MCTS pruning in Turn-Based Strategy Games

Yu-Jhen Hsu, Diego Perez-Liebana
School of Electronic Engineering and Computer Science
Queen Mary University of London, UK

Abstract

Large action spaces is one of the most problematic aspects of turn-based strategy games for all types of AI methods. Some of the state-of-the-art algorithms such as Online Evolutionary Planning and Evolutionary Monte Carlo Tree Search (MCTS) have tried to deal with this problem, but they required a fixed number of actions in each turn. In general strategy games, this assumption can't be held, as the number of actions that can be executed in a turn is flexible and will vary during the game. This paper studies pruning techniques and the insertion of domain knowledge to deal with high branching factors in a new turn-based strategy game: Tribes. The experiments show that, with the help of these techniques, MCTS can increase its performance and outperform the rule-based agents and Rolling Horizon Evolutionary Algorithms. Moreover, some insights into the tree shape and the behaviour of MCTS with and without pruning techniques are provided.

1 Introduction

Turn-based multi-action strategy games raise the interest in the AI domain due to, among other things, their huge branching factors. These kinds of games contain a considerable number of actions within a turn, which the player executes in sequence. The order of actions is important as previous actions influence latter ones. The branching factor can significantly increase up to billions next states. Compared to Go and Chess, the branching factor is much higher in turn-based strategy games (Perez et al. 2020). Taking Bot Bowl (Justesen et al. 2019) as an example, in which the goal is to manage a football team where each unit can execute several actions, the branching factor can be as high as 10^{51} in a given turn. The branching factor for Tribes (Perez et al. 2020) and HeroAIcamdy (Justesen, Mahlmann, and Togelius 2016) are 10^{31} and 10^{8}, lower than Bot Bowl but still much higher than Go and Chess. Tribes is a strategy game that introduces extra management complexities, as it requires controlling not only the units but also cities, tech trees and resources. Some recent progress has been made by analysing how huge branching factors impact the performance of agents. High branching factors drop the performance of search algorithms (Justesen et al. 2017; Baier and Cowling 2018). The difficulty of developing an evaluation function makes the situation even worse (Justesen et al. 2017; Browne et al. 2012), as the search algorithms require them to guide decision making. Although Monte Carlo Tree Search (MCTS) can handle large branching factors, it fails when branching factors reach the sizes mentioned above (Baier and Cowling 2018; Chaslot et al. 2008). In the default tree policy of MCTS, all nodes are required to be visited before exploitation happens, making non or rarely visited less informative for making a good decision (Gelly and Wang 2006).

New search algorithms have been developed to tackle strategy games, such as Online Evolutionary Planning (OEP) (Justesen et al. 2017) and Evolutionary MCTS (EvoMCTS) (Baier and Cowling 2018), obtaining decent performance in HeroAIcamdy. Instead of using a single action as genomes or nodes, they use a sequence of actions as the representation. The difference between OEP and EvoMCTS is the way to obtain an action set over the turn. OEP uses crossover and mutation operators to choose the best action set. EvoMCTS combines MCTS and an evolutionary algorithm to mutate action sets to guide search in a tree structure. These two algorithms choose a fixed number of actions for a turn, and each action is assigned to a specific unit. This assumption makes it hard to adapt for a game like Tribes (or most strategy games) for three reasons. First, the number of actions might change after executing an action during the same turn. Secondly, the branching factor generally increases during the game. Lastly, instead of managing units, in Tribes agents require to manage not only unit actions but also cities and general faction actions. These factors increase the difficulty to estimate the length of a fixed size genome.

Using a single action rather than a complete turn as a representation for MCTS is one possible solution. It decreases the branching factor and makes the tree deeper (Baier and Cowling 2018). However, vanilla MCTS without improvement fails as the tree is still shallow (Justesen et al. 2017; Baier and Cowling 2018). Numerous enhancements for MCTS have been proposed to deal with this issue. One possible solution focuses on pruning, reducing the branching factors by removing some weak actions based on evaluation functions or domain knowledge that increases the converge speed to find good action.

This paper aims to investigate classic pruning techniques and analysing the tree structure and behaviours of MCTS for Tribes. This paper compares the performance of MCTS with pruning with the original MCTS agent, Rolling Horizon Evolutionary Algorithms (RHEA) agent and rule-based agent which are agents that outperformed MCTS in the original Tribes paper (Perez et al. 2020). Moreover, the different MCTS pruning techniques are analysed by their tree shape in terms of the influence of turn and branching factor.

2 Background and related work

2.1 Monte Carlo Tree Search (MCTS)

MCTS (Browne et al. 2012) is a tree search algorithm that explores the most promising parts of the state space by balancing exploitation and exploration, and it has been successfully applied to many games. The tree, being asymmetric, contains nodes representing the state and edges representing the actions. There are four main steps in the default MCTS: selection, expansion, simulation and back-propagation. In the selection step, the tree policy (Upper Confidence Bounds - UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) - in the most used version) is applied to guide the search from the root until finding a node with non-visited children. An action chosen at random is added in the expansion step followed by the simulation step, performing a random rollout (Monte Carlo simulation) until reaching a terminal state. While it is one of the main steps, some recent studies (Perez et al. 2020; Baier and Cowling 2018) have shown that skipping the rollout and directly evaluating the expanded node's state can provide better results. The final step, back-propagation, evaluates the terminal state and then updates all nodes that are visited in this iteration. MCTS ends when it runs out of budget and returns the most visited child of the root.

There are different enhancements used to improve the performance of MCTS under large branching factors. These methods either reduce the branching factor by domain knowledge or handle the requirement of visiting all nodes before exploitation. First-Play urgency (Gelly and Wang 2006) encourages early exploitation by assigning the initial value to non-visited nodes. It reduces the need for visiting every node before exploitation under UCT and allows the tree to go deeper. Move groups (Childs, Brodeur, and Kocsis 2008) is another technique for reducing the branching factor and increasing the performance of MCTS in Go. Instead of choosing a single action, a move group (which contains similar moves with a highly-correlated expected value) is picked and evaluated. Rapid Action Value Estimation (Gelly and Silver 2007) updates the value for all of the nodes with the same action and state regardless of its position in the tree. A script approach such as Hierarchical Portfolio Search (Churchill and Buro 2015) and Portfolio Greedy Search (Churchill and Buro 2013) specialized in dealing with huge branching factors in real-time strategy games, searches over the small amount of hand-coded scripts rather than the whole action set.

2.2 Pruning techniques for MCTS

Pruning is another technique used to deal with huge branching factors. The benefits of pruning are that poor actions can be ignored, focusing more on promising actions, resulting in a much deeper tree and potentially a quicker convergence. Moreover, it can be used without heuristic functions and can be applied in any domains (Browne et al. 2012). Some pruning also uses domain knowledge to remove some trap states (Ramanujan, Sabharwal, and Selman 2010), which are states which have a high reward but are leading to loss.

Pruning techniques can be divided into two categories: hard and soft pruning. The hard version prunes the tree by eliminating actions with the lowest evaluated score. Soft pruning prunes actions after certain iterations, but the eliminated actions will be chosen in a later iteration. The benefit of soft pruning is alleviating the risk of removing the good actions (Browne et al. 2012) while hard pruning allows the tree to go deeper (Justesen et al. 2017). Progressive widening (PW) (Coulom 2007) or Progressive Unpruning (Chaslot et al. 2008) is an example of soft pruning, showing big success in the game Go. N nodes are pruned based on evaluation functions after the iteration number exceeds a threshold. After several simulations, nodes are unpruned and can be chosen in the selection step. In this way, all actions will be visited, given enough budget. This technique provides a similar effect to First-Play urgency, encouraging early exploitation and allowing the tree to go deeper. While pruning techniques can be used without heuristics, the performance of progressive widening depends on the heuristic function. (Teytaud and Teytaud 2009) show that progressive widening without the help of heuristic has a little impact on improving the strength of the agents in the game Havannah.

2.3 Rolling Horizon Evolutionary Algorithms (RHEA)

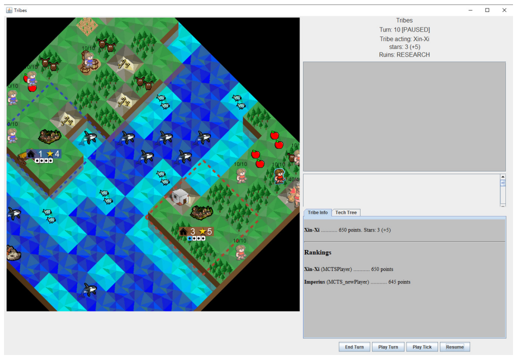
RHEA is an evolutionary algorithm that achieves good results in several real-time games(Perez et al. 2013). RHEA learns online, during the game, in a different way to other evolutionary algorithms that learn offline. It first generates random individuals, each one representing a sequence of actions. It then evaluates the final state arrived after executing, from the current state, all the actions in the individual sequentially. Evolutionary operators such as selection, crossover and mutation are used to generate new individuals based on old ones. The first action in the individual with the highest value is returned once the budget is exhausted.

3 Methods

3.1 Tribes

Tribes (Perez et al. 2020) is a re-implementation of the game The Battle of Polytopia (Midjiwan AB 2016), a popular award-winning turn-based strategy game, which can be seen as a simplified version of Sid Meier's Civilization. Fig. 1 shows the interface of Tribes.

The game takes place in $N \times N$ tiles. Each tile contains the terrain type (plain, mountain, shallow water or deep water). Each tile can only have one resource, building and unit at the same time. Villages, resource and ruins will be generated in


Tribes
Turn: 10 [PAUSED]
Tribe acting: Xin-Xi
stars: 3 (+5)
Ruins: RESEARCH

Xin-Xi .... 650 points. Stars: 3 (+5)

Rankings

Xin-Xi (MCTSPlayer) .... 650 points

Imperius (MCTS_newPlayer) .... 645 points

Figure 1: The user interference of Tribes.

tiles at the beginning of the game. Ruins contain randomized bonuses (special unit, technology, resources, etc.) and can be explored by units. There are two game modes, being *Capital* and *Score* mode. The goal in the Score mode is to maximize the score or take all the enemies' capitals within 30 turns. Instead of reaching as much score as possible within 30 turns, the Capital mode only requires to capture the enemy's capital, or initial city. The score comes from a variety of actions such as exploring, resource gathering, capturing villages and cites, and constructing determined buildings. At the beginning of the game, each player controls one of the four available Tribes (Xin_Xi, Imperius, Bardur and Oumaji). They have a starting unit and a researched technique, being different for each tribe. Each player starts with five stars, which are the resource currency. Each player executes as many actions as possible within a turn until they decide to end it, or no more actions are available.

There are three crucial aspects of the game, technology, economy management and combat, which determine the playing strategy. The critical element in economy management is handling stars, which is used for building, gathering resources, spawning units and researching technologies. Stars are produced by cities on each turn. Buildings and resources provide population to the city. When the population exceeds the level of the city, the city is upgraded, and a bonus (extra production, units, city border growth, etc.) is unlocked. For city levels 5 and higher, the player can select a bonus between the strongest game unit (super unit or Giant) or building a park (extra score). The higher the city level is, the higher the number of stars it produces per turn. The building can only be built in their own territory, which is formed by the 3×3 tiles around the city. The types of units and buildings that can be spawned and constructed are limited at the beginning of the game, and more types are unlocked by researching the different 24 technologies available. Two types of combat units can be spawned, being melee (Warrior, Rider, Defender, Swordsman, Knight) and ranged (Archer, Catapult). Each unit has different attack power, range and health points (HP). They can also embark in port buildings and become a boat unit. Boats have three levels: boat, ship and battleship. Attack power, movement range and HP are increased when upgrading the boat unit. Moreover, a different type of units can perform a distinct combination of actions per turn. Most of them can attack and/or move in a turn, but others can do multiple consecutive actions. Once a unit defeats three other units, the former is upgraded to a veteran, having much higher HP. Units can capture the enemy's city or neutral villages after staying in that city/village for more than a turn. The Mind Bender is a special unit which can heal friendly units and convert enemies to their faction. More details about the game can be found in the original paper (Perez et al. 2020).

There are three kinds of actions in Tribes, being *unit* (the only set without requiring stars to perform actions), *city* and *tribe*. The *unit* actions include attack, move, capture, convert, disband, examine, upgrade and become a veteran. Some of these actions are specialized for the particular unit type, or can only be performed when its corresponding technique is researched. The *city* actions include constructing and destroying buildings, burning, growing and clearing forests, resource gathering, spawning a unit and levelling up. The *tribes* action includes research, build a road in a non-enemy tile and end the player's turn. Roads speed up the units' movement and can be used by any player. Some actions decrease the action space for a turn. For example, the build action consumes resources and results in lacking enough stars for executing subsequent actions in that turn. The other actions, such as researching unlocking features, increases the action space for a turn. This situation makes Tribes a complex environment for decision making, with huge branching factors when a player owns many units and cities. A strong playing strategy must balance actions for technology, economy management and combat, as well as a proficient order of actions per turn.

3.2 Baseline Approaches

We use 3 algorithms from the original framework (Perez et al. 2020) as baseline approaches: MCTS, RHEA and rule-based agents. It is interesting to compare to RHEA and rule-based agents as they outperform MCTS in the original paper.

Rule Based (RB): The simple rule-based agent is the only agent that does not rely on the forward model in this paper. The forward model allows the agent to sample a possible future state given a particular action in the current state. In (Perez et al. 2020), RB has the second-highest winning rate and in particular beats MCTS in 56.2% of the games. The agent only uses the information of the current state, evaluating each action separately and returning the one with the highest value. If there is more than one action with the highest value, it picks up randomly to break the tie. The evaluation is hand-crafted and relies on domain knowledge. The main weakness of this agent is that it considers every action independently and does not consider the action's effect in the game state. The insights of this agent are that it disables the destroy and disband action and focus on capturing, examining ruins and turning units into veteran ones when possible. For more details about rule-based agents can refer to (Perez et al. 2020).

Rolling Horizon Evolutionary Algorithms (RHEA): RHEA, described in Section 2.3, achieved the best performance in (Perez et al. 2020) with a 63% winning rate against MCTS. In the experiments of this paper, the RHEA agent uses the same parameters: population size of 1 and individual length of 20. Similar to the RB agent, the disbanding action and destroying actions are removed.

Monte Carlo Tree Search (MCTS): MCTS, described in Section 2.1, has below 50% wining rate agents RHEA and RB in (Perez et al. 2020). This MCTS variant prioritizes the root action by picking up different subsets of actions from city, unit and tribe actions. This prioritization is only applied to the root, not the rest of the nodes. The maximum depth of the tree is 20. UCB1 is chosen as the tree policy, with an exploration constant set to $\sqrt{2}$. No rollouts are performed in the simulation step, instead of evaluating the state being rolled based on default policy until the tree reaches certain depth or terminal state, the state of the current node is evaluated, and the value is updated in the back-propagation step. To differentiate it from the other MCTS versions proposed in this paper, we refer to this as *Default* MCTS.

Heuristic: A heuristic function is used to evaluate a game state. It compares the difference between two states and evaluates them based on seven features. The features in (Perez et al. 2020) include the differences between production, number of technologies, score, number of units, number of kills and the total level of cities. Different features have different weights, set to 5, 4, 0.1, 4, 2, 3 and 2 respectively, by trial and error. Two features added in this paper, stressing the importance of building and occupying the enemy's cities. Every new building will provide 5 points and each unit in the enemy's city will add 10 points to the score of the state.

3.3 MCTS with hard pruning

MCTS with hard pruning modifies the tree policy of MCTS to reduce the branching factor. The tree policy not only guides search, it also prunes the node's children if they are visited from the parent more than the 20 times. The number of remaining nodes, being the number of children kept after pruning, is based on the branching factor and the limit for minimum nodes, shown in Equation 1. It will prune the children of nodes until the number of children matches the number of remaining nodes by evaluating its state based on heuristic described above. The children with the lowest value are pruned and never visited in the later iteration.

$$remainingNodes = \max(\alpha \log n, T) \qquad (1)$$

α controls the speed of pruning when the number of actions grows. This parameter plays a vital role in deciding the number of the remaining nodes when the branching factor becomes huge. The higher it is, the more children are kept. T is the minimum number of actions that need to be kept after pruning. If there is any action set smaller than T, pruning is not applied. The reason for using logarithm to prune the tree is to reduce most of the actions when the actions set becomes extremely large. The parameters are decided by running the experiment under a different setting. The setting with the highest winning rate is used.

Besides pruning by the tree, the destroy action is excluded as it does not provide any rewards and results in a reduced score. Furthermore, MCTS with hard pruning has another parameter to decide to prune the move actions or not. Thus there are two versions of MCTS with hard pruning: with and without move pruning. The reason for pruning the move is that it increases the action space significantly: every unit has multiple move actions based on its move distance and the road. When it comes to the later game, a player tends to have many units, increasing the action space significantly. The move action is pruned by the equation 1. It firstly extracts the move action from all action sets. The move actions whose destination is not a ruins or a city tile are extracted and picked by the equation 1 with $\alpha = 1$ and $T = 5$. The pruned move action will then be removed and never be considered in the action set of the current node. Although the pruned actions are removed in the current node, they might appear in the following nodes meaning they still will be considered in a later step.

3.4 MCTS with progressive widening

This variant of MCTS is a modification of MCTS with hard pruning. Instead of permanently removing the pruned nodes, the algorithm unprunes the nodes in later iterations. This process allows the tree to search for every node and alleviate the risk of pruning the best action. It is done by increasing the number of remaining nodes if the iteration number is over the value given by Equation 2 (Chaslot et al. 2008). When the number of iterations is greater than Υ, a pruned action picked uniformly at randomly is unpruned.

$$\Upsilon = \alpha \beta^{n - n\_init} \qquad (2)$$

α and β control the unpruning ratio, and they are set to 20 and 1.7 respectively. The higher value β is, the higher the number of iterations is needed for unpruning, approximating the behaviour of hard-pruning. n is the size of the unpruned set of the current node and n_init is the initial number of unpruned nodes, set to 5. As for the hard pruning case, two variants of MCTS with progressive widening have been tested: with and without move pruning.

4 Experiments and results

This section provides the experimental setup for testing MCTS with pruning followed by the results of our tests. Finally, an analysis of the MCTS tree is provided.

4.1 Experimental Setup

Both versions (hard pruning and progressive widening) of MCTS with pruning (i.e. with and without move pruning) were pitched against the rule-based, the default MCTS and RHEA agents, as included in the framework. For all search agents, the stopping condition is set to 2000 usages of the forward model's *next* method.

Each agent pairing plays 5000 games (25 seeds with 200 repetitions). The seeds are the same as in the original paper (Perez et al. 2020) and they are used to generate different levels, all of size 11 × 11 tiles. Because levels might not be balanced for both contestants, players swap starting positions within the 40 repetitions of every seed. Each game is independent, meaning that no information is carried from one game to the next. The game mode is Capital with full observability, thus the winning condition is set to take over the

Table 1: Averaged winning rate for each version of MCTS vs 3 baseline agents over 5000 games.

	Hard Pruning			**Progressive Widening**		
	RB	**Default MCTS**	**RHEA**	**RB**	**Default MCTS**	**RHEA**
Move Pruning	56.9%	63.3%	57.8%	57.8%	62%	57.7%
No Move Pruning	54.7%	61.2%	54.5%	55.8%	60.1%	57.7%

enemy's capital city. In order to have finite games, a winner is declared after 50 turns, corresponding to the player with the highest score at that point.

4.2 Game Results

Table 1 shows the results of both versions of MCTS competing with three baseline approaches. The first row and first column indicates the version (hard pruning or progressive widening) and with or without move pruning respectively. Default MCTS shows 43.8% and 37% winning rates when playing against the RB and RHEA agents, respectively (not in the table). With the help of pruning techniques, all versions of MCTS have more than 50% winning rate playing against the baseline approaches.

With the help of hard pruning, MCTS increases its win rate approximately 8% and 12% compared to the original MCTS when playing against RB and RHEA, respectively. Move pruning and progressive widening further increase the performance of MCTS with hard pruning. MCTS with hard pruning and move pruning shows the highest winning rate when competing against the original MCTS and RHEA. As can be seen, pruning moves increases the performance of MCTS in relation to only applying progressive widening or hard pruning. Although using hard pruning or progressive widening is clearly better than using no pruning at all, the difference between these two methods is minimal. The winning rate is higher for progressive widening when playing against RB, but lower than hard pruning when the opponent is default MCTS and RHEA. The difference however is likely not to be significant, and different parameters for Equation (**?**) may alter the unpruning rate and provide a better value estimate of the different moves. Overall, although hard pruning, progressive widening, and domain knowledge (move pruning) can improve the performance, using domain knowledge has clearer effects in the winning rate.

Table 2 shows the statistics of the different versions of MCTS over 15000 games (aggregated for each 5000 pairings against the baseline approaches). It shows the statistics of average winning rate over three baseline approach, final score, researched technologies percentage, number of owned cities at the end game and final production. The values obtained for these statistics for the default MCTS in (Perez et al. 2020) (not in this table) are 9966.55, 85.27%, 2.23 and 16.96, respectively. The score and researched technologies percentage of every version of MCTS are considerably smaller than the original MCTS and have a similar value to the ones of the RB agent (8076.32 for scores and 71.14% for researched technologies). However, the number of cities owned at the game end and final production values are slightly larger, now between 2.24 and 2.26 for the former, and above 17 for the latter. Results also show that MCTS with progressive widening has higher number of researched technologies, and move pruning produces higher final scores, cities and production. This statistic suggests that the modified MCTS is more focused on occupying cities rather than trying to maximize the final scores, which may be a consequence of the new features in the heuristic function that gives extra value for units in the enemy's cities. In comparison to RB and RHEA, the score and researched technologies percentage of MCTS sit between those two agents, while the number of owned cities and final production is smaller than RB and RHEA. The statistics of the different versions of MCTS explored in this work, in relation to RB and RHEA, are comparable with those of default MCTS.

4.3 Tree analysis

This section shows which action groups and actions are executed more often, an analysis of the tree depth and the fully expanded rate of the recommend action node in terms of different action spaces and turns in the game. The fully expanded rate of the recommended action node is measured between 0 and 1. If the value is 1, it means the recommended action node has visited all of its possible children (excluding the pruned nodes), from its current state. A value of 0 indicates that no children have been explored[1].

During the game, the *unit* action group is the most frequently chosen: 68% of the time. The *city* and *tribe* action groups are only chosen 20.5% and 11.4% of the time respectively. This difference are a consequence of unit actions taking a large proportion of the action space. The most frequently executed action is *move*. The move action has the largest percentage (58.9%) among all actions, also explaining why the *unit* action group has the largest proportion. The move action does not provide a significant reward per se, but it can lead to future reward if the unit moves to ruins or enemy cities. Applying the move pruning does not change the proportion between the action groups. This may be caused by move actions being taken towards the end of the turn.

Figures 2 and 3 show the influence of the branching factor in the depth of MCTS and fully expanded rate of the recommend action node. The original MCTS is unable to go deeper than 2 levels and visit all the possible actions from the current state when the branching factor is over 50. With the help of pruning techniques, this problem happens only when the size of the action space grows over 180. This is three times larger than in the original MCTS, providing better action value estimates in early game turns (when the action

[1]Note that these are the children of the node of the recommended action, not the children of the root.

Table 2: The aggregated statistics for each version of MCTS over 15000 games

	Hard Pruning					**Progressive widening**				
	Win Rate	**Score**	**Techs**	**Cities**	**Prod.**	**Win Rate**	**Score**	**Techs**	**Cities**	**Prod.**
Move Pruning	59.3%	8500.06	77.88%	2.26	17.38	59.1%	8564.36	79.08%	2.25	17.34
No Move Pruning	56.8%	8458.52	77.46%	2.24	17.16	57.8%	8446.04	78.17%	2.24	17.07

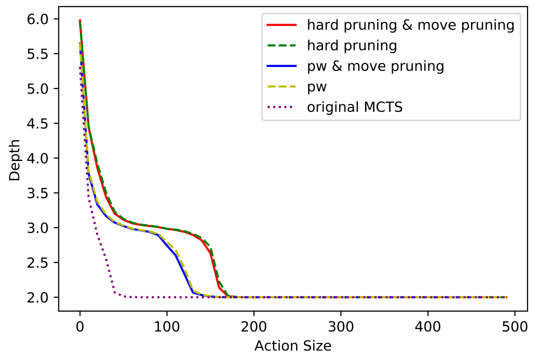


Figure 2: MCTS depth under different action space sizes.

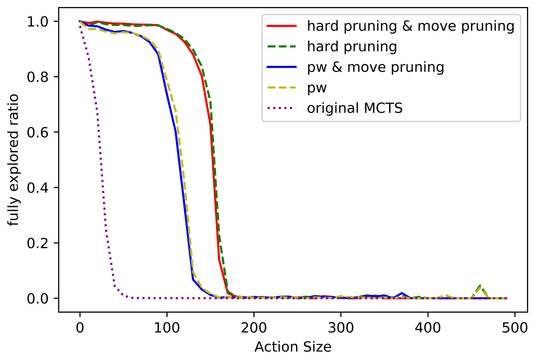


Figure 3: The fully explored rate for the recommended action in MCTS under different action space sizes.

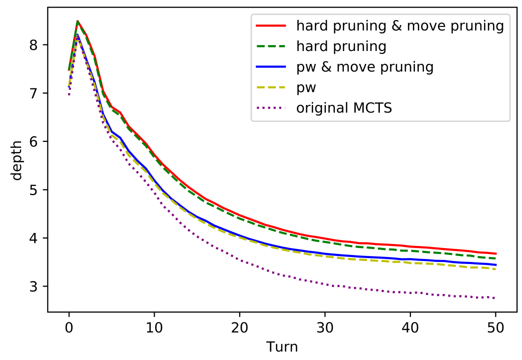


Figure 4: The depth of MCTS in different turns.

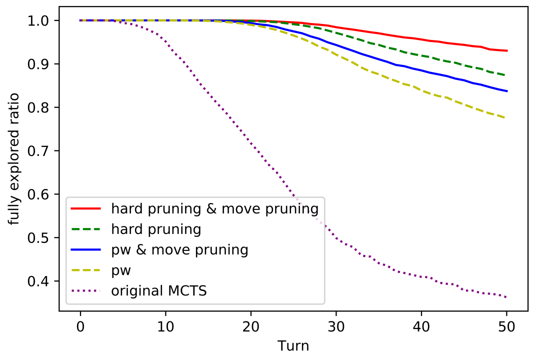


Figure 5: The fully explored rate for the recommended action in MCTS under different turn

space tends to be smaller). The depth of the tree sharply decreases when the branching factor grows. As expected, due to the unpruning mechanism, progressive widening explores more shallow trees than hard pruning, which is most noticeable when the action space size is between 100 and 150. Compared to hard pruning, progressive widening is unable to fully explore the chosen action when the branching factor becomes over 100. Although hard pruning also faces this issue, it happens later than progressive widening. The rate of exploration shown in Figure 3 shows a similar trend: hard pruning is able to delay the decay of children explored further than progressive widening, and both approaches show clearly that they allow the tree to visit a higher proportion of actions from the root than when there is no pruning at all. With no pruning, the rate of default MCTS decays dramatically when the action space size reaches 50. Figures 2 and 3 show the effect of unpruning in progressive widening and how the search tree is explored differently in each case.

Figures 4 and 5 show the influence of the game turn in the depth of MCTS and the fully expanded rate of the recommended action node. Note that, in Tribes, the size of the action space tends to grow (particularly if the game is favourable to the player) with the number of turns. An analysis of this can be found at (Perez et al. 2020). The depth of not-pruned MCTS drops to around 3 when the game turn reaches 30, and the fully explored rate of chosen action starts to drop in turn 10 and ends up with the 40% in turn 50. The pruned versions of MCTS show a similar trend but dropping more slowly than the original MCTS. The depth of modified MCTS is around 4 at the game end, and the fully explored rate of the chosen action is considerably higher than that of the original MCTS in turn 50. Hard pruning decreases slower than progressive widening, and move pruning further slows down the decrease. This shows the significant effects of move pruning in increasing the explored ratio of the chosen action in later turns.

5 Conclusion and future work

This paper focuses on exploring pruning techniques for MCTS in a multi-action strategy game, Tribes, in which players must execute multiple actions in a turn while managing other game aspects such as economy, technology research and resource gathering. Moreover, the size of action space is not fixed and generally increases during the game, resulting in a high and difficult to manage branching factor.

MCTS can handle large branching factors but it struggles when the action space size reaches more than a hundred (Baier and Cowling 2018; Chaslot et al. 2008), which is actually the case in Tribes. This paper shows that using pruning techniques help MCTS deal with this issue. We explored progressive widening, hard pruning and the insertion of domain knowledge pruning for move actions. The inclusion of progressive widening and hard pruning improves the performance of MCTS in a similar fashion, and results are even better when adding the move action pruning.

We propose different avenues of future work. One possibility is to explore other pruning algorithms, like NaiveMCTS (Ontanón 2017) or other types of heuristic pruning (Sephton et al. 2014), or to dive deeper in the use of domain knowledge for pruning. This paper shows that decreasing the proportion of *move* actions increases the performance of the agent. However, several actions might able to benefit from using domain knowledge, such as Level Up or the Build actions. A more general approach would be to use machine or reinforcement learning techniques to learn which actions must be pruned. Another interesting possibility is to explore variants of OEP or evolutionary MCTS that deal with varying number of actions per turn. Finally, there is abundant work on portfilio approaches for strategy games, and exploring synergies between these and different pruning techniques can lead to other ways of dealing with large action space sizes.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Baier, H., and Cowling, P. I. 2018. Evolutionary MCTS for multi-action adversarial games. In *EEE Conference on Computational Intelligence and Games*, 1–8. IEEE.

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* 4(1):1–43.

Chaslot, G. M. J.; Winands, M. H.; HERIK, H. J. V. D.; Uiterwijk, J. W.; and Bouzy, B. 2008. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation (NMNC)* 4(03):343–357.

Childs, B. E.; Brodeur, J. H.; and Kocsis, L. 2008. Transpositions and move groups in monte carlo tree search. In *2008 IEEE Symposium On Computational Intelligence and Games*, 389–395.

Churchill, D., and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in starcraft. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 1–8. IEEE.

Churchill, D., and Buro, M. 2015. Hierarchical portfolio search: Prismata's robust AI architecture for games with large search spaces. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*.

Coulom, R. 2007. Computing "elo ratings" of move patterns in the game of go. *ICGA journal* 30(4):198–208.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning (ICML)*, 273–280. Association for Computing Machinery.

Gelly, S., and Wang, Y. 2006. Exploration exploitation in go: Uct for monte-carlo go.

Justesen, N.; Mahlmann, T.; Risi, S.; and Togelius, J. 2017. Playing multiaction adversarial games: Online evolutionary planning versus tree search. *IEEE Transactions on Games* 10(3):281–291.

Justesen, N.; Uth, L. M.; Jakobsen, C.; Moore, P. D.; Togelius, J.; and Risi, S. 2019. Blood bowl: A new board game challenge and competition for AI. In *2019 IEEE Conference on Games (CoG)*, 1–8.

Justesen, N.; Mahlmann, T.; and Togelius, J. 2016. Online evolution for multi-action adversarial games. In *Applications of Evolutionary Computation*, 590–603. Springer International Publishing.

Midjiwan AB. 2016. *The Battle of Polytopia*.

Ontanón, S. 2017. Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research* 58:665–702.

Perez, D.; Samothrakis, S.; Lucas, S.; and Rohlfshagen, P. 2013. Rolling horizon evolution versus tree search for navigation in single-player real-time games. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, 351–358.

Perez, D.; Hsu, Y.-J.; Emmanouilidis, S.; Khaleque, B. D. A.; and Gaina, R. D. 2020. Tribes: A new turn-based strategy game for AI research. In *Proceedings AIIDE'20*.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2010. On adversarial search spaces and sampling-based planning. In *Twentieth International Conference on Automated Planning and Scheduling*.

Sephton, N.; Cowling, P. I.; Powley, E.; and Slaven, N. H. 2014. Heuristic move pruning in monte carlo tree search for the strategic card game lords of war. In *2014 IEEE Conference on Computational Intelligence and Games*, 1–7. IEEE.

Teytaud, F., and Teytaud, O. 2009. Creating an upper-confidence-tree program for havannah. In *Advances in Computer Games*, 65–74. Springer.