Action Abstractions for Combinatorial Multi-Armed Bandit Tree Search

Rubens O. Moraes  
Departamento de Informática  
Universidade Federal de Viçosa  
Viçosa, Minas Gerais, Brazil

Levi H. S. Lelis  
Departamento de Informática  
Universidade Federal de Viçosa  
Viçosa, Minas Gerais, Brazil

Julian R. H. Mariño  
Instituto de Ciências Matemáticas e Computação  
Universidade de São Paulo  
São Carlos, São Paulo, Brazil

Mario A. Nascimento  
Department of Computing Science  
University of Alberta  
Edmonton, Alberta, Canada

Abstract

Search algorithms based on combinatorial multi-armed bandits (CMABs) are promising for dealing with state-space sequential decision problems. However, current CMAB-based algorithms do not scale to problem domains with very large actions spaces, such as real-time strategy games played in large maps. In this paper we introduce CMAB-based search algorithms that use action abstraction schemes to reduce the action space considered during search. One of the approaches we introduce use regular action abstractions (A1N), while the other two use asymmetric action abstractions (A2N and A3N). Empirical results on µRTS show that A1N, A2N, and A3N are able to outperform an existing CMAB-based algorithm in matches played in large maps, and A3N is able to outperform all state-of-the-art search algorithms tested.

Introduction

Real-time strategy (RTS) games are challenging for planning systems because they present large action spaces and the time allowed for planning is very limited, typically in the order of milliseconds. Ontañón (2013; 2017a) proposed the use of combinatorial multi-armed bandits (CMAB) (Gai, Krishnamachari, and Jain 2010; Chen, Wang, and Yuan 2013; Combes et al. 2015) as a method for allowing Monte Carlo tree search (MCTS) algorithms (Browne et al. 2012) to focus their search on a set of promising actions; Ontañón’s search algorithm was named NaïveMCTS. Although the CMAB approach helps the MCTS search to focus on a subset of actions, the number of actions to be evaluated during search grows quickly with the size of the map. As a result, NaïveMCTS does not scale to matches played in large maps.

In this paper we introduce three CMAB-based MCTS algorithms that leverage on the concept of action abstractions to further reduce the number of promising actions one has to consider during search. Action abstractions were originally introduced in the context of RTS combats by Churchill and Buro (2013), who used hard-coded scripts to induce action abstractions. A script is a function that, given a game state and a unit, returns which action the unit should perform in that state. Given a set of scripts, instead of searching over all possible actions, Churchill and Buro’s search algorithm considers only the actions returned by the scripts. Moraes and Lelis (2018) introduced, also in the context of RTS combats, the concept of asymmetric actions abstractions. In contrast to Churchill and Buro’s action abstractions, asymmetric abstractions restrict the number of actions available to a subset of units, while the search algorithm considers all available actions for the remaining units. Search algorithms using asymmetric abstractions can deliver a “finer control” to units deemed as more important at a given moment of the game by considering a greater number of actions for those units.

One of the CMAB-based algorithms we introduce uses regular actions abstractions. We call this algorithm A1N, where ‘A’ stands for abstraction and “N” for naïve, since A1N is based on NaïveMCTS. We also introduce other two algorithms, which we call A2N and A3N, that use asymmetric action abstractions. A2N and A3N differ in the way that their abstractions are defined. A2N uses two sets of scripts to define its abstractions, one set is composed of economy-based scripts and is used to reduce the set of actions of economy-based units, and the other set is composed of combat-based scripts and is used to reduce the set of actions of military units. A3N uses Moraes and Lelis (2018)’s approach, i.e., it uses a single set of scripts to reduce the set of actions available during search for a subset of units; all actions of the remaining units are considered during search.

We evaluate our CMAB-based search algorithms that use action abstractions with an extensive set of experiments on µRTS. µRTS is a great testbed for our research because it offers an efficient forward model of the game, which is required by search-based approaches. Moreover, the game is much simpler than commercial video games, which allows us to quickly test different approaches. Finally, µRTS codebase\(^1\) contains most of the current state-of-the-art search-based methods, thus facilitating our empirical evaluation.

---

\(^1\)https://github.com/santiontanon/microrts
The results of our experiments show that A1N, A2N, and A3N are all able to outperform NaïveMCTS. Our results also show that A3N is able to outperform all state-of-the-art algorithms tested, including those that competed in the latest μRTS competition (Ontañón et al. 2018).

Related Work
In an RTS combat each player controls a set of units that battles until one of the players is able to eliminate all the other player’s units. Previous works proved action abstraction schemes to be effective in such scenarios (Churchill and Buro 2013; Justesen et al. 2014; Wang et al. 2016; Lelis 2017; Moraes and Lelis 2018). In this work we show that such abstractions can also be effective in the context of complete RTS matches. Also, previous works used simplified versions of the game in which features such as unit collision were disregarded. This distinction is important because scripts used to generate action abstractions are computationally more expensive in μRTS due to the need of running path-finding algorithms such as A* (Hart, Nilsson, and Raphael 1968). The time required to run the scripts subtracts from the very limited amount of time available for planning. We show that action abstractions can be valuable in the context of complete RTS matches despite their computational cost, as our experiments show that algorithms that use action abstractions are able to outperform un-abstracted ones.

Other methods for RTS games include non-CMAB MCTS approaches (Chung, Buro, and Schaeffer 2005; Sailer, Buro, and Lancot 2007; Balla and Fern 2009) and Alpha-Beta pruning (Churchill, Saffidine, and Buro 2012). Similarly to NaïveMCTS, these methods do not scale to large maps due to the large number of actions available during search.

Scripts have also been used to guide the search by means other than action abstractions. Puppet Search (PS) (Barriga, Stanescu, and Buro 2017b) defines a search space over the parameter values of scripts. Strategy Tactics (STT) (Barriga, Stanescu, and Buro 2017a) combines PS’s search in the script-parameter space with a NaïveMCTS search in the original state space for the combat units. Strategy Creation via Voting (SCV) generates scripts via voting (Silva et al. 2018), which are then used to play the game, showing the potential behind the scripts and how they can be combined to generated new and potentially powerful variations. In contrast with PS, STT, and SCV that generate novel scripts during the game, the algorithms we introduce in this paper use a set of scripts to generate action abstractions.

Although we tested our action abstraction schemes by introducing variants of NaïveMCTS, in principle, our schemes could be used to reduce the action space of other CMAB-based algorithms such as 2-Phase-CMAB approaches (Shleyman, Komenda, and Domshlak 2014) and of search algorithms that use enhancements such as Hierarchical Expansion (Roelofs 2015).

Definitions and Notations
Following the notation of Ontañón (2017a), an RTS match can be described as a finite zero-sum two-player game with simultaneous-move and be denoted as $\langle P, S, A, \tau, L, W, s_{init} \rangle$, where:

- $P = \{\text{max}, \text{min}\}$ is the set of players.
- $S$ is the match’s set of states. A state $s \in S$ describes the match at a given moment. A state contains the position on the map of each unit controlled by the players as well as properties for these units, i.e., the current amount of hit points and if the units are currently performing an action.
- $A = A_{\text{max}} \times A_{\text{min}}$ is a set of joint player-actions, where $A_{\text{max}}$ and $A_{\text{min}}$ denote the player-actions of max and min, respectively. A player-action is represented by a vector of unit-actions, with one unit-action for each unit controlled by the player that is ready to act; a unit is not ready if it is already performing a unit-action.
- $\tau : S \times A_{\text{max}} \times A_{\text{min}} \rightarrow S$ is a deterministic transition function, that given a state $s$ and a set of joint player-actions, returns a resulting state in $S$.
- $L : S \times A \times P \rightarrow \{\text{true}, \text{false}\}$ is a function that given a state $s$, a player-action $a$ and a player $p$, determines whether it is legal for $p$ to execute $a$ in $s$.
- $W : S \rightarrow P \cup \{\text{draw}, \text{ongoing}\}$ is a function that determines whether one of the players won the match, if the match finished in a draw, or if the match is still ongoing.
- $s_{init}$ is the initial state of the match.

A decision point of player $p$ is a state $s$ in which $p$ has at least one ready unit. A pure strategy is a function for player $p$ mapping a state $s$ to a player-action $a$. Although in general one might have to play a mixed strategy to optimize the player’s payoffs in simultaneous move games (Gintis 2000), we follow the current state-of-the-art RTS methods (Barriga, Stanescu, and Buro 2017b; Lelis 2017; Churchill and Buro 2013; Churchill, Saffidine, and Buro 2012) and consider only pure strategies in this paper. In fact, we use state-space search algorithms to derive pure strategies in real time.

CMAbs for RTS Games
Ontañón (2017a) modeled the search problem of deriving strategies for a player in an RTS match as a combinatorial multi-armed bandits (CMAB) problem. A CMAB problem can be defined by a tuple $\langle X, \mu \rangle$. Here,

- $X = \{X_1, \cdots, X_n\}$, where each $X_i$ is a variable that can assume $K_i$ different values $X_i = \{v_{i1}, \cdots, v_{iK_i}\}$, with $X = \{(v_1, \cdots, v_n) \in X_1 \times \cdots \times X_n\}$ being the possible combinations of value assignments for the variables in $X$; a value assignment $V \in X$ is called a macro-arm.
- $\mu : X \rightarrow \mathbb{R}$ is a reward function, that receives a macro-arm and returns a reward value for that macro-arm.

The goal in a CMAB problem is to find a macro-arm that maximizes the expected reward. This can be achieved by balancing exploration and exploitation until converging to an optimal macro-arm. In the context of RTS games, each decision point $s$ can be cast as a CMAB problem in which $X$ contains one variable for each ready unit of a player in $s$, thus a macro-arm $V \in X$ represents a player-action and each value $v \in V$ represents a unit-action. The set
If explore is chosen, then a macro-arm (player-action) that maximizes the player’s reward. The reward is defined by a game-specific function.

Naïve Sampling
Since the number of macro-arms in $X$ is often too large for decision points in RTS matches, Ontaño (2017a) derived a sampling procedure called Naïve Sampling (NS) to help deciding which macro-arms should be evaluated during search.

NS divides a CMAB problem with $n$ variables in $n + 1$ multi-armed bandit (MAB) problems.

- $n$ local MABs, one for each variable $X_i \in X$. That is, for variable $X_i$ representing a unit, the arms of the MAB are the $K_i$ values (unit-actions) in $X_i$.
- 1 global MAB, denoted MAB$_g$, that treats each macro-arm $V$ evaluated thus far in search as an arm in MAB$_g$. Naturally, MAB$_g$ has no arms in the beginning of search.

At each iteration, NS uses a policy $\pi_0$ to determine whether it adds an arm to MAB$_g$ through the local MABs (explore) or evaluates an existing arm in MAB$_g$ (exploit).

1. If explore is chosen, then a macro-arm $V$ is added to MAB$_g$ by using a policy $\pi_t$ to independently choose a value for each variable in $X$. Here, NS assumes that the reward of a macro-arm $V$ can be approximated by the sum of the rewards of the individual values $v_i \in V$, denoted $\mu'(v_i)$. That is, $\mu(V) \approx \sum_{v_i \in V} \mu'(v_i)$.

2. If exploit is chosen, then a policy $\pi_g$ is used to select an existing macro-arm in MAB$_g$.

In this paper we use $\epsilon$-greedy for policies $\pi_0$, $\pi_t$, and $\pi_g$.

Ontaño (2017a) showed that NS can be used in the context of a MCTS algorithm to choose a subset of player-actions to be considered during search; the resulting algorithm was named NaïveMCTS. NaïveMCTS differs from other instantiations of MCTS algorithms in that it uses NS to decide which player-action to explore next. Instead of uniformly sampling which player-action to evaluate next as a vanilla MCTS algorithm would do, NaïveMCTS focuses through NS its search effort on player-actions composed of unit-actions that tend to yield good reward values. While NS allows NaïveMCTS to focus its search on a subset of promising player-actions, NaïveMCTS is still outperformed by search algorithms that rely on the guidance of scripts in matches played in large maps (Silva et al. 2018). Next, we describe how NaïveMCTS can leverage the domain knowledge encoded scripts through action abstractions.

Action-Abstracted NaïveMCTS (A1N)
An action abstraction for player $p$ is a function mapping the set of legal actions $A_p$ at a decision point $s$ to a subset $A'_p$ of $A_p$. Similarly to previous works (Churchill and Buro 2013; Wang et al. 2016; Lelis 2017; Moraes and Lelis 2018), we use a set of scripts $P$ to define an abstraction. A script $\sigma$ is a function that receives a decision point $s$ and a unit $u$ as input, and returns a legal unit-action $m$ for $u$ to perform at $s$. In an action abstraction, $M(s, u, P) = \{\sigma(s, u) | \sigma \in P\}$ is the set of unit-actions a unit can perform at $s$, and $A'_p = \{(m_1, \cdots, m_n) \in M(s, u_1, P) \times \cdots \times M(s, u_n, P)\}$.

We call A1N a version of NaïveMCTS that uses an action abstraction induced by a set of scripts $P$. The difference between NaïveMCTS and A1N is in the unit-actions sampled by NS while adding macro-arms to MAB$_g$. Instead of being able to sample from all legal unit-actions, A1N’s NS is allowed to sample only from unit-actions in $M(s, u, P)$ for each unit $u$. As a consequence, the macro-arms (player-actions) added to MAB$_g$ are restricted to those in $A'_p$.

Asymmetrically Action-Abstracted NaïveMCTS (A2N and A3N)
Regular action abstractions use the same set of scripts $P$ to reduce the unit-actions available during search for all units controlled by the player. Moraes and Lelis (2018) introduced an abstraction scheme they called asymmetric action abstraction to allow a larger set of unit-actions to be available to a subset of key units (e.g., units that are engaged in combat). Moraes and Lelis’s asymmetric abstractions are defined by selecting a subset of the player’s units, called unrestricted units, for which all legal unit-actions would be accounted for during search, while all the other units (restricted ones) would have access only to the actions returned by the set $P$.

In addition to Moraes and Lelis’s asymmetrical scheme, we also consider the asymmetry produced by two sets of scripts $P_e$ and $P_b$, where $P_e$ contains scripts for economy-related units (i.e., barracks, base, and workers) and $P_b$ contains scripts for combat units (i.e., military units)—see Ontaño et al. (2018) for information about $\mu$RTS’s units.

We call A2N the version of NaïveMCTS that searches in an asymmetrically-abstracted action space defined by two sets of scripts, one for economy units and one for combat units. We call A3N the version of NaïveMCTS that allows all legal unit-actions to the set of unrestricted units and only the actions returned by a set of scripts for all the other units.

A3N’s Unrestricted Units
By changing the number of unrestricted units, which we denote by $N$, one can make A3N more similar or less similar to NaïveMCTS and A1N. That is, if all units are unrestricted, then A3N becomes NaïveMCTS; if no units are unrestricted, then A3C becomes A1C. In addition to the number of unrestricted units, one needs to define which units compose the unrestricted set. We test empirically several unrestricted set sizes and 7 strategies for selecting the unrestricted units, which are described below. The first four strategies are introduced in this paper (CE, FE, HP+, HP), while the last three (AV+, AV-, R) were introduced by Moraes and Lelis (2018).

1. Closest to an enemy (CE). CE selects the $N$ units that are closest to an enemy unit at every decision point. The intuition behind CE is to allow a finer control for the units that are likely to be more threatened by enemy units.

2. Farthest from an enemy (FE). FE selects the $N$ units that are the farthest of an enemy unit at every decision point.
More attack value (AV+)  

Less life (HP-)  

Random (R)  

Less attack value (AV-)

Random (R)
scripts. NOKA V is a strategy that chooses an attack action that will not cause more damage than that required to eliminate the enemy unit from the match; Kiter allows the units to move away from the enemy, and Cluster groups the units together. All experiments were run on 2.1 GHz CPUs.

**First Experiment: A3N vs A1N**

Table 1 shows the winning rate of each A3N variant against A1N in 40 matches played in each of the 7 maps. The winning rate is computed by summing the total number of victories and half of the number of draws of a variant of A3N and then dividing this sum by the total number of matches played; the result of the division is then multiplied by 100. For example, if a variant had won all 40 matches in all 7 maps, then it would have a winning rate of 100.0. The rows show the strategies used for selecting the unrestricted set while the columns show the size of the set. We highlight the cells in which A3N had a winning rate larger than 50.0.

A3N tends to perform better with smaller unrestricted set sizes ($N \leq 5$). This is because the number of macro-arms that can be sampled by A3N grows quickly as one increases $N$, making it harder for the search procedure to encounter macro-actions that result in strong gameplay. As anticipated, the strategies HP+ and AV- were outperformed by their counterparts HP- and AV+. In contrast with the results of Moraes and Lelis (2018) for combat scenarios, the best performing strategy for playing full RTS matches is CE with $N = 1$, which had a winning rate of 76.9, and not AV+, which with its best $N$-value had a winning rate of 56.5. We hypothesize that this result is related to the characteristics of the game. Compared to commercial games such as StarCraft, all units in μRTS have low hit point values, thus not lasting long in the match. If preserving units longer in the match is a good strategy, then one should probably provide a fine control to the units closest to the enemy, rather than the units with largest AV+ value as the former are more likely to be eliminated than the latter.

**Second Experiment: State-of-the-Art Comparisons**

Table 2 shows the total number of matches won by each approach tested in all 18 maps; matches finishing in draws are not included in these results. The maximum achievable number of victories is $18 \times 9 \times 10 = 1,620$ (18 maps, 9 opponents, and 10 matches played against each opponent). Overall, A3N wins more matches than any approach tested, suggesting that if a large diversity of maps are considered, then A3N is the current state-of-the-art in μRTS matches. It is also interesting to observe how the NaïveMCTS approaches perform with respect to each other methods. NaïveMCTS (denoted as NAV) wins a total of 231 matches, being the method that wins the least amount of matches. A1N, which uses an action abstraction scheme, is already substantially better than NAV. The employment of an asymmetric abstraction through two sets of scripts further improves the results as A2N has more victories than A1N, but they are still behind some of the state-of-the-art approaches. The advantage of A2N over A1N is that the former allows a finer control over the combat units with the scripts NOKA V, Kiter, and Cluster. The best results are achieved when NaïveMCTS is combined with an asymmetric action abstraction that allows the unit closest to an enemy unit to consider all its legal moves in a given decision point. These results show that NaïveMCTS is able to leverage the knowledge encoded in scripts through abstraction schemes.

It is interesting to note how A3N’s search procedure relates to how humans play RTS games. Professional players often control one unit at a time when optimizing for battles, and they often switch which unit will receive a finer control. A3N acts similarly as it provides a coarse abstraction to all units but the unit closest to an enemy unit, for which all actions are accounted for during search. Moreover, the strategy of selecting the unit that is closest to an enemy unit also allows A3N to quickly switch, depending on the game scenario, which unit will receive a finer control.

Table 3 shows the winning rate of the row method against the column method in maps of different sizes. The numbers shown in the table are truncated to one decimal place. We highlight the background of a cell of A1N, A2N, and A3N if they have a winning rate greater or equal to 50.0 against an opponent. We also highlight the cell with the highest entry in the “Avg” column, which shows the average winning rate of the row method in a given map size. This table allows ones to understand the circumstances in which a method is stronger than another. For example, NAV is one of the strongest methods in maps of size $8 \times 8$ because the number of macro-arms available to be sampled by NAV is much smaller in matches played in smaller maps, which allows NAV to effectively find good player-actions within the game’s time constraint. However, as one increases the size of the maps, the performance of NAV drops quickly. For example, it drops from 77.2 in the smallest maps to 27.8 in maps of size $16 \times 16$.

In contrast with NAV, the action-abstracted approaches, specially A3N, present similar performance across different maps. The worst performances of A3N are in matches played on maps of size $8 \times 8$ and $32 \times 32$. The reason of A3N’s lower performance on these maps is related to their “actual size”. Although maps $32 \times 32$ are comparatively larger than maps of size $16$, $24$, one of the maps of size $32$ has the players starting the game close to each other, similarly to what happens in a map of size $8$, which favors the LR strategy. A3N wins almost all the other matches played on the other map of size $32$, where the players start far apart. The difference between NAV and A3N suggests a direction of future research, which is to develop adaptive methods for

<table>
<thead>
<tr>
<th></th>
<th>A3N</th>
<th>SCV</th>
<th>PS</th>
<th>STT</th>
<th>LR</th>
<th>A2N</th>
<th>WR</th>
<th>A1N</th>
<th>AHT</th>
<th>NAV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>1,125</td>
<td>1,044</td>
<td>931</td>
<td>860</td>
<td>729</td>
<td>686</td>
<td>680</td>
<td>661</td>
<td>311</td>
<td>198</td>
</tr>
</tbody>
</table>

Table 2: Total number of victories by each method in 18 maps (possible maximum of 1,620); draws are disregarded.
Table 3: Winning rate of the row player against the column player. The winning rate is computed by summing the total number of victories and half of the number of the row player's matches against a column player, and then dividing this sum by the total number of matches played; the result of the division is then multiplied by 100. If a method wins all matches, then it has a winning rate of 100. We highlight the cells in which A1N, A2N or A3N has a winning rate greater or equal to 50.0.

Table 2.

Conclusions and Future Work

In this paper we have introduced three CMAB-based search algorithms that use action abstraction schemes for RTS games, A1N, A2N, and A3N. Our algorithms combine the guidance provided by scripts through action abstractions, the sampling procedure given by a CMAB approach, and MCTS search. An extensive set of experiments on μRTS, a simple and yet complete RTS game, shows that A1N, A2N, and A3N are able to outperform NaïveMCTS, the CMAB-based algorithm that inspired us to do our research. Our experiments also show that A3N, which uses an asymmetric action abstraction scheme, is able to outperform all state-of-the-art approaches tested in our experiment, which includes the algorithms that competed in the latest µRTS competition.
tion (Onta˜n´on et al. 2018). Our algorithms and results suggest several directions of future research to further improve the current state of the art, which includes the development of stronger scripts for inducing better action abstractions as well as asymmetric action abstraction schemes that automatically adjust the size of the unrestricted set.

Acknowledgements
This research was partially supported by FAPEMIG, CNPq and CAPES, Brazil and NSERC, Canada, and done while R.O. Moraes and L.H.S. Lelis were visiting the University of Alberta. The authors thank the great suggestions provided by the anonymous reviewers.

References


Barriga, N. A.; Stanescu, M.; and Buro, M. 2017b. Game tree search based on non-deterministic action scripts in real-time strategy games. IEEE Transactions on Computational Intelligence and AI in Games.


