Improving the Performance of MCTS-Based μRTS Agents Through Move Pruning

Abdessamed Ouessai
Dept. Computer Sciences
University of Mascara
Mascara, Algeria
abdessamed.ouessai@univ-mascara.dz
ORCID: 0000-0002-1305-8602

Mohammed Salem
Dept. Computer Sciences
University of Mascara
Mascara, Algeria
salem@univ-mascara.dz
ORCID: 0000-0001-7052-5978

Antonio M. Mora
Dept. Signal Theory, Telematics and Communications
ETSIIT-CITIC, University of Granada
Granada, Spain
amorag@ugr.es
ORCID: 0000-0003-1603-9105

***Abstract*—The impressive performance of Monte Carlo Tree Search (MCTS) based game-playing agents in high branching-factor domains such as Go, motivated researchers to apply and adapt MCTS to even more challenging domains. Real-time strategy (RTS) games feature a large combinatorial branching factor and a real-time aspect that pose significant challenges to a broad spectrum of AI techniques, including MCTS. Various MCTS enhancements were proposed, such as the combinatorial multi-armed bandit (CMAB) based sampling, state/action abstractions, and machine learning. In this paper, we propose to employ move pruning as a way to improve the performance of MCTS-based agents in the context of RTS games. We describe a class of possibly detrimental player-actions and propose several pruning approaches targeting it. The experimentation results in μRTS indicate that this could be a promising direction.**

***Index Terms*—Monte Carlo Tree Search, Move Pruning, Real-Time Strategy, Game AI, μRTS**

I. Introduction

The complexity of real-time strategy (RTS) games, from an AI perspective, originates from the combinatorial structure of their state and decision spaces. In comparison with classic benchmark games such as Chess or Go, the dimensionality of both state and decision spaces in an RTS game is many orders of magnitude higher [22]. Instead of controlling a single unit in a turn-based fashion, RTS players control multiple units simultaneously in real-time, and usually in a much larger board (map) size. Moreover, the branching factor in an RTS game grows exponentially with the increase in the number of units positioned on the map.

Due to the game's complexity, conceiving a human-challenging RTS game-playing agent is a difficult task to undertake. The predominant approach followed by researchers and practitioners in the domain is to decompose the task into manageable subtasks targeting various degrees of abstraction. Most commonly, an RTS agent combines high-level strategic components and low-level tactical components. Such decomposition is inspired by the way human players interweave micro- and macro-management, and it is shown to be effective by numerous implementations [3], [15], [17].

Holistic search-based approaches such as Monte Carlo Tree Search (MCTS) [5] enjoyed a remarkable success in computer Go, as demonstrated by AlphaGo [25]. However, in RTS games, MCTS-based agents struggle with the enormous decision space and fail to scale suitably when the branching factor grows past a certain threshold. Such downside restricts MCTS applicability to limited scenarios, such as tactical planning or small maps. Abstracting the decision space is a tried and tested technique for scaling MCTS-based agents to larger scenarios [22], at the expense of sacrificing tactical performance due to the coarser actions considered.

We propose an approach to increase the performance and scalability of search-based techniques, particularly MCTS-based, by pruning unnecessary and detrimental player-actions from the decision space of an RTS game. We inspect the low-level structure of the search space and identify detrimental player-actions through domain knowledge. Next, we apply multiple hard-pruning approaches to remove those player-actions during the search. The goal is to reduce the branching factor and explore more promising player-actions. Our approach targets a class of player-actions we identify as Inactive Player-Actions (IPAs) because they tend to keep at least one unit in an inactive state, which can be problematic. The experiments' results using UCT (Upper Confidence bounds for Trees) and NaïveMCTS in μRTS show a considerable performance gain relative to the map's size.

The rest of this paper is organized as follows: Section II reviews some background knowledge about RTS games, μRTS and MCTS. Section III presents the related state of the art and Section IV describes IPAs and the proposed move pruning approaches. Experimental results are presented and discussed in Section V, and Section VI ends the paper with some conclusions and future perspectives.

II. Background

A. Real-Time Strategy Games

A sub-genre of strategy video games, real-time strategy games usually simulate a warfare situation where each side of the game is given control over a military base and is tasked

This work has been supported in part by projects B-TIC-402-UGR18 (FEDER and Junta de Andalucía), RTI2018-102002-A-I00 (Ministerio Español de Ciencia, Innovación y Universidades), projects TIN2017-85727-C4-1-2-P (Ministerio Español de Economía y Competitividad), and TEC2015-68752 (also funded by FEDER).

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

with collecting resources and recruiting troops. To emerge victorious, the player must fully destroy his opponent's forces. RTS games progress in real-time, signifying that players may act simultaneously under a very short decision cycle, and the effect of executing an action is not necessarily immediate. Usually, an RTS game is played from a top-down perspective over a large grid-based map, covered by a fog-of-war layer reducing observability and increasing the game's difficulty and complexity. Moreover, the execution of a unit-action can be influenced by some stochastic parameters, introducing non-determinism to the mix. Players control their units by issuing unit-actions to each. A player-action is the combination of unit-actions issued simultaneously in the same game cycle.

A typical RTS game is defined as a zero-sum, multi-player, non-deterministic game with imperfect information. The size of an RTS state space and branching factor, as estimated in a typical StarCraft setting [22], reaches 10^{1685} possible states and 10^{50} possible actions at a decision point. In contrast, Chess and Go possess a state-space estimate of 10^{47} and 10^{171}, respectively, with an average branching factor equaling 36 in Chess and 180 in Go. Such proportions predict a difficult task for an RTS game-playing AI. As per [19], an RTS game can be defined as a tuple $G = (S, A, P, \tau, L, W, s_{init})$, where:

- S : The set of all possible states (state space).
- A : The set of player-actions (decision space).
- P : The set of players. $P = \{max, min\}$ for two players.
- $\tau : S \times A \times A \rightarrow S$: The state transition function. It takes a game state at time t and a player-action for each player, then returns a new game state at time $t + 1$.
- $L : S \times A \times P \rightarrow \{true, false\}$: Determines the legality of a player-action in a given state for a specific player.
- $W : S \rightarrow P \cup \{ongoing, draw\}$: Determines the winner, if any, or whether the game is a draw or still ongoing.
- $s_{init} \in S$: The initial state.

B. μRTS

Conducting AI research on commercial RTS games can be a daunting experience since most games do not offer a suitable API for this purpose. To mitigate this shortcoming several independent solutions were developed, such as ORTS, the WarCraft port, Wargus, and the unofficial StarCraft interface, BWAPI. Much later, an official API and toolset for StarCraft II were made available in a collaborative effort between Blizzard and DeepMind [28]. Additionally, several independent RTS AI research platforms have emerged, such as μRTS [19], ELF [27], and DeepRTS [1].

μRTS is a stripped-down RTS game simulator designed for AI research, featuring all the challenging aspects of an RTS game in a minimalistic form, and including an efficient forward model useful for implementing simulation-based search approaches. Figure 1 shows a typical μRTS match. Each player controls two types of structures (Base and Barracks) and four types of mobile units (Worker, Light, Ranged, and Heavy). The Base produces Worker units, and the Barracks produce assault units, in exchange for resources harvested by Workers. The game map consists of an arbitrary-sized 2D grid.

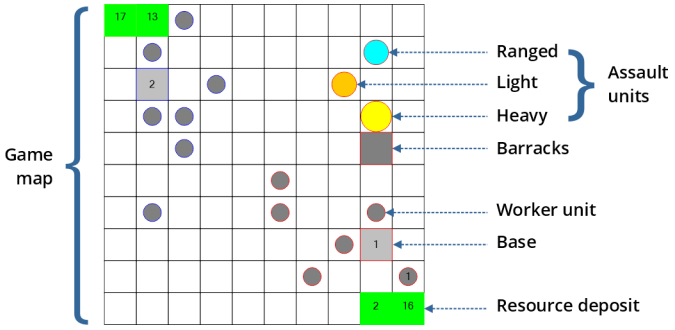

Fig. 1. A μRTS match. A player's units can be distinguished by the color of their outline. The number of resources held by a unit is displayed on it.

Multiple interesting online adversarial planning approaches were conceived in μRTS, as detailed in [23].

Since 2017, a μRTS AI competition is held as part of the IEEE Conference on Games (CoG, formerly CIG) [20]. The competing agents participate in a round-robin tournament, in at least one of the three tracks: full-observability, partial-observability, and non-determinism (dropped in the 2020 edition). Our proposed approach can be useful in any track.

C. Monte Carlo Tree Search (MCTS)

The goal of an RTS game-playing agent is to compute an optimal player-action $a \in A$ at each decision cycle t where the agent can act. MCTS is a sampling-based search framework applicable to sequential decision problems in large decision spaces unapproachable to systematic search techniques. MCTS estimates the value of actions, sampled using a tree policy, through random simulations. The MCTS algorithm iteratively constructs a game tree following a 4-step process at each iteration. The algorithm can be halted anytime to obtain a decision. An MCTS iteration proceeds as follows:

1) *Selection:* Select a node with unexplored children following a tree policy.
2) *Expansion:* Create and attach a new child node.
3) *Simulation:* Start a simulation (playout) from the new node following a playout policy.
4) *Backpropagation:* Backpropagate the simulation's results starting from the new node up to the root node.

The most visited decision is usually the one returned. Given enough computation budget and a proper exploration/exploitation balance in the tree policy, MCTS is guaranteed to find the minimax solution in the limit [13]. UCT uses UCB1 as a tree policy, treating the selection phase as a multi-armed bandit (MAB) problem [5]. Although remarkably successful in Go, UCT does not perform as well in RTS games, due to the rapid growth in the branching factor with the increase in the unit count. NaïveMCTS was designed to better handle the combinatorial search space in RTS games by formulating the selection phase as a combinatorial MAB (CMAB) [19]. NaïveMCTS builds on a naïve sampling approach based on a naïve assumption that considers the reward estimate of a player-action as the sum of the reward estimates of the underlying unit-actions. In our experiments, we tested our approach on both UCT and NaïveMCTS.

III. State of the Art

Dealing with the enormous RTS decision space in the context of MCTS is an open problem continuously receiving contributions. By treating the selection phase as a CMAB, NaïveMCTS effectively adapts MCTS to combinatorial search spaces. Nevertheless, the decision space remains the same, and the algorithm still suffers from high dimensionality. Downsizing the search space's dimensionality is usually done through action abstraction or machine learning.

Abstracting the search space through expert-authored scripts is an effective way to considerably reduce the branching factor. Instead of searching in the low-level player-actions space, Justesen et al [12] adapted UCT to search in the space of player-actions suggested by scripts. NaïveMCTS was similarly adapted by Moraes et al [16] using asymmetric abstractions [15] to search in multiple levels of abstraction. Puppet Search [4] uses UCT to search in the space of choice points inserted in scripts. Guided NaïveMCTS (GNS) [30] biases the selection phase to consider scripted actions first. Other search algorithms were also combined with action abstraction, such as local search in Portfolio Greedy Search (PGS) [6] and minimax in Adversarial Hierarchical Task Networks (AHTN) [21].

Machine learning can be used to guide node selection by learning a tree policy from expert traces. AlphaGo [25] employed a CNN-based policy network, trained from a large database of expert Go replays, and improved via a reinforcement learning phase. Later work on AlphaZero [26] fully discarded expert knowledge. Ontañón informed node selection using a learned Bayesian model in InformedMCTS [18]. Yang and Ontañón [29] later demonstrated the effectiveness of the C4.5 classifier for such tasks, due to its speed and accuracy.

If we adjust our perspective, all the aforementioned approaches can be regarded as move pruning approaches [31]. By focusing on a set of promising expert-based player-actions, these approaches effectively prune the search space of all the remaining player-actions, significantly reducing the branching factor. Such practice can also become unsafe and prone to exploitation, due to the coarser player-actions considered, resulting in a loss of tactical performance. To address this issue, several approaches combining low- and high-level search have emerged, such as [3], [17] and [16].

Directly pruning the player-actions responsible for weak performance can be an alternative approach towards focusing the search on promising actions, without compromising tactical strength. In the context of Chess [9] and Shogi [10], several forward pruning methods such as Null-move pruning and futility pruning were utilized to reduce the branching factor and enhance $\alpha\beta$ search. In Go, a domain-dependent pruning approach was implemented in UCT [11], exploiting territory information. Similar MCTS improvements were applied in Hex [2], Havannah, [7] and DeadEnd [8]. In video-games, Sephton et al [24] enhanced MCTS by applying a knowledge-based move pruning approach for the strategic card game, Lords of War.

We propose a domain knowledge-based hard-pruning approach for MCTS agents in RTS games, targeting a specific type of player-actions prevalent in all RTS games.

TABLE I
The unit-action types available for each unit-type in μRTS

	Move	Attack	Harvest	Return	Produce	Wait
Worker	•	•	•	•	•	•
Light	•	•				•
Ranged	•	•				•
Heavy	•	•				•
Base					•	•
Barracks					•	•

IV. Move Pruning

We propose to act directly on the decision space and hard-prune a subset of decisions we deem irrelevant and/or detrimental to the performance of MCTS. By doing so, MCTS will be freed from sampling those decisions and simulating their outcomes. The recovered computation time will be spent on exploring more relevant and significant decisions, which would improve the playing strength and scalability of MCTS.

As a first attempt, we chose to focus on player-actions having the highest chance of misleading search and negatively impacting the playing strength. Out of these player-actions, we believe Inactive Player-Actions (IPAs) naturally come first. Thus, we implemented several pruning approaches that keep a predefined number (fixed or relative) of those player-actions and prune the remaining. We will briefly discuss the structure of RTS player-actions in the next section and then define IPAs.

A. Unit-Actions and Player-Actions

In a typical RTS game, each unit type can execute a distinct set of actions known as unit-actions. Table I enumerates the unit-action types executable by each unit-type in μRTS. The Worker unit-type is the most versatile, followed by assault units (Light, Ranged and Heavy) and structures (Base and Barracks). The attributes of a unit-type define the effect of its unit-actions. For instance, the damage attribute controls how much damage a unit-type causes when executing the Attack unit-action. Thus, even for common unit-actions, each unit-type may behave differently. All unit-action types, except Wait, require an argument that determines the target of the action. The Wait unit-action type requires a numeric argument specifying the number of cycles ahead at which the unit must remain inactive. Wait is the only unit-action type unaffected by unit attributes and executable by all unit-types.

A player-action $p \in A$ issued to n units at a given game cycle can be regarded as a tuple, $p = (a_1, a_2, \ldots, a_n)$, where each component a_i is a unit-action issued to the i-th unit. Given the average number of legal unit-actions available to each unit, m, the number of all possible player-actions or the branching factor, b, can be estimated as $b = m^n$. We seek to lower b by finding ways to decrease m without negatively impacting the playing strength.

B. Inactive Player-Actions

We define an Inactive Player-Action (IPA) as a player-action having at least one Wait (inactive, idle, no-op) unit-action as a component. Being the most prevalent non-critical unit-action, Wait unit-actions make for a good pruning target. The Wait unit-action is continuously available to all units, regardless of their situation. Thus, it strongly contributes to the inflation of the search space. Nonetheless, Wait unit-actions can be advantageous for a unit, usually in the following situations:

- *Trapped unit:* No active unit-action is possible. The unit is caught in a situation where all possible unit-actions are illegal. Waiting for a predefined duration is the only option to choose in hopes the situation is resolved.
- *Tactical waiting:* The unit anticipates for a chance to execute a high-value unit-action. Here, the unit expects a sub-optimal action by an opponent unit (via lookahead) and chooses to Wait in anticipation for it. Executing the high-value action happens afterward. This behavior is frequently observed in tactical skirmishes.

Although potentially useful, Wait unit-actions can also have a devastating effect on the playing strength if improperly chosen. According to our observations, it is not unlikely for a search-based agent (MCTS or otherwise) to assign a Wait unit-action to a unit in a situation where better options exist. In such cases, doing nothing can be the worst decision possible. We identify three disadvantageous situations where Waiting cannot be a sound decision:

- *Waiting in front of opportunity:* Here, the unit can seize an immediate opportunity, such as Harvest resources, Return harvested resources, or safely remove an opponent unit. Instead, the unit is assigned Wait.
- *Waiting in face of danger:* The unit is facing an immediate danger and holds the necessary options to avoid it, but instead, it is assigned a Wait unit-action.
- *Waiting frequently:* The unit is assigned Wait unit-actions more often than the other unit-actions, in the absence of immediate dangers/opportunities, making it less effective in pursuing opportunities and almost passive.

The presence of one Wait unit-action in a player-action (thus, IPA) is enough to introduce a risk of encountering one of the disadvantageous situations. The more Wait unit-actions in an IPA, the higher this risk gets. Thus, we believe that pruning the majority of IPAs from the search space, while preserving a fraction as a safety measure, to account for trapped units and tactical waiting, can be beneficial to MCTS.

C. Pruning Techniques

The radical pruning approach would be to remove all IPAs from the search space, basically removing the Wait unit-action from the set of unit-actions of all unit types. Thus, diminishing m by 1 and obtaining a branching factor $b' = (m-1)^n$, which represents a considerable decline from b. As an example, if we have $m = 5$ unit-actions on average in a given game state with $n = 6$ units, then $b \approx 1.56 \times 10^4$ and $b' \approx 4 \times 10^3$. The total number of IPAs removed would be: $v = b - b' = 1.16 \times 10^4$. The reduction is significant, but we intend to keep a portion of IPAs to deal with trapped units and tactical waiting.

Detecting trapped units is a simple task. But dealing with tactical waiting can be elusive since there is no simple way to differentiate between waiting as a tactical choice, and waiting as a bad decision until witnessing the consequences. Random playouts do not offer a reliable answer in that regard. We propose four pruning approaches that capture IPAs and decide whether to allow or prune them according to a given parameter. These approaches preserve all IPAs involving trapped units and allow a predetermined number/rate of random IPAs in hopes of preserving tactical waiting situations. The remaining IPAs are all considered disadvantageous and are systematically hard-pruned. We do not re-insert pruned IPAs because we consider the non-pruned player-actions more urgent to explore. The pruning approaches are described as follows:

- **Random Inactivity Pruning - Fixed (RIP-F(k)):** Allow a fixed number k of IPAs.
- **Random Inactivity Pruning - Relative (RIP-R(p)):** Allow a percentage of IPAs p relative to the total number of removable IPAs.
- **Dynamic RIP-F (DRIP-F(k_1, k_2)):** Allow k_1 IPAs when the agent's units outnumber the opponent's units and k_2 IPAs otherwise.
- **Dynamic RIP-R (DRIP-R(p_1, p_2)):** Allow p_1 percent of IPAs when the agent's units outnumber the opponent's units and p_2 percent of IPAs otherwise.

The intuition behind dynamic approaches is to equalize the chances of performing tactical waiting when the agent does not hold a numerical advantage. This is done by allowing more IPAs when the agent is outnumbered ($k_2 > k_1$ or $p_2 > p_1$).

These approaches can be easily implemented as part of any search algorithm, as shown in Algorithm 1. Each time a player-action a gets sampled, PRUNE(a) is called to decide whether to keep or replace a, in case it is an IPA. If a is a Non-IPA or an IPA involving a trapped unit, it will be returned as-is (lines 3 and 9). ISIPA(a) returns $true$ if a has at least one Wait unit-action, and TRAPPEDUNIT(a) returns $true$ if a Wait unit-action in a belongs to a trapped unit. The algorithm keeps track of previously-pruned IPAs in the *prunedIPAs* list to prevent re-insertions. The conditional expression in line 4 defines the pruning condition, which is satisfied either if a was previously pruned, or if CHECKPARAM() returns $true$.

The pruning approaches differ in the implementation of CHECKPARAM(). For instance, in RIP-F(k), CHECKPARAM() returns $true$ only if the number of IPAs allowed is higher than k. The loop in lines 5-8 will keep re-sampling for new player-actions until a is replaced with a non-IPA. Finally, a new non-IPA or an IPA allowed by CHECKPARAM() is returned.

V. EXPERIMENTATION AND RESULTS

To study the effect of pruning IPAs on MCTS, we implemented the four aforementioned pruning techniques in UCT and NaïveMCTS and conducted various experiments in μRTS. Indeed, UCT's performance suffers greatly in RTS scenarios due to UCB1's limitations in combinatorial search spaces [19],

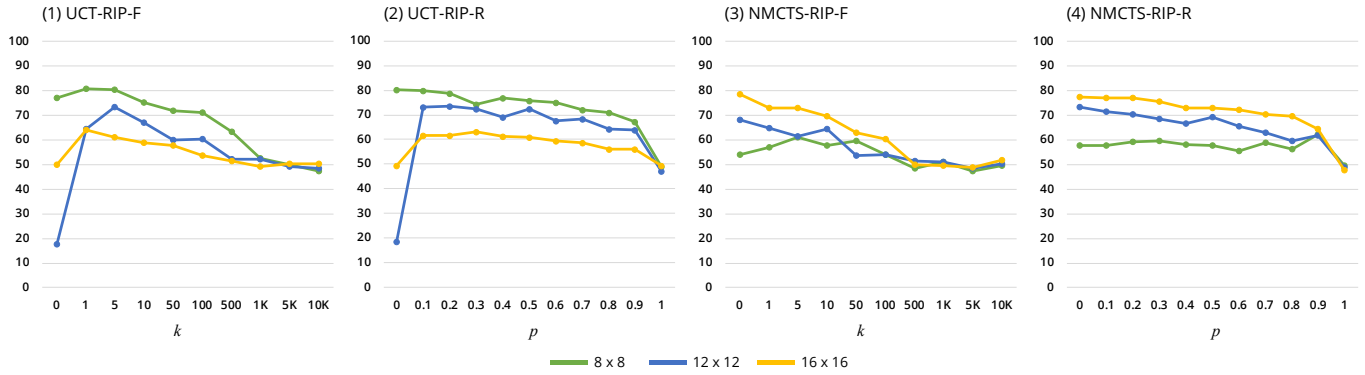

Fig. 2. Results of the pruning analysis experiments. Each data point represents 500 matches between a basic MCTS agent and the same agent enhanced with IPA pruning. The vertical axes represent the score obtained by the latter agent. The score is calculated as such: $score = ((Wins + (Draws/2))/500) \times 100$. A match is considered a draw if no winner has been decided after 3000, 3500, and 4000 cycles in each map of size 8×8, 12×12, and 16×16, respectively.

Algorithm 1 The general IPA pruning algorithm.

```
1: function PRUNE(a)                    ▷ a : A sampled player-action
2:     if ISIPA(a) then
3:         if TRAPPEDUNIT(a) then return a
4:         if a ∈ prunedIPAs or CHECKPARAM() then
5:             repeat
6:                 prunedIPAs.addIfNotExist(a)
7:                 a ← SAMPLEACTION(gameState)
8:             until ISIPA(a) == false
9:     return a
```

nevertheless we wanted to test if pruning IPAs would alleviate the dimensionality burden and results in performance improvement. Integrating IPA pruning in UCT and NaïveMCTS generated new agents that we refer to by suffixing the acronym of the technique to that of the original search approach. For instance, the agent using RIP-R with UCT or NaïveMCTS is noted as UCT-RIP-R(p) or NMCTS-RIP-R(p).

We first analyzed the performance of RIP-F and RIP-R relative to the number of IPAs allowed, the map's size, and the MCTS algorithm in use. We then took the top-performing pruning approaches for each MCTS algorithm and map size and performed a round-robin tournament with other μRTS agents. Afterward, we examined the impact on the branching factor and performed a scalability test in larger maps. The experiments were carried out on two PCs with Intel Core i5 and i7 CPUs, clocked at 3.1Ghz and 3.4Ghz, respectively, using the latest version of μRTS as of 30th March 2020.

A. Pruning Analysis

To analyze the influence of IPA pruning on the performance of MCTS, we ran a series of experiments involving each MCTS agent and non-dynamic IPA pruning approach. We defined two distinct sets, F and R, composed of a selection of values that can be taken by the parameters of RIP-F(k) and RIP-R(p), respectively. Next, we ran 500 matches (switching sides after 250 matches) between the MCTS agent enhanced with an IPA pruning approach, and the non-pruning version of the same MCTS agent for each respective value in F or R. The process was repeated for each *basesWorkers* map of size 8×8, 12×12 and 16×16. We define F and R as follows:

- $F = \{0, 1, 5, 10, 50, 100, 500, 1000, 5000, 10000\}$
- $R = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$

The total number of matches played for a single MCTS agent amounts to $(500 \times |F| \times 3) + (500 \times |R| \times 3)$, yielding 63000 matches for both UCT and NaïveMCTS. In all experiments, we kept the default UCT and NaïveMCTS parameters as defined in the μRTS codebase, for all variants. Agents were given $100ms$ per frame as a computation budget. The experiment results are expressed in Figure 2's plots.

Overall, we can see that IPA pruning is responsible for a performance gain of variable rates, relative to the number of IPAs allowed and the branching factor represented by the map's size. In UCT-RIP-F, allowing a small number of IPAs ($0 < k \leq 5$) significantly increases UCT's performance. However, the more IPAs are allowed, the more performance decreases until pruning losses its effect ($k \geq 1000$). The same trend is witnessed in UCT-RIP-R. Allowing a small percent of IPAs ($0.1 \leq p \leq 0.3$) increases UCT's performance, but the more we raise p the more performance drops. We note that UCT-RIP-R($p = 1$) and UCT-RIP-F($k \geq 1000$) are equivalent to non-pruning UCT.

The highest performance gain was recorded in the smallest 8×8 map with the lowest branching factor, and the lowest gain was recorded in the largest 16×16 map. This is expected from UCT since its sampling strategy (UCB1) is ineffective in combinatorial search spaces. Thus, pruning IPAs is not enough to scale UCT's performance. Pruning all IPAs ($k = 0$ or $p = 0$) hurts UCT's performance in larger maps due to the large number of IPAs encountered. This causes frequent player-action re-samplings that rapidly consume the computation budget, leaving very little time to exploitation. Although this effect is offset in the 16×16 map by the large number of draws, it is quite clear in the 12×12 map.

From the perspective of NaïveMCTS, IPA pruning exhibits the same effect as in UCT, with two key differences. First, in

the smallest 8×8 map, NaïveMCTS performs optimally when more IPAs are allowed (in comparison with UCT), that is, when $5 \leq k \leq 50$ in NMCTS-RIP-F and $p = 0.9$ in NMCTS-RIP-R. This is probably because naïve sampling already handles small scenarios well and can gain an advantage if a portion of IPAs is kept to explore tactical waiting situations. However in larger maps, pruning all IPAs ($k = 0$, or $p = 0$) yields the highest performance gain. Here, due to the bigger branching factor, pruning IPAs significantly contributes to the better utilization of the computation budget.

The second difference with respect to UCT is the scalability of performance, relative to the increase in the branching factor. As opposed to UCT, NaïveMCTS enhanced with IPA pruning delivers its best performance in the largest 16×16 map, followed by the 12×12 map. Having fewer IPAs in the search space seems to allow naïve sampling to sample more interesting player-actions, instead of wasting time on IPAs. Moreover, pruning IPAs increases the movement frequency of units, resulting in an enhanced ability to explore large maps. We will see further how this translates to larger maps.

B. Best Pruning Approaches

Concerning dynamic pruning approaches (DRIP-F and DRIP-R), we have conducted a similar experiment using UCT, by fixing k_1 (or p_1) to the optimal k (or p) value found for each map size and performing 500 matches for each k_2 (or p_2) value from F (or R). We omitted the results because of space constraints. For NaïveMCTS, dynamic pruning did not bring any improvement over non-dynamic approaches, based on preliminary tests. Thus, performing extensive experiments was not necessary. The best performing IPA pruning approaches for each map-size and MCTS algorithm are shown in Table II.

TABLE II
BEST PERFORMING PRUNING APPROACHES

	8×8	Score	12×12	Score	16×16	Score
UCT	DRIP-F(1,0)	82.2	DRIP-R(0.2,0.4)	76.8	RIP-F(1)	63.9
NaïveMCTS	RIP-R(0.9)	62.3	RIP-R(0)	73.3	RIP-F(0)	78.6

For UCT, dynamic approaches work best in the small and medium-sized maps due to the frequent encounters between opposing units. In the largest map, exploration becomes more urgent, rendering the dynamic approaches ineffective.

C. Performance Analysis

To further assess the performance impact of IPA pruning on MCTS agents, we have run a round-robin tournament between 8 μRTS agents, including two IPA pruning MCTS agents, under μRTS competition conditions. The tournament consists of 100 iterations, where in each iteration, every agent plays a match against the other agents resulting in $8 \times 7 \times 100 = 5600$ matches in each of the maps used previously. The participating μRTS agents include four baseline agents and one top performing agent from 2019's μRTS competition, MixedBot:

- *NaïveMCTS:* The original unmodified NaïveMCTS.

TABLE III
GLOBAL TOURNAMENT RESULTS (ROW AGENT VS. COLUMN AGENT)

	NMCTS-RIP	UCT-RIP	NaïveMCTS	UCT	RandomBiased	POWorkerRush	POLightRush	MixedBot	Average
NMCTS-RIP	-	**80.6**	**72.1**	**96**	**100**	42.1	47.3	26	**66.3**
UCT-RIP	20.8	-	30.6	**78**	**98.1**	26.8	35.5	14.1	43.4
NaïveMCTS	31	**69.3**	-	**94**	**100**	34.5	36.3	9.3	**53.5**
UCT	2.3	21.3	7.5	-	**84.8**	13	34.6	1	23.5
RandomBiased	0	1	0	13	-	0	6.6	0	2.9
POWorkerRush	**60.6**	**72.3**	**69.8**	**89.5**	**100**	-	**100**	**74.5**	**80.9**
POLightRush	**57.8**	**65.3**	**66.3**	**66.6**	**98.3**	0	-	30.8	**55**
MixedBot	**74.8**	**86.8**	**89.6**	**99**	**100**	30.5	**82**	-	**80.4**

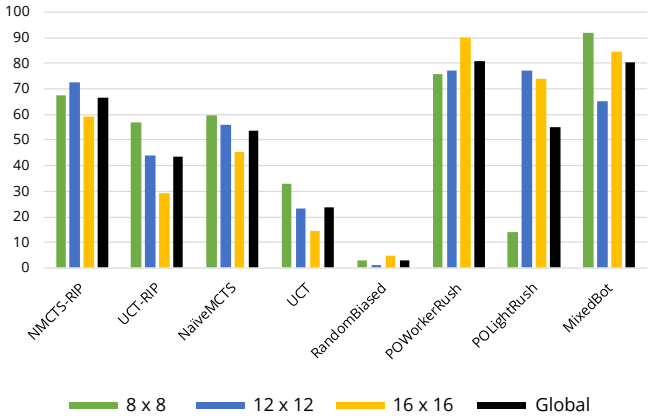

Fig. 3. The tournament results in each of the three map sizes. The vertical axis represents the score obtained against all agents.

- *RandomBiased:* Selects actions randomly, with a bias towards attacking and harvesting.
- *POWorkerRush:* Continuously produces workers and sends them to attack the opponent.
- *POLightRush:* Same as the above, but using Light units.
- *MixedBot:* Relies on two separate agents; Tiamat [14] for strategic decisions and Capivara [16] for tactical decisions. Both based on search space abstraction through scripts and tuned for different map sizes.

In addition to the following IPA pruning agents:

- *NMCTS-RIP:* NaïveMCTS integrating the best performing IPA pruning approach for each map size, as defined in Table II.
- *UCT-RIP:* Same as the above, but based on UCT.

We also included unmodified UCT for the sake of comparison. The global tournament results are reported in Table III, and the results by map-size are shown in Figure 3. The score is calculated similarly to the previous experiment.

The results demonstrate how IPA pruning positively affects the playing strength of UCT and NaïveMCTS. Looking at the average scores, NMCTS-RIP achieved a 12.8 points increase with respect to NaïveMCTS, and UCT-RIP achieved a near 20 points increase, relative to UCT. IPA pruning in UCT (UCT-RIP) managed to shrink the performance gap between UCT and NaïveMCTS from 30 points to 10.1 points, noticeable in

TABLE IV
BRANCHING FACTOR, SAMPLED ACTIONS AND IPA PRUNING STATISTICS

Agent	Map	Avg. Branching Factor, b (Incl IPAs)	Avg. Unit Count, n	Avg. Unit-Actions, m (Incl Wait)	Avg. Branching Factor, b' (w/o IPAs)	Avg. Sampled Actions	IPAs % in Sampled Actions	Pruned IPAs % (WRT Sampled Actions)
NMCTS-RIP	8×8	1782	5.19	3.15	54	70.81	33.82%	6.04%
	12×12	2.48×10^7	14.39	2.67	1561	75.27	71.81%	14.32%
	16×16	9.519×10^{11}	20.92	3.11	6.12×10^6	71.25	76.72%	24.56%
NaïveMCTS	8×8	784	5.32	2.71	17	71.76	80.03%	0%
	12×12	6.51×10^6	14.39	2.54	494	80.11	99.24%	0%
	16×16	1.842×10^{11}	20.06	3.05	1.87×10^6	76.85	99.88%	0%

Figure 3 where UCT-RIP's performance closely matches that of NaïveMCTS in the 8×8 map. Moreover, NMCTS-RIP was able to score a higher average than POLightRush, one of the strongest scripts usually outranking NaïveMCTS. Against each agent, both NMCTS-RIP and UCT-RIP obtained significantly higher scores than those of NaïveMCTS and UCT respectively.

As expected, scripts and script-based approaches exhibit a superior performance versus low-level MCTS search approaches, due to the presence of expert knowledge in the form of hard-coded scripts. Expert knowledge helps in avoiding detrimental player-actions by focusing the search on a limited set of player-actions judged more rewarding. However, this comes at the cost of lower decision granularity and higher exploitation risk. By pruning detrimental player-actions, we hope to focus the search on a wider range of interesting player-actions and keep a higher degree of decision granularity. The fact that NMCTS-RIP could achieve a higher average score than MixedBot in the 12×12 map signifies that our approach could be promising.

Under the μRTS competition settings, both holistic MCTS agents and those that rely on MCTS only for low-level tactical planning, e.g., [3] and [16], would benefit from IPA pruning and see a tactical performance gain that positively impacts their overall performance. Furthermore,

D. Branching Factor & Scalability

To better grasp how IPA pruning affects MCTS performance, we took 100 mid-game states from matches between NMCTS-RIP and NaïveMCTS and ran a $100ms$ search, starting from those states for both agents. The search was limited to one ply ($maxDepth = 1$), and the mid-game was defined at 400, 600 and 1000 game cycles for each map of size 8×8, 12×12 and 16×16, respectively. The statistics collected during these searches are reported in Table IV.

We can see that the branching factor b in mid-game states is higher in NMCTS-RIP, as a result of the similarly higher unit-actions average m. This, in turn, is the consequence of the units being spread out on the map due to lesser IPAs (more movements), leading to more space between units, and more possible actions for each. NMCTS-RIP samples player-actions from a subset of the decision space having a lower bound branching factor b', expanded by the number of IPAs involving trapped units, and a random set of IPAs allowed by the pruning approach.

TABLE V
NMCTS-RIP-F(0) RESULTS IN LARGER MAPS

	Wins	Losses	Draws	Score
24×24	52	6	42	73
32×32	43	1	56	71

The rate of IPAs in both branching factor and sampled actions grows proportionally to the branching factor, as expected. The rate of IPAs in sampled actions is significantly high in NaïveMCTS, reaching near 100% in larger maps. Whereas in NMCTS-RIP, this rate drops under 34% and would not go beyond 77% in the tested maps. Moreover, the rate of pruned IPAs increases proportionally with the branching factor. Therefore, we may conclude that in larger maps NaïveMCTS gets fully overwhelmed by IPAs, while NMCTS-RIP prunes more IPAs and focuses on a larger number of possibly better player-actions. This further highlights the detrimental effect of the overabundance of IPAs.

We have run 100 matches (switching sides after 50 matches) between NMCTS-RIP-F(0) and NaïveMCTS in larger 24×24 and 32×32 maps, to test the performance scalability of IPA pruning in larger scenarios. The results in Table V suggest a stable score trend and an increasing win/loss ratio proportional to the map's size. Further experiments in these scenarios are planned in the context of our next works.

VI. CONCLUSIONS AND FUTURE WORK

Throughout this paper, we have studied the possibility of employing move pruning as a way to enhance MCTS performance in the context of RTS games. We have identified a class of player-actions that can negatively impact the performance of low-level MCTS approaches. We labeled those actions as Inactive Player-Actions (IPAs) due to their tendency to keep at least one unit in an inactive state. Several pruning approaches were conceived to prune IPAs, taking into account the existence of possibly useful IPAs. We then carried out a range of experiments to test the validity of our approaches and discussed the obtained results. According to the results, pruning IPAs is associated with a meaningful performance gain, due to the reduced branching factor and the increased focus on more interesting player-actions. IPA pruning in NaïveMCTS has demonstrated an impressive performance across increasingly larger maps, especially when all excessive IPAs get pruned. Therefore, we conclude that NaïveMCTS can

safely ignore all superfluous IPAs in such situations, which will grant a risk-free performance boost in an RTS game.

Move pruning in this context can be seen as "inverse" action abstraction, since we are trying to find the set of player-actions to avoid, whereas, action abstraction methods seek to find the set of player-actions to focus on exclusively. Pruning low-quality player-actions could result in a more flexible and granular decision space, rather than the coarser space induced by action abstractions (scripts).

μRTS agents integrating a low-level search technique with IPA pruning could gain an improved tactical reasoning ability. This improvement would positively influence the agent's performance in the μRTS competition.

Researching more prunable player-action types falls into the scope of our next work, along with further analysis of IPA pruning in larger scenarios, and the analysis of the impact of IPA pruning in multi-level search approaches such as STT [3] and A3N [16]. We believe that further research into the low-level structure of the RTS decision space could lead to a deeper understanding of the general features of higher-quality decisions.

REFERENCES

[1] P.-A. Andersen, M. Goodwin, and O.-C. Granmo, "Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug. 2018, pp. 149–156.

[2] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo Tree Search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, Dec. 2010.

[3] N. A. Barriga, M. Stanescu, and M. Buro, "Combining Strategic Learning with Tactical Search in Real-Time Strategy Games," in *AIIDE'17*, Snowbird Ski Resort, Utah, Sep. 2017, pp. 9–15.

[4] ——, "Puppet Search: Enhancing Scripted Behavior by Look-Ahead Search with Applications to Real-Time Strategy Games," in *AIIDE'15*, Santa Cruz, California, Sep. 2015, pp. 9–15.

[5] 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. 1–49, Mar. 2012.

[6] D. Churchill and M. Buro, "Portfolio greedy search and simulation for large-scale combat in starcraft," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. Niagara Falls, ON, Canada: IEEE, Aug. 2013.

[7] J. Duguépéroux, A. Mazyad, F. Teytaud, and J. Dehos, "Pruning Playouts in Monte-Carlo Tree Search for the Game of Havannah," in *Computers and Games*. Cham: Springer International Publishing, 2016, vol. 10068, pp. 47–57.

[8] S. He, Y. Wang, F. Xie, J. Meng, H. Chen, S. Luo, Z. Liu, and Q. Zhu, "Game Player Strategy Pattern Recognition and How UCT Algorithms Apply Pre-knowledge of Player's Strategy to Improve Opponent AI," in *2008 International Conference on Computational Intelligence for Modelling Control & Automation*. Vienna, Austria: IEEE, 2008, pp. 1177–1181.

[9] E. A. Heinz, "Adaptive Null-Move Pruning," *ICGA Journal*, vol. 22, no. 3, pp. 123–132, Jan. 1999.

[10] K. Hoki and M. Muramatsu, "Efficiency of three forward-pruning techniques in shogi: Futility pruning, null-move pruning, and Late Move Reduction (LMR)," *Entertainment Computing*, vol. 3, no. 3, pp. 51–57, Aug. 2012.

[11] J. Huang, Z. Liu, B. Lu, and F. Xiao, "Pruning in UCT Algorithm," in *2010 International Conference on Technologies and Applications of Artificial Intelligence*. Hsinchu City, TBD, Taiwan: IEEE, Nov. 2010, pp. 177–181.

[12] 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*. Dortmund, Germany: IEEE, Aug. 2014.

[13] L. Kocsis, C. Szepesvari, and J. Willemson, "Improved Monte-Carlo Search," *Univ. Tartu, Estonia, Tech. Rep 1*, 2006.

[14] J. R. H. Mariño, R. O. Moraes, C. Toledo, and L. H. S. Lelis, "Evolving Action Abstractions for Real-Time Planning in Extensive-Form Games," in *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2018.

[15] R. O. Moraes and L. H. S. Lelis, "Asymmetric Action Abstractions for Multi-Unit Control in Adversarial Real-Time Games," in *The Thirty-Second AAAI Conference on Artificial Intelligence AAAI-18*, 2018, pp. 876–883.

[16] R. O. Moraes, J. R. H. Mariño, L. H. S. Lelis, and M. A. Nascimento, "Action Abstractions for Combinatorial Multi-Armed Bandit Tree Search," in *AAAI Publications, Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

[17] X. Neufeld, S. Mostaghim, and D. Perez-Liebana, "A Hybrid Planning and Execution Approach Through HTN and MCTS," in *The 3rd Workshop on Integrated Planning, Acting, and Execution - ICAPS'19*, Jul. 2019, pp. 37–45.

[18] S. Ontañón, "Informed Monte Carlo Tree Search for Real-Time Strategy games," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. Santorini, Greece: IEEE, Sep. 2016.

[19] ——, "Combinatorial Multi-armed Bandits for Real-Time Strategy Games," *Journal of Artificial Intelligence Research*, vol. 58, pp. 665–702, Mar. 2017.

[20] S. Ontañón, N. A. Barriga, C. R. Silva, R. O. Moraes, and L. H. S. Lelis, "The First microRTS Artificial Intelligence Competition," *AI Magazine*, vol. 39, no. 1, pp. 75–83, Mar. 2018.

[21] S. Ontañón and M. Buro, "Adversarial Hierarchical-Task Network Planning for Complex Real-Time Games," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 2015, pp. 1652–1658.

[22] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 4, pp. 293–311, Dec. 2013.

[23] A. Ouessai, M. Salem, and A. M. Mora, "Online Adversarial Planning in μRTS : A Survey," in *2019 International Conference on Theoretical and Applicative Aspects of Computer Science (ICTAACS)*, Dec. 2019.

[24] N. Sephton, P. I. Cowling, E. Powley, and N. H. Slaven, "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*. Dortmund, Germany: IEEE, Aug. 2014.

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[26] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018.

[27] Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick, "ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games," in *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.*, 2017.

[28] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "StarCraft II: A New Challenge for Reinforcement Learning," *arXiv:1708.04782 [cs]*, Aug. 2017.

[29] Z. Yang and S. Ontañón, "Extracting Policies from Replays to Improve MCTS in Real Time Strategy Games," in *The 2nd Workshop on Knowledge Extraction from Games Co-Located with 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, Honolulu, Hawaii, 2019.

[30] ——, "Guiding Monte Carlo Tree Search by Scripts in Real-Time Strategy Games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, Oct. 2019, pp. 100–106.

[31] ——, "Integrating Search and Scripts for Real-Time Strategy Games: An Empirical Survey," in *AAAI-20 Workshop on Reinforcement Learning in Games*, New York, 2020.