Evolutionary MCTS for Multi-Action Adversarial Games

Hendrik Baier
Digital Creativity Labs
University of York
York, UK
hendrik.baier@york.ac.uk

Peter I. Cowling
Digital Creativity Labs
University of York
York, UK
peter.cowling@york.ac.uk

***Abstract*—*Turn-based multi-action adversarial games* are games in which each player turn consists of a sequence of atomic actions, resulting in an extremely high branching factor. Many strategy board, card, and video games fall into this category, for which the current state of the art is Online Evolutionary Planning (OEP) – an evolutionary algorithm (EA) that treats atomic actions as genes, and complete action sequences as genomes. In this paper, we introduce *Evolutionary Monte Carlo Tree Search* (EMCTS) to tackle this challenge, combining the tree search of MCTS with the sequence-based optimization of EAs. Experiments on the game *Hero Academy* show that EMCTS convincingly outperforms several baselines including OEP and an improved variant of OEP introduced in this paper, at different time settings and numbers of atomic actions per turn. EMCTS also scales better than any existing algorithm with the complexity of the problem.**

***Index Terms*—game tree search, Monte Carlo Tree Search, strategy games**

I. Introduction

Computer programs typically play adversarial games with a form of search, choosing paths to desirable future game states as determined by e.g. a heuristic evaluation function. *Monte Carlo Tree Search* (MCTS) [1], [2] is the state of the art search framework for a variety of classical board games with moderate branching factors of up to a few hundred [3], as well as many card games, video games, and non-game domains [4].

However, most *turn-based multi-action adversarial games* – games in which each turn consists of a sequence of atomic actions, instead of just a single action – have much higher branching factors. This class of games includes board games such as Arimaa and Risk, mobile games such as Battle of Polytopia, and PC games such as Civilization, XCOM, Heroes of Might and Magic, and Into the Breach. A turn in a strategy game could for example consist of moving nine units with ten available actions each, resulting in a branching factor of one billion. Vanilla MCTS cannot handle this complexity, even with the help of various techniques for reducing the effective branching factor. Finding a good action sequence for a single turn, even without considering the next turns, is a challenging search problem in such domains. That is the problem we tackle in this paper. While some of the games in this class feature indeterminism (e.g. Risk) or partial observability (e.g. Civilization), our initial focus here is on deterministic multi-action adversarial games with perfect information.

One possible approach is searching a tree in which each edge represents an atomic action instead of a complete turn, resulting in a much smaller branching factor, but also a much deeper tree (see [5] for a similar trade-off). According to Kozelek [6] and Justesen et al. [7] however, vanilla MCTS is often not able to search the tree of its current turn deeply enough, and focuses too much on optimizing the first actions compared to the last actions. MCTS can be enhanced with pruning techniques that make the search spend the same amount of time on each action [8] – but this still suffers from the problem that MCTS has to find the actions of its turn in a fixed order, so that choices on earlier actions can influence later actions but not vice versa. Justesen et al. therefore proposed a different, tree-less search approach: *Online Evolutionary Planning* (OEP), an evolutionary algorithm that treats atomic actions as genes and complete turns as genomes [9], [7]. By searching over the space of possible next turns with the help of crossover and mutation, it can optimize each action equally and simultaneously. OEP is the current state of the art in multi-action adversarial games.

In this paper, we propose an alternative approach called *Evolutionary MCTS* (EMCTS), combining some of the features of MCTS and evolutionary algorithms. It searches a tree with nodes representing genomes (in multi-action adversarial games: complete turns instead of partial turns, or the states resulting from them), and with edges representing mutations of those genomes (in multi-action adversarial games: mutations of turns instead of additional atomic actions). EMCTS therefore explores the mutation landscape of evolutionary algorithms in a systematic, best-first manner, providing evolution with lookahead search.

We use the same testbed game as Justesen et al. [7] in this paper: the turn-based multi-action adversarial game *Hero Academy*. We also introduce an improved variant of OEP called greedy OEP by transferring some ideas from EMCTS to OEP. EMCTS is then compared to vanilla OEP, greedy OEP, and four other baseline search algorithms including two vanilla MCTS variants specifically designed for Hero Academy, at different CPU time per turn and at different numbers of actions per turn.

This paper begins with a brief review of relevant related work in Section II. Section III describes our testbed, Hero Academy, outlines the baseline algorithms we are comparing, and introduces Evolutionary MCTS. Section IV presents our experimental setup and results, and Section V gives our conclusions and suggests future work.

II. Background and Related Work

This section reviews work on MCTS for very large branching factors, on the current state of the art for multi-action adversarial games – Online Evolutionary Planning – and on previous attempts at combining evolution and tree search.

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [1], [2] is a best-first tree search algorithm based on stochastic simulations for state evaluation, which has been successfully applied to a large variety of games and other tasks [4]. The algorithm typically constructs a search tree with nodes representing game states, and edges representing actions leading from one state to another. In a deterministic game and ignoring transpositions, this can also be seen as a tree in which nodes represent the list of actions that have been applied from the root state to reach their respective state – this view will be helpful later. MCTS begins its search at a root node corresponding to the current game state. It then repeats the following four-phase loop until computation time runs out:

1. In the selection phase, a *selection policy* is used to traverse the tree until an unexpanded action is chosen. The selection policy should balance the exploitation of states with high value estimates and the exploration of states with uncertain value estimates. In this paper, the popular UCB policy is used [10].

2. In the expansion phase, the previously unexpanded action and a node representing its successor state are added to the tree.

3. In the rollout phase, a *rollout policy* is used to play out the remaining part of the simulated game, starting from the state represented by the newly added node. This rollout policy can be uniformly random, but can also profit from heuristic game knowledge. In this paper, we use ϵ-greedy rollouts, which select a random action with probability ϵ, and otherwise follow simple greedy heuristics.

4. In the backpropagation phase finally, the value estimates of all states traversed during the simulation are updated with the result of the finished game.

Several MCTS variants and enhancements have been proposed over time in order to apply MCTS to games with increasingly higher branching factors.

First-play urgency [11] encourages exploitation by providing a value for unvisited child nodes, removing the need for MCTS to visit every child of a node before a selection policy like UCB can be applied. Progressive widening [12] and unpruning [13] approach the branching factor problem in Go by first limiting the number of actions expanded in a new MCTS node, then growing it over time so as to improve value estimates and still guarantee convergence in the limit. For games with much higher branching factors such as real-time strategy (RTS) games, script-based approaches have been developed in order to search over a small number of hand-coded scripts instead of a larger number of atomic actions: Hierarchical Portfolio Search [14] and Script-based UCT [15] fall into this category, as well as the non-MCTS approach of Portfolio Greedy Search [16]. Some previous works have applied MCTS variants to domains with very large or continuous action spaces by making strongly simplifying assumptions such as independence of units in an RTS game [17], or similarity of "close" actions in a physics-based domain [18]. Often, the assumption is made that each unit can perform one action per time step, as is typical for RTS games. In this paper, we do not assume independence of units, do not tie actions to units, and do not assume the existence of predefined policies or scripts. We do however use a heuristic evaluation function – which is hand-coded in our test domain, but could in future work be automatically learned [3].

We are using two specifically adapted variants of MCTS as baselines in our experiments, described in Subsection III-B. The proposed EMCTS is similar to vanilla MCTS in the sense that it uses the same tree search structure of selection, expansion, rollout, and backpropagation, while working on a new, evolution-inspired search space.

B. Online Evolutionary Planning

Evolutionary algorithms (EAs) are a class of optimization algorithms inspired by natural selection that has been used extensively for evolving and training AI agents for games [19], [20]. In the classic, *offline* evolutionary approach, an AI's parameters are evolved using its performance at playing the game as a fitness function. No evolution is applied after the training has finished and the AI is deployed in the game [21], [22], [23], [24].

Online evolution is a newer approach, in which evolutionary algorithms are applied during gameplay. This can take the form of evolving the AI's parameters while it is playing [25]. However, it is also possible to evolve the next *action(s)* to take in the currently running game. *Rolling Horizon Evolutionary Algorithm* (RHEA) [26], [27] for example evolves fixed-length future sequences of actions in a single-player game, which are compared by simulating them and evaluating the resulting game states. When a time limit is reached, the algorithm executes the first action in the best action sequence found, and continues search on action sequences starting from the next time step ("rolling" search horizon).

Online Evolutionary Planning (OEP) [28], [7] is a recent evolutionary approach that is applicable to adversarial multi-action games. It optimizes only the action sequence of the current turn, without lookahead to future turns of the player or the opponent. It can therefore be seen as doing one iteration of RHEA at the beginning of each turn, and with a search horizon of one turn. The best action sequence found is then executed without "rolling" the horizon forward action by action.

OEP begins its search by creating an initial population of genomes, each genome representing a complete turn (fixed-length sequence of actions). Vanilla OEP chooses each of these genomes by repeatedly selecting random actions starting from the current game state. This population is then improved from generation to generation, until a given computation time runs out. Each generation consists of the following four phases:

1. All genomes are translated to their respective phenotypes, the game states resulting from applying their action sequence to the current game state. The fitness of these phenotypes is then evaluated with the help of a static heuristic evaluation.

2. The genomes with the lowest fitness are removed from the population. The proportion of genomes to be removed is a parameter called the *kill rate*.

3. The surviving genomes are each paired with a randomly chosen different genome, and create an offspring through uniform crossover. If this crossover operator leads to an illegal action in the offspring, it is repaired by replacement with an action from the other parent, or otherwise with a random legal action.

4. A proportion of the offspring, determined by a parameter called the *mutation rate*, undergoes mutation. One randomly chosen action of the sequence is changed to another action randomly chosen from all legal actions. If this leads to illegal actions later in the sequence, they are replaced with random legal actions as well.

When the time budget is exhausted, OEP returns the action sequence represented by the current best genome, which is then executed action by action. In the words of Wang et al. "the action selection problem is seen as an optimization problem rather than a planning problem" [29]. This is currently the best-performing approach for turn-based multi-action adversarial games, in particular the test domain of this paper: Hero Academy [7]. It has also been applied to other problems such as micro battles [29] or online build order adaptation [30] in RTS games.

We are using the original OEP, as well as a new improved variant, as baselines in our experiments. The proposed EMCTS is similar to OEP in the sense that in multi-action adversarial games, it also searches a space of complete turns, which are connected to each other through the same mutation operator. It is different in being a tree search algorithm.

C. Hybrids of tree search and evolution

Several other methods have been published that combine ideas from tree search algorithms and evolutionary algorithms.

Gaina et al. [31] experimented in General Video Game AI (GVGAI) with splitting the total search time in two, using MCTS in the first half to provide an initial solution, which is then refined by RHEA in the second half. This was able to outperform RHEA, but not MCTS. Horn et al. [32] hybridized MCTS and RHEA in two different ways: By making use of limited-depth Monte Carlo simulations in the evaluation of RHEA genomes, and by running RHEA and MCTS separately and choosing the best solution found by either of them for execution. EMCTS on the other hand uses a single search algorithm, and a tree search with static state evaluation instead of an evolutionary search with rollouts for evaluation. Lucas et al. [33] used an evolutionary algorithm to improve the rollout policy of MCTS while the search is running. Perez-Liebana et al. [34] adapted a similar method for GVGAI, combining it with a knowledge base to improve reward calculations of given states. While improving MCTS or RHEA performance in various single-player games, the algorithms developed for the GVGAI framework are not straightforwardly applicable to multi-action adversarial games.

For adversarial games, Hong et al. [35] proposed a strategy to evolve paths through a game tree with the help of an evolutionary algorithm. While their approach assumes identical actions to be available in all states at the same search depth, which is not the case in most real-world games including our testbed Hero Academy, it gives an interesting indication for possible future work that could take opponent actions into account.

III. Methods

This section briefly describes the game we use as testbed, lists the search algorithms we are comparing to, and finally presents our approach: Evolutionary MCTS.

A. Test Domain: Hero Academy

Rules. Our test domain is a simplified[1] Java clone [36] of Hero Academy [37], a two-player turn-based tactics game. Players can use a variety of combat units, items, and spells by first drawing them from a card deck onto their hand, and then deploying, casting, or moving them on a battlefield of 9×5 squares. Special squares on this battlefield allow for unit deployment, boost the stats of individual units, or represent a player's two crystals. The game is won by the first player to either eliminate all enemy units, or to destroy both enemy crystals. More details on implementation and rules can be found in [28].

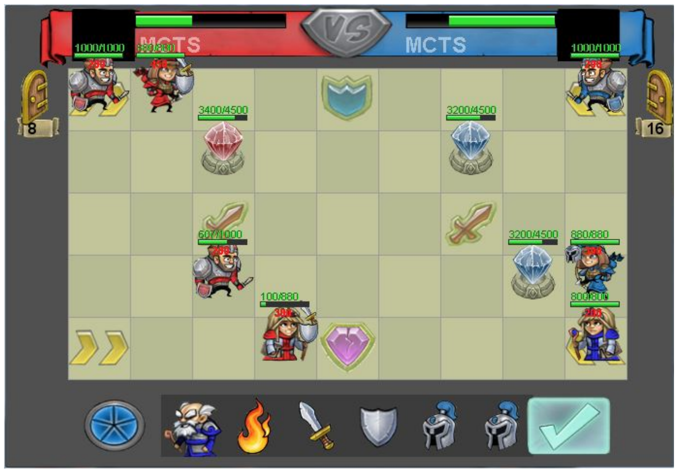

Fig. 1: The testbed game Hero AIcademy. The six symbols at the bottom represent the current player's hand, and the numbers below the doors represent the deck sizes. One of the red player's crystals has already been destroyed.

A central mechanic of the game are the *action points* (AP). For each turn, the player to move receives a number of action points – five in the standard form of the game. Each action

[1]For example, only the "Council" team of units is available.

point can be used for any one atomic action such as deploying a unit from the player's hand onto the battlefield, moving a unit on the battlefield, attacking an enemy unit, healing a friendly unit, and others. The player can spend any number of action points on a single unit, for example by moving it several times. With an average of 30-60 actions available per game state, depending on the playstyle, the full branching factor per turn can be roughly estimated to be $30^5 \approx 2.4 \times 10^6$ to $60^5 \approx 7.8 \times 10^8$. Finding the best sequence of actions for any given turn is therefore a challenging search problem in itself.

The order of cards in the deck as well as the opponent's cards are unknown to the Hero Academy player. However, this paper focuses on the challenge of multi-action turns, ignoring the aspects of hidden information and indeterminism as in [7].

In line with Justesen et al.'s prior work on Hero Academy, we use game knowledge for state evaluation as well as action pruning and ordering:

State evaluation. All algorithms compared in this paper use the same heuristic evaluation function. This function is a linear combination of features such as the current health of individual units, whether they are equipped with certain items, and whether they are standing on special squares. Improving this hand-coded function with machine learning, and testing if our conclusions still hold, could be worthwhile future work.

Action pruning and ordering. All algorithms compared in this paper use a form of hard pruning, removing a number of redundant or provably suboptimal actions from the set of available actions considered in any given state. The two MCTS variants considered as baselines also make use of static action ordering, giving the more promising actions priority in their expansion and rollout phases. The heuristics used for this are simpler and faster than those of the evaluation function.

The interested reader can refer to [28] for a full definition of the heuristic evaluation function and the pruning and ordering strategies.

B. Baseline Approaches

In order to make our results directly comparable to the literature, we are testing our approach against five of the algorithms described in [7]. Four of them are tree search techniques, and one is Online Evolutionary Planning representing the state of the art for Hero Academy.

Greedy Action. The Greedy Action AI chooses the first action of its turn with a simple one-ply search of all legal actions, maximizing the heuristic evaluation of the immediately resulting state. This is repeated for each action point, i.e. for all five actions of the turn.

Greedy Turn. The Greedy Turn AI chooses its actions by attempting a five-ply depth-first search of the entire turn, maximizing the heuristic evaluation of the leaf states resulting from full turns. It uses a transposition table in order to avoid re-visiting states. Actions are ordered for search with the evaluation function, which is especially important since Greedy Turn can usually not exhaustively search the entire turn in the given time limit.

Non-exploring MCTS. This AI is the first MCTS variant adapted for multi-action adversarial games in [7]. It searches a game tree as shown in Figure 2, in which each edge represents an additional action for the turn under consideration (or its application). The opponent's next turn can be reached by a tree deeper than five plies, the number of action points. The selection policy of this MCTS variant is UCB, and the rollout policy deterministically follows the action ordering heuristics. It was found to improve performance when rollouts are just long enough to complete the current turn of the player to act in the leaf node, calling the heuristic state evaluator at the end of the turn for a rollout result. The MCTS exploration factor is set to $C = 0$ in an attempt to grow a deep enough tree (pure exploitation).

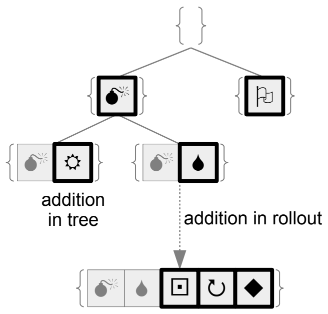

Fig. 2: Tree structure as searched by vanilla MCTS and its variants (non-expl. MCTS, BB-MCTS). Nodes represent partial action sequences, or the states resulting from them. Edges represent the addition of an atomic action to an action sequence, or the application of an atomic action to a state. After each node expansion, a rollout is performed for evaluation. (We use symbols to represent different atomic actions.)

Bridge-burning MCTS (BB-MCTS). This MCTS variant searches the same kind of tree shown in Figure 2. Instead of deterministic rollouts, it uses ϵ-greedy rollouts with $\epsilon = 0.5$, which also only reach to the end of the current turn of the leaf node. Its exploration factor is $C = 1/\sqrt{2}$. In order to grow a deep enough tree for multi-action turns however, it employs a technique called "bridge burning" in [7] – a re-invention of move-by-move search [8]. We are keeping the term "bridge burning" here, as the term "move" is ambiguous in Hero Academy, and also because we are going to generalize the concept of bridge burning to a different kind of tree in the next subsection.

The idea of BB-MCTS is to split the time budget for the current move search into five phases, equal to the number of actions per turn. During each phase, the MCTS search proceeds normally, but at the end of each phase, the most promising action at the root is executed, leading to the root state for the next phase. This can be implemented as the hard pruning strategy shown in Figure 3.

Online Evolutionary Planning. The OEP baseline is as described in Subsection II-B. In our experiments, we use the same parameter settings as suggested in [7]: A population size of 100, a kill rate of 0.5, a mutation rate of 0.1, and uniform crossover and mutation operators.

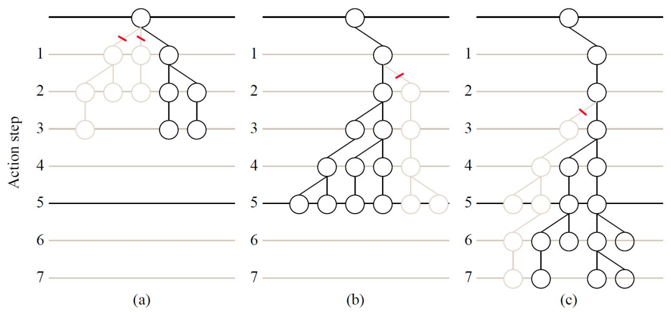

Fig. 3: The “bridge burning” search strategy (illustration adapted from [7]). (a) After phase 1, all branches but the best one are pruned at the root. (b,c) After phases 2, 3, . . . n, pruning is applied at depth 2, 3, . . . n. The partial tree below the best branch is retained.

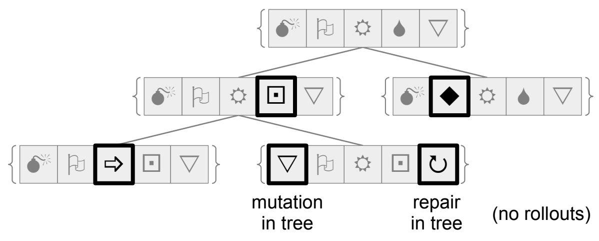

Fig. 4: Tree structure of Evolutionary MCTS. Nodes represent complete action sequences (genomes), or the states resulting from them. Edges represent the mutation of an atomic action within a genome. Repairs can be necessary if those mutations can lead to illegal genomes. After each node expansion, the evaluation function is called instead of a rollout. (We use symbols to represent different atomic actions.)

This algorithm is currently the best-performing approach for multi-action turn-based games such as Hero Academy. Although [7] shows it to be of similar strength to non-exploring MCTS and BB-MCTS in the standard form of the game with 5 action points per turn, OEP was shown to scale better to the tougher challenges of Hero Academy using 10 AP or more. Our experiments include those exponentially more complex variants as well.

C. *Evolutionary MCTS*

This subsection proposes our new search algorithm, *Evolutionary MCTS* or *EMCTS*, as applied to playing multi-action turn-based adversarial games. It combines the tree search of MCTS with the genome-based approach of evolutionary algorithms.

Instead of the vanilla MCTS tree seen in Figure 2, EMCTS builds a tree as shown in Figure 4. Instead of starting from an empty turn in the root, EMCTS starts from a complete sequence of five (or more, depending on the domain) actions – just like the genomes of OEP. Instead of growing a tree that adds one action to the current sequence with every edge, EMCTS grows a tree that mutates the current sequence with every edge – using the same mutation operator as OEP. And instead of using rollouts to complete the current turn and then evaluating it as our MCTS baselines do, we simply evaluate the solutions at the leaf nodes[2]. Backpropagation is unchanged.

EMCTS does not apply mutations randomly, but can choose exactly which action in the sequence to mutate and which other legal action to mutate it to[3]. While OEP turned the planning of the action sequence into an optimization problem, EMCTS thus takes the evolutionary optimization of the sequence and turns it back into a planning problem. It can be seen as tree search, but it can also be seen as a systematic exploration of the mutation landscape of OEP, giving evolution the benefit of lookahead.

Two questions need to be answered to fully flesh out EMCTS. First, where does the root sequence come from? EMCTS needs a *starting solution* to its search, just like EAs such as OEP need a starting population of solutions. Different approaches are possible – in this paper, we are using the Greedy Action algorithm described above for a quick and greedy initialization of the root. Second, what happens when a mutation leads to an illegal action sequence? We could filter these out by simulating every possible mutation in advance, but that would be computationally expensive. Instead, like OEP we are taking the classic evolutionary algorithm approach of using a *repair strategy* – in this paper, we are using the Greedy Action AI for repairs as well whenever necessary.

Note that the use of Greedy Action does not introduce additional heuristic knowledge, as all algorithms compared in this paper are working with the same evaluation function. However, we noticed that like EMCTS, OEP can also be significantly improved by using a Greedy Action repair policy instead of a random repair policy. This results in higher quality repairs on average. And just like EMCTS profits from a greedy root genome, OEP can profit from filling 20% of the starting population with Greedy Action sequences instead of random ones[4]. This kick-starts the search with higher-quality starting solutions. We are calling this new variant *greedy OEP* here, as opposed to *vanilla OEP* with random repairs and a purely random starting population as described in [28], [9], [7], and include it in our experiments for a fair comparison.

Finally, EMCTS results in an even larger branching factor than the vanilla MCTS variants. While the branching factor in Hero Academy games between the MCTS baselines was measured to be between 30 and 40, the branching factor of the mutation tree of EMCTS is about 30 *per action point* – so around 150 for the standard settings of the game with five action points. We found that an effective way of dealing with this is “bridge burning”, just as applied to the regular MCTS tree by BB-MCTS. Instead of executing the most promising action at the root after every search phase like BB-MCTS, EMCTS executes the most promising mutation at the root after each phase. The number of bridge burning phases, of successive

[2]Evaluating at the leaf nodes is a well-known MCTS variant that was successfully employed for example in AlphaGo Zero and AlphaZero [3].

[3]No crossover operator is used.

[4]This performed better than filling 1%, 10%, and 50% of the starting population with Greedy Action sequences.

searches and prunings/mutations, is the only parameter of EMCTS we tuned (see the following section). The MCTS exploration factor was set to $C = 0$. The selection policy is UCB as in the other MCTS variants.

IV. Experiments and Results

This section describes our experimental setup for testing the proposed Evolutionary MCTS, as well as the results.

A. Experimental Setup

We tested EMCTS in Hero Academy against Greedy Action, Greedy Turn, non-exploring MCTS, BB-MCTS, and vanilla OEP as proposed in [7], as well as the improved greedy OEP as proposed in the previous section. All comparisons were performed on the standard settings of the game with 5 action points per turn, but also with altered rules allowing 10 AP or even 15 AP per turn[5]. This increases the complexity of a single turn exponentially, but gives a stronger indication of generalizability to other games which can have higher numbers of possible actions per turn. Furthermore, all comparisons were done at different time budgets of 200 ms per turn, 1 second per turn, and 5 seconds per turn. Each comparison consisted of 400 games, with EMCTS playing 200 games as the first player and 200 games as the second player. The map used is shown in Figure 1. Games that had no winner after 200 turns were counted as draws, i.e. half a win for each player.

All algorithms used the parameter settings described in Section III. The number of "bridge burning" phases for EMCTS was determined in preliminary experiments and set to 20 for 200 ms, 40 for 1 second, and 100 for 5 seconds per turn time controls. The number of phases for BB-MCTS were identical to the AP per turn, since it searches the type of tree shown in Figure 2 and does not profit from deeper searches. As no other algorithm was modified based on the AP per turn, EMCTS was also not specifically tuned for different AP.

B. Results

Table I shows the performance of the proposed Evolutionary MCTS against the five baselines and the improved greedy Online Evolutionary Planning.

EMCTS is significantly stronger than all baselines (Greedy Action, Greedy Turn, BB-MCTS, non-expl. MCTS, and vanilla OEP) at all time controls and all numbers of action points per turn. Its relative strength increases with the complexity of the search problem as measured in action points per turn. The newly proposed greedy OEP is a dramatic improvement over vanilla OEP as described in [7], but still significantly weaker than EMCTS at all action points at 200 ms per turn, and at all action points except for the lowest setting (5) at 1 s and 5 s per turn, where the two algorithms perform similarly. The results therefore show that Evolutionary MCTS is highly effective at a variety of time controls, and scales better with the complexity of the domain than all other tested approaches.

[5]20 or even 25 AP as in [7] were not included. As the authors noted, such high numbers of AP make it possible to win the game within very few turns, and make the winner very strongly depend on who gets the first turn. Strength differences between AIs are therefore harder to measure. More significant rule changes would have to be made to balance the game with such high AP.

Opponent	Action points per turn		
	5	10	15
200 ms per turn			
Greedy Action [7]	87.6%***	97.8%***	98.3%***
Greedy Turn [7]	96.9%***	100.0%***	100.0%***
BB-MCTS [7]	68.6%***	88.8%***	93.0%***
non-expl. MCTS [7]	74.5%***	91.8%***	92.0%***
vanilla OEP [7]	77.8%***	92.0%***	94.8%***
greedy OEP [this paper]	60.6%**	59.5%**	65.3%***
1000 ms per turn			
Greedy Action [7]	88.1%***	98.5%***	99.3%***
Greedy Turn [7]	92.8%***	99.0%***	100.0%***
BB-MCTS [7]	67.1%***	90.3%***	94.5%***
non-expl. MCTS [7]	65.5%***	93.5%***	97.3%***
vanilla OEP [7]	70.5%***	84.8%***	91.0%***
greedy OEP [this paper]	52.5%	58.8%*	61.8%***
5000 ms per turn			
Greedy Action [7]	91.9%***	99.0%***	99.8%***
Greedy Turn [7]	78.1%***	98.8%***	100.0%***
BB-MCTS [7]	67.0%***	90.3%***	94.8%***
non-expl. MCTS [7]	56.9%*	94.8%***	98.5%***
vanilla OEP [7]	69.0%***	80.3%***	87.5%***
greedy OEP [this paper]	51.4%	59.0%*	61.3%**

TABLE I: Win rates of EMCTS vs. all baselines at different time controls. 400 games per data point. Asterisks indicate significantly stronger play by EMCTS: $*p < 0.05$, $**p < 0.01$, $***p < 0.001$

Note that there is a tradeoff for "bridge burning" EMCTS between doing more phases (pruning all but the best mutation and continuing search from there), and having more time for each phase (to identify the best mutation). With search time, both the optimal number of phases as well as the optimal time per phase seem to increase. The settings found to perform best in our experiments have such high numbers of phases, and such little time for them, that EMCTS could be seen as a type of local search [38] or $(1, \lambda)$-Evolution Strategy [39]. At longer time settings though, deeper trees can form, and EMCTS turns into a new kind of genome-based planning, or evolution with lookahead. These connections are worth exploring more deeply in future work.

V. Conclusions and Future Work

This paper proposes a new algorithm for playing turn-based adversarial games, where each turn consists of a sequence of multiple actions. Such action sequences, common in strategy games, lead to the challenge of extremely large branching factors per turn. This is difficult to handle even for selective tree search methods such as MCTS, which typically search a tree of atomic actions, and specifically developed evolutionary algorithms such as OEP, which optimize entire action sequences.

Our new algorithm, called *Evolutionary MCTS* (EMCTS), is based on the idea of combining the tree search of MCTS with the sequence-based optimization of evolutionary algorithms. Instead of searching a vanilla MCTS tree, EMCTS searches a tree in which each edge mutates one action in a complete action sequence. Experiments on the game Hero Academy show that EMCTS convincingly outperforms several baselines from the literature, including the state of the art OEP and an improved variant of OEP introduced in this paper, at different time settings and numbers of actions per turn. EMCTS also scales better than any existing algorithm with the complexity of the problem. It is therefore the currently strongest algorithm for playing Hero Academy, and a promising candidate for other turn-based multi-action games such as Civilization, XCOM, Heroes of Might and Magic, or Into the Breach.

Several directions appear interesting for future work. First, the comparison between Evolutionary MCTS and the baseline algorithms could be deepened, including experiments with different initialization and repair strategies, different evaluation functions, more careful tuning of algorithm parameters such as OEP's population size, mutation rate, and kill rate, and possible improvements to MCTS methods such as stronger rollout policies. Second, various aspects of EMCTS could be considered in more detail, such as speed optimizations – it currently only evaluates roughly 20% as many action sequences per second as OEP. Mutations for expansion could for example be generated lazily in the tree nodes, and various MCTS enhancements could be used to improve their ordering. Third, the performance of EMCTS in other games could be tested, such as strategy games with longer matches and larger numbers of units. We are planning to apply it to Battle of Polytopia, a mobile turn-based strategy game in which armies can grow to 15 to 20 units or more in the late game. Unlike Hero Academy, Battle of Polytopia does not allow for any unit to move more than once per turn; however, additional complexity arises from units whose actions themselves consist of several atomic parts such as moving, attacking, and retreating. An interesting challenge for the application to commercial games is that the existence of a heuristic state evaluation function cannot generally be assumed, requiring machine learning approaches. Just like OEP, EMCTS could also be generalized to other problems such as micro battles [29] or online build order adaptation [30] in RTS games. In the former scenario, the genomes would consist of a list of scripts representing simple policies assigned to each unit, instead of a list of atomic actions for the player. In the latter scenario, the genomes would be candidate build orders, i.e. fixed-length sequences of future units and buildings to construct. Fourth, the problem of considering future actions of the opponent has not been tackled successfully yet, neither by OEP nor by EMCTS. Generalizing to larger classes of games will also require dealing with indeterminism and partial observability. And last but not least, the algorithmic similarities between Evolutionary MCTS and certain local search algorithms and evolutionary algorithms deserve further study, in order to further explore the idea of "evolution with lookahead".

References

[1] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *17th European Conference on Machine Learning, ECML 2006*, ser. Lecture Notes in Computer Science, vol. 4212, 2006, pp. 282–293.

[2] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo Tree Search," in *5th International Conference on Computers and Games, CG 2006. Revised Papers*, ser. Lecture Notes in Computer Science, vol. 4630, 2007, pp. 72–83.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *CoRR*, vol. abs/1712.01815, 2017.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez-Liebana, 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, 2012.

[5] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 241–257, 2012.

[6] T. Kozelek, "Methods of MCTS and the Game Arimaa," Master's thesis, Faculty of Mathematics and Physics, Charles University, Prague, 2009.

[7] N. Justesen, T. Mahlmann, S. Risi, and J. Togelius, "Playing Multi-Action Adversarial Games: Online Evolution versus Tree Search," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017, in print.

[8] M. P. D. Schadd, M. H. M. Winands, M. J. W. Tak, and J. W. H. M. Uiterwijk, "Single-Player Monte-Carlo Tree Search for SameGame," *Knowledge-Based Systems*, vol. 34, pp. 3–11, 2012.

[9] N. Justesen, T. Mahlmann, and J. Togelius, "Online Evolution for Multi-action Adversarial Games," in *19th European Conference on Applications of Evolutionary Computation (EvoApplications 2016)*, ser. Lecture Notes in Computer Science, G. Squillero and P. Burelli, Eds., vol. 9597. Springer, 2016, pp. 590–603.

[10] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-Time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[11] S. Gelly and Y. Wang, "Exploration Exploitation in Go: UCT for Monte-Carlo Go," in *Neural Information Processing Systems Conference (NIPS), On-line trading of Exploration and Exploitation Workshop*, 2006.

[12] R. Coulom, "Computing elo ratings of move patterns in the game of Go," in *Computer Games Workshop*, 2007.

[13] G. M. J. B. Chaslot, M. H. M. Winands, J. v. d. Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.

[14] D. Churchill and M. Buro, "Hierarchical Portfolio Search: Prismata's Robust AI Architecture for Games with Large Search Spaces," in *11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015*, A. Jhala and N. Sturtevant, Eds. AAAI Press, 2015, pp. 16–22.

[15] N. Justesen, B. Tillman, J. Togelius, and S. Risi, "Script- and cluster-based UCT for StarCraft," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014*, 2014, pp. 1–8.

[16] D. Churchill and M. Buro, "Portfolio Greedy Search and Simulation for Large-scale Combat in StarCraft," in *IEEE Conference on Computational Intelligence in Games, CIG 2013*. IEEE, 2013, pp. 1–8.

[17] S. Ontañón, "The Combinatorial Multi-Armed Bandit Problem and Its Application to Real-Time Strategy Games," in *9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13*, G. Sukthankar and I. Horswill, Eds. AAAI, 2013.

[18] T. Yee, V. Lisý, and M. H. Bowling, "Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, S. Kambhampati, Ed. IJCAI/AAAI Press, 2016, pp. 690–697.

[19] S. M. Lucas and G. Kendall, "Evolutionary Computation and Games," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 10–18, 2006.

[20] S. Risi and J. Togelius, "Neuroevolution in Games: State of the Art and Open Challenges," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 1, pp. 25–41, 2017.

[21] N. Cole, S. J. Louis, and C. Miles, "Using a genetic algorithm to tune first-person shooter bots," in *2004 Congress on Evolutionary Computation (CEC 2004)*, 2004, pp. 139–145.

[22] G. M. J. B. Chaslot, M. H. M. Winands, I. Szita, and H. J. van den Herik, "Cross-entropy for Monte-Carlo Tree Search," *ICGA Journal*, vol. 31, no. 3, pp. 145–156, 2008.

[23] A. M. Alhejali and S. M. Lucas, "Using genetic programming to evolve heuristics for a Monte Carlo Tree Search Ms Pac-Man agent," in *2013 IEEE Conference on Computational Intelligence and Games, CIG 2013*, 2013, pp. 1–8.

[24] A. Benbassat and M. Sipper, "Evomcts: A scalable approach for general game learning," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 382–394, 2014.

[25] C. F. Sironi and M. H. M. Winands, "On-line parameter tuning for Monte-Carlo Tree Search in General Game Playing," in *Computer Games Workshop*, 2017, pp. 75–95.

[26] D. Perez-Liebana, S. Samothrakis, S. M. Lucas, and P. Rohlfshagen, "Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games," in *2013 Genetic and Evolutionary Computation Conference, GECCO '13*, C. Blum and E. Alba, Eds. ACM, 2013, pp. 351–358.

[27] R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *20th European Conference on Applications of Evolutionary Computation, EvoApplications 2017*, ser. Lecture Notes in Computer Science, G. Squillero and K. Sim, Eds., vol. 10199, 2017, pp. 418–434.

[28] N. Justesen, "Artificial Intelligence for Hero Academy," Master's thesis, IT University of Copenhagen, 2015.

[29] C. Wang, P. Chen, Y. Li, C. Holmgard, and J. Togelius, "Portfolio Online Evolution in StarCraft," in *Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-16*, 2016, pp. 114–120.

[30] N. Justesen and S. Risi, "Continual Online Evolutionary Planning for In-game Build Order Adaptation in StarCraft," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017*, P. A. N. Bosman, Ed. ACM, 2017, pp. 187–194.

[31] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing," in *2017 IEEE Congress on Evolutionary Computation, CEC 2017*. IEEE, 2017, pp. 1956–1963.

[32] H. Horn, V. Volz, D. P. Liebana, and M. Preuss, "MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation," in *IEEE Conference on Computational Intelligence and Games, CIG 2016*. IEEE, 2016, pp. 1–8.

[33] S. M. Lucas, S. Samothrakis, and D. Perez-Liebana, "Fast Evolutionary Adaptation for Monte Carlo Tree Search," in *17th European Conference on Applications of Evolutionary Computation, EvoApplications 2014*, ser. Lecture Notes in Computer Science, A. I. Esparcia-Alcázar and A. M. Mora, Eds., vol. 8602. Springer, 2014, pp. 349–360.

[34] D. Perez-Liebana, S. Samothrakis, and S. M. Lucas, "Knowledge-based Fast Evolutionary MCTS for General Video Game Playing," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014*, 2014, pp. 1–8.

[35] T. Hong, K. Huang, and W. Lin, "Adversarial Search by Evolutionary Computation," *Evolutionary Computation*, vol. 9, no. 3, pp. 371–385, 2001.

[36] Niels Justesen, "Hero AIcademy." [Online]. Available: https://github.com/njustesen/hero-aicademy

[37] Robot Entertainment, "Hero Academy." [Online]. Available: http://www.robotentertainment.com/games/heroacademy/

[38] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

[39] H. Beyer and H. Schwefel, "Evolution Strategies - A Comprehensive Introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.