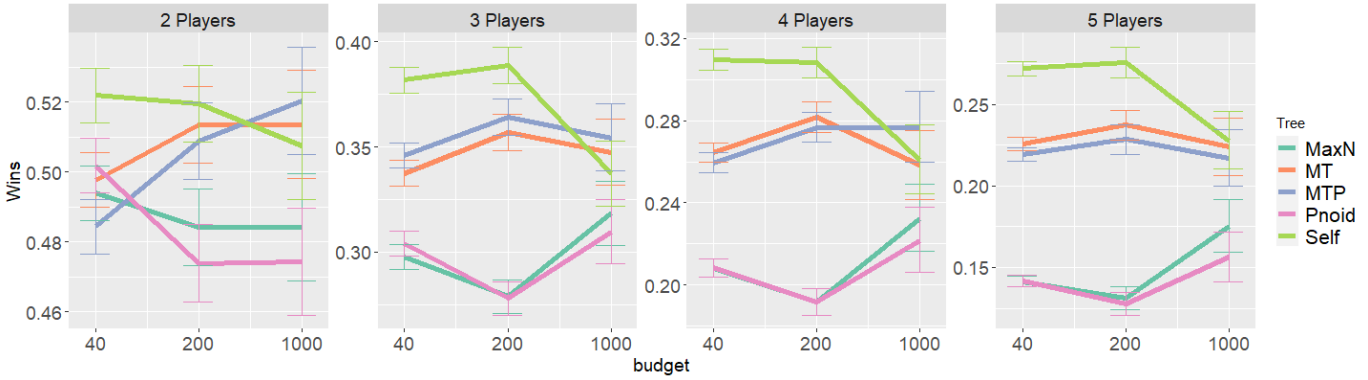

Fig. 5: Tuned Quadrant Tournament results for Colt Express. Self-MCTS is the best performer, but its advantage declines as the computational budget increases. MaxN and Paranoid increase in performance with budget, but are poor even at 1000ms.

Colt Express has two phases. A planning phase in which players play cards (some hidden from opponents) that form an order stack, followed by an execution phase that implements this stack in sequence (alternating between players). There is a particular benefit in a deeper tree search focusing only on one's own actions in the first phase, as this requires a coherent set of actions (a plan) to be formed. This result of Self and MT-MTCS working well is therefore expected.

Diamant is the second simultaneous move game in the set, but with a branching factor of two, as each round players decide simply on whether to 'Stay' or 'Leave'. Like Sushi GO! it is one in which MT-MCTS performs most strongly overall. Independent trees for each player are better for modelling the first decision of other players which should not be conditional on the other (unknown) actions, even though actions on later turns should formally be conditioned on previous actions of all players.

E. MT-MCTS not helpful

In several games MT-MCTS is downright harmful, or provides no benefit over vanilla MCTS. One subset consists of perfect information, non-simultaneous games: Tic-Tac-Toe, Dots and Boxes. These are exactly the games where we expect MT-MCTS to be poor. In Dots and Boxes a bad move is immediately exploitable by the other players, and vanilla MCTS methods that condition on opponent moves are essential. MaxN and PN play a perfect game of TicTacToe at 40ms, and either draw or win. MT-MCTS in either form loses the occasional game against them and is strictly worse, although not within the statistical significance bounds used in Figure 3. In this trivial game with a maximum tree depth of 9, assuming the opponent acts randomly is not a good strategy and there is no trade-off with depth of search.

All variants do equally well in Exploding Kittens, despite the existence of instant-win situations, in which modelling an adversarial response might be expected to help. The reasons for this are not clear, but may be related to the very stochastic environment.

The unexpected game in this set is Dominion, in which there is little direct player interaction, and the game is mostly a race to get the victory points available. This is a game that MT-MCTS and Self-MCTS were expected to do well in. In this group the exemplar is reverse-cherry-picked to avoid giving the impression that all games fit the pattern. Figure 6 shows the detail of tournament winners by player and budget count. Unexpectedly we see a decline in MaxN and PN performance in 2-player Dominion with an increasing budget, but the expected increase in performance for 3/4-players, along with the concomitant fall in performance of Self-MCTS. Dominion is a relatively long game in turns which limits the number of games in each tournament, and hence the wider error bars in Figure 6, but the changes are still clear, even if the reason for them is not.

VII. Conclusions

We have introduced a new variant of MCTS that constructs one tree independently for each player, including only their own actions. This permits deeper tree search that focuses on the agent's own plan, at the expense of less effective modelling of how other players may respond, in line with the inspiration behind [8]. We have tested this in a variety of 11 modern tabletop games with differing player counts. MT-MCTS is of particular benefit in games with simultaneous moves, where its independent trees better model the information available for the current decision. For relatively low computational budgets, MT-MCTS or an extreme version, Self-MCTS, that assumes a single-player game environment, are superior to standard MCTS in many games. However, games in which there may be specific counter-moves an opponent can make (TicTacToe, or Dots and Boxes) do not benefit from this and performance can be very poor. As the computational budget increases, the average balance shifts in favour of standard MCTS, although the details are highly game-specific.

One limitation in this work is that MCTS parameters have been tuned over just four settings (quadrants) based on average performance with a 40ms budget, although these settings are suggested from previous work over a much richer search space. This quadrant is used for all player counts, MCTS variants and budgets. For a fairer comparison, MCTS parameters should be tuned independently for each variant, game, budget and player

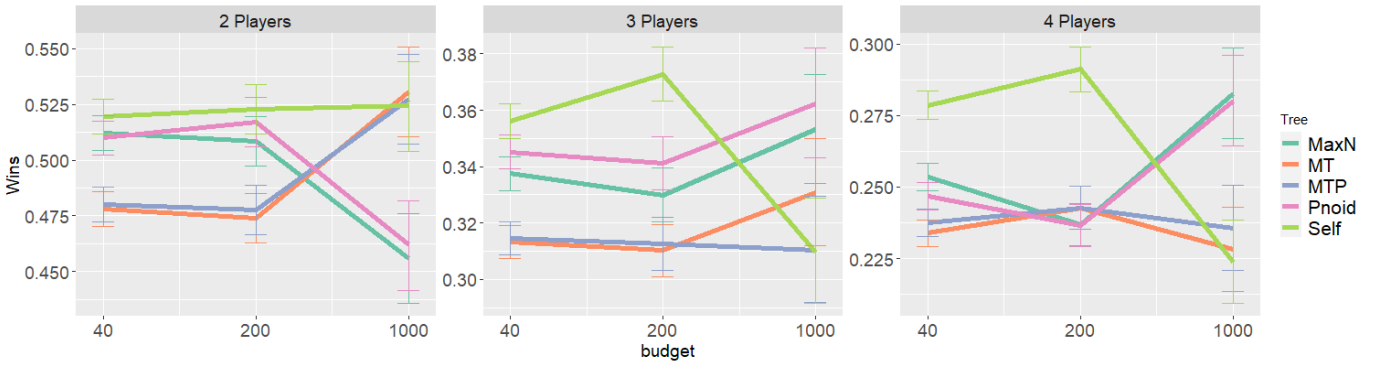

Fig. 6: Tuned Quadrant Tournament results for Dominion. This shows very different patterns for 2-player versus 3/4 players. In the latter increasing budget harms Self-MCTS and helps MaxN/PN as expected. With 2-players MaxN/PN get significantly worse with budget. This example illustrates that not all games follow the overall pattern.

count. None of the games showed changes in best performing quadrant with player count, and Figure 2 is representative, but it is possible that the best parameters for a game could be significantly different at 1000ms than the 40ms used for this initial tuning. This is for future work.

Other areas for future work are to understand in more detail why some games are robust to any MCTS variant, while others are very sensitive; and to compare MT-MCTS explicitly with OMA [8], which weights the contribution of the full conditional action tree with a Self-only tree.

One other avenue is to consider whether there are specific parts of a game in which MT-MCTS or Self-MCTS can be used producutively, shifting to vanilla MCTS at other times. An example here is Colt Express with its distinct Planning and Execution phases; using MT-MCTS in the first and vanilla MCTS in the second may perform better than either alone.

Acknowledgment

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/S022325/1.

References

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

[2] K. Burgun, "Why Many Eurogames Are Inherently Single-Player Games," Feb. 2015. [Online]. Available: https://keithburgun.net/why-eurogames-are-inherently-single-player-games/

[3] P. J. Jansen, "Using Knowledge about the Opponent in Game-Tree Search." CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 1992.

[4] J. Schaeffer, "A gamut of games," *AI Magazine*, vol. 22, no. 3, pp. 29–29, 2001.

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

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017. [Online]. Available: http://www.nature.com/doifinder/10.1038/nature24270

[7] M. P. Schadd and M. H. Winands, "Best reply search for multiplayer games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 1, pp. 57–66, 2011, publisher: IEEE.

[8] H. Baier and M. Kaisers, "Guiding Multiplayer MCTS by Focusing on Yourself," in *IEEE Conference on Games (CoG)*, 2020.

[9] ——, "Opponent-Pruning Paranoid Search," in *International Conference on the Foundations of Digital Games*, 2020, pp. 1–7.

[10] G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, and H. J. Van Den Herik, "Monte-carlo strategies for computer go," in *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, 2006, pp. 83–91.

[11] 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, pp. 1–43, Mar. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6145622/

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

[13] R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, "Design and Implementation of TAG: A Tabletop Games Framework," *arXiv preprint arXiv:2009.12065*, 2020.

[14] S. V. Albrecht and P. Stone, "Autonomous agents modelling other agents: A comprehensive survey and open problems," *Artificial Intelligence*, vol. 258, pp. 66–95, 2018.

[15] S. Nashed and S. Zilberstein, "A Survey of Opponent Modeling in Adversarial Domains," *Journal of Artificial Intelligence Research*, vol. 73, pp. 277–327, 2022.

[16] C. Luckhart and K. B. Irani, "An Algorithmic Solution of N-Person Games." in *AAAI*, vol. 86, 1986, pp. 158–162.

[17] N. R. Sturtevant and R. E. Korf, "On pruning techniques for multi-player games," *AAAI/IAAI*, vol. 49, pp. 201–207, 2000.

[18] M. Esser, M. Gras, M. H. Winands, M. P. Schadd, and M. Lanctot, "Improving best-reply search," in *International Conference on Computers and Games*. Springer, 2013, pp. 125–137.

[19] 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, pp. 120–143, Jun. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6203567/

[20] T. Furtak and M. Buro, "Recursive Monte Carlo search for imperfect information games," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[21] D. Anderson, M. Stephenson, J. Togelius, C. Salge, J. Levine, and J. Renz, "Deceptive games," in *International Conference on the Applications of Evolutionary Computation*. Springer, 2018, pp. 376–391.

[22] J. Goodman, D. Perez, and S. M. Lucas, "Visualising Multiplayer Game Spaces," *IEEE Transactions on Games*, 2021, publisher: IEEE.

[23] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search." in *AAAI*, 2010.