Evolving Action Abstractions for Real-Time Planning in Extensive-Form Games

Julian R. H. Mariño,[1] Rubens O. Moraes,[2] Claudio Toledo,[1] Levi H. S. Lelis[2]
[1]Departamento de Sistemas de Computação, ICMC, Universidade de São Paulo, Brazil
[2]Departamento de Informática, Universidade Federal de Viçosa, Brazil
julianmarino@usp.br, rubens.moraes@ufv.br, claudio@icmc.usp.br, levi.lelis@ufv.br

Abstract

A key challenge for planning systems in real-time multi-agent domains is to search in large action spaces to decide an agent's next action. Previous works showed that handcrafted action abstractions allow planning systems to focus their search on a subset of promising actions. In this paper we show that the problem of generating action abstractions can be cast as a problem of selecting a subset of pure strategies from a pool of options. We model the selection of a subset of pure strategies as a two-player game in which the strategy set of the players is the powerset of the pool of options—we call this game the subset selection game. We then present an evolutionary algorithm for solving such a game. Empirical results on small matches of μRTS show that our evolutionary approach is able to converge to a Nash equilibrium for the subset selection game. Also, results on larger matches show that search algorithms using action abstractions derived by our evolutionary approach are able to substantially outperform all state-of-the-art planning systems tested.

Introduction

Real-time planning in multi-agent scenarios such as real-time strategy (RTS) games are challenging for current search-based systems. In such domains search algorithms have to choose an action from an often large number of options, and the time allowed for planning is in the order of milliseconds. A promising approach for dealing with large action spaces in the context of real-time planning is the use of action abstractions to reduce the action space (Churchill and Buro 2013). Instead of accounting for all actions available to an agent during search, algorithms searching with action abstractions consider only a subset of the legal actions. Churchill and Buro (2013) used domain knowledge through a set of strategies to induce action abstractions, where a strategy is a function mapping a state to an action. Churchill and Buro's idea was to only consider during search the actions returned by a strategy from a set of expert-designed options, with all the other actions being ignored. Several algorithms were introduced for searching in action-abstracted state spaces (Justesen et al. 2014; Wang et al. 2016; Lelis 2017; Moraes and Lelis 2018).

The main drawback of these approaches is that they are limited to a small number of hard-coded expert-designed strategies. That is, the action-abstracted space from which a search algorithm derives a strategy in real time is limited to a small number of handcrafted strategies, which often limit the behavior of the agent controlled by the system. In this paper we introduce an approach to automate the generation of action abstractions for two-player extensive form games while still using the knowledge encoded in expert-designed strategies. Our approach generates a large pool of strategies $\mathcal{Z}$ from a small set of options $\mathcal{I}$ designed by domain experts. This is achieved by defining the strategies in $\mathcal{I}$ as rule-based strategies, which are strategies composed by a set of rules, that are themselves defined by a Boolean expression and an action to be performed by the agent. For example, in RTS games, a Boolean expression of a rule could be "controls 3 worker units", where the number 3 is a parameter of the expression, and the action the agent performs if the expression returns true could be "construct a base". By changing the parameter values of rule-based strategies we are able to generate a large pool of strategies $\mathcal{Z}$, and any non-empty subset of $\mathcal{Z}$ can be used to induce an action abstraction.

We cast the problem of generating an action abstraction from $\mathcal{Z}$ as a two-player simultaneous-move game which we name the subset selection game (SSG). In the SSG each player selects a subset of $\mathcal{Z}$, A and B, which are then used to induce an action abstraction for each player. The payoff of the SSG is defined by the result of the original two-player extensive-form game played by two versions of a planning system: one using the action abstraction induced by A and another using the action abstraction induced by B.

We introduce an evolutionary algorithm for solving the SSG and evaluate it in the domain of μRTS, an RTS game developed for research purposes. Empirical results on small μRTS matches show that our evolutionary approach is able to converge to a Nash equilibrium profile for the SSG. Results in medium-sized and large μRTS matches show that two search-based planning systems using action abstractions derived by our method are able to outperform all state-of-the-art planning systems tested in our experiments.

Related Works

All previous works that used strategies to induce action abstractions (Churchill and Buro 2013; Wang et al. 2016;

Lelis 2017; Moraes and Lelis 2018; Tavares et al. 2018) relied on a limited number of strategies, with six being the largest number of strategies considered. Instead of using only strategies designed by experts, we use a large set of strategies generated from an initial set of expert-designed strategies. Puppet Search (PS) (Barriga, Stanescu, and Buro 2017b) uses an approach similar to ours for generating strategies from a small set of rule-based strategies. Instead of searching in the actual space of the problem, PS searches in the parameter space of a set of strategies. Our approach differs from PS in that we generate novel strategies as a preprocessing step, while PS searches for novel strategies as it plays the game. Strategy Tactics (STT) (Barriga, Stanescu, and Buro 2017a) is an enhanced version of PS which combines PS's search in the parameter space with a search in the game's actual space.

Strategy Creation through Voting (SCV) also generates a large set of strategies from a small number of expert-designed ones (Silva et al. 2018). In contrast with our approach, SCV's strategies are not used to induce action abstractions, they are used to play the game directly.

Spronck et al. (2006) introduced a reinforcement learning method for exploring sequences of rules to define strategies. Others have evolved strategies (Canaan et al. 2018; Ponsen et al. 2006). In contrast with these works that derive strategies to play a game directly, we derive a set of strategies to induce action abstractions.

Action abstractions have also been applied to the domain of poker (Sandholm 2015). Action abstractions for poker are fundamentally different from our action abstractions. While most of the action abstractions for poker are manually crafted by human experts, a few works describe methods to automatically find a discretization of the continuous bet sizing in no-limit poker (Hawkin, Holte, and Szafron 2011; 2012; Brown and Sandholm 2014). By contrast, our action abstraction scheme is used to select a subset of actions from a discrete set of options. Action abstractions in poker are used to reduce the size of the game so that the abstracted game can be solved in an offline procedure, while we use action abstractions to search for a strategy in real time.

Background

Our evolutionary approach is designed to automatically craft action abstractions to real-time zero-sum simultaneous-moves extensive-form games, which can be defined by a tuple $\nabla = (\mathcal{N}, \mathcal{S}, s_{init}, \mathcal{A}, \mathcal{R}, \mathcal{T})$. Here,

- $\mathcal{N} = \{i, -i\}$ is the set of **players**.
- $\mathcal{S} = \mathcal{D} \cup \mathcal{F}$ is the set of **states**, where $\mathcal{D}$ denotes the set of **non-terminal states** and $\mathcal{F}$ the set of **terminal states**.
- $s_{init} \in \mathcal{D}$ is the start state of the game.
- $\mathcal{A} = \mathcal{A}_i \times \mathcal{A}_{-i}$ is the set of **joint actions**. $\mathcal{A}_i(s)$ is the set of legal **actions** that player i can perform in state s.
- $\mathcal{R}_i : \mathcal{F} \rightarrow \mathbb{R}$ is a **utility function** with $\mathcal{R}_i(s) = -\mathcal{R}_{-i}(s)$, for any $s \in \mathcal{F}$, since the game is zero sum.
- The **transition function** $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{A}_{-i} \rightarrow \mathcal{S}$ determines the successor state for a state s and a set of join actions taken at s.

The game is represented by a **game tree** rooted at s_{init} in which each node in the tree represents a state in $\mathcal{S}$ and every edge represents a joint action in $\mathcal{A}$. For states $s_k, s_j \in \mathcal{S}$, there exists an outgoing edge from s_k to s_j if there exists $a_i \in \mathcal{A}_i$ and $a_{-i} \in \mathcal{A}_{-i}$ such that $\mathcal{T}(s_k, a_i, a_{-i}) = s_j$. Nodes representing states in $\mathcal{F}$ are leaf nodes. We call a **decision point** for player i a state s in the tree such that i is the player to act in s. Since we consider real-time settings in this paper, each system playing the game is allowed a planning time in the order of milliseconds in every decision point.

A **pure strategy** is a function $\sigma_i : \mathcal{S} \rightarrow \mathcal{A}_i$ for player i, mapping a state s to an action a. In simultaneous-moves games one might have to play a **mixed strategy** to optimize the player's payoffs (Gintis 2000). A mixed strategy is a function $\sigma_i : \mathcal{S} \times \mathcal{A}_i \rightarrow [0, 1]$ for player i that maps a state s and an action a to the probability of the player i taking action a at state s. We denote as Σ_i and Σ_{-i} the set of all strategies for i and $-i$. A **player strategy profile** $\sigma = (\sigma_i, \sigma_{-i})$ defines the strategy of both players.

The optimal **value of the game** rooted at state s is denoted by $V^*(s)$. We denote an optimal profile $\sigma^* = (\sigma_i^*, \sigma_{-i}^*)$, known as a Nash equilibrium profile, which yields $V^*(s)$. Backward induction methods (Ross 1971) can be used to find optimal profiles. However, depending on the problem's size, finding optimal profiles becomes impractical due to the computational cost required to compute such a profile. An approach for handling very large extensive-form games is by employing action abstraction schemes to reduce the number of actions available at nodes in the game tree. One then derives strategies by searching in the action-abstracted game tree (Churchill and Buro 2013).

An **action abstraction** (or abstraction for short) for player i is defined as a function mapping the set of legal actions $\mathcal{A}(s)$ to a subset $\mathcal{A}'(s)$ of $\mathcal{A}(s)$. Previous works have shown that one can build abstractions through a collection $\mathcal{P}$ of pure strategies (Churchill and Buro 2013). The abstracted set of legal actions of player i at state s is defined as the set of actions returned by the strategies in $\mathcal{P}$,

$$\mathcal{A}'_{\mathcal{P}}(s) = \{\sigma_i(s) | \sigma_i \in \mathcal{P}\}.$$

Once an abstraction is defined, real-time planning algorithms derive a strategy for the player by searching in the abstracted game tree in every decision point of the game.

Generation of Action Abstractions as a Two-Player Zero-Sum Game

Previous works have employed a small pool of hand-crafted pure strategies to derive an action abstraction to reduce the size of the game tree. By contrast, in this paper we generate a large pool $\mathcal{Z}$ of pure strategies for the game by changing the parameter values of a small set $\mathcal{I}$ of expert-designed rule-based strategies. Then, we select a subset A of $\mathcal{Z}$ that is then used to define an action abstraction for the game. The subset A is selected for player i assuming that the opponent $-i$ is also able to select a subset B of $\mathcal{Z}$ to define $-i$'s action abstraction. Let $V^*(s, A, B)$ be the optimal value of the game rooted at state s if i and $-i$ use action abstractions induced

by A and B, respectively, defined by,

$$V^*(s,A,B) = \max_{\sigma_i \in \Sigma_i^A} \min_{\sigma_{-i} \in \Sigma_{-i}^B} \sum_{a_i \in \mathcal{A}(s)} \sum_{a_{-i} \in \mathcal{A}_{-i}(s)} \sigma_i(s,a_i) \cdot \sigma_{-i}(s,a_{-i}) \cdot V(\mathcal{T}(s,a_i,a_{-i}),A,B)\,. \quad (1)$$

The base of the recurrence is defined by $V^*(s,A,B) = \mathcal{R}_i(s)$, if $s \in \mathcal{F}$. Here, Σ_i^A and Σ_i^B are the set of all possible strategies for i and $-i$ when i is restricted by an action abstraction induced by A and $-i$ is restricted by an action abstraction induced by B. $V^*(s)$ differs from $V^*(s,A,B)$ in that the latter is the optimal value of the game while players i and $-i$ account only for actions derived from strategies in Σ_i^A and Σ_i^B, respectively, for every s. The former is the optimal value computed while accounting for all legal actions $\mathcal{A}(s)$, i.e., the un-abstracted game tree.

We model the problem of selecting a subset of strategies to induce action abstractions as a two-player zero-sum simultaneous-move game in which player i chooses a subset A that maximizes the game value, while player $-i$ chooses a subset B that minimizes the game value. We call such a game the **subset selection game** (SSG). Note that if the players optimize $V^*(s_{init},A,B)$ in the SSG, then $A = B = \mathcal{Z}$ is a trivial optimal solution, since the value of V^* can only increase for larger sets A, and only decrease for larger sets B in zero-sum games (Moraes and Lelis 2018). However, the solution $A = B = \mathcal{Z}$ is unlikely to be effective in practice as it might yield game trees that are too large to be practical, specially for planning in games with real-time constraints. Instead of choosing A and B that optimize $V^*(s_{init},A,B)$, in the SSG we choose A and B that solves the following,

$$\max_{A \in 2^{\mathcal{Z}}} \min_{B \in 2^{\mathcal{Z}}} V(s_{init},A,B)\,. \quad (2)$$

Here, $V(s_{init},A,B)$ is an approximation of $V^*(s_{init},A,B)$ as computed by a search algorithm using abstractions induced by A and B, for players i and $-i$, respectively. We write $V(A,B)$ instead of $V(s_{init},A,B)$ whenever s_{init} is clear from the context. By optimizing $V(s_{init},A,B)$ instead of $V^*(s_{init},A,B)$, we account for real-time constraints and imperfect search schemes. For example, if $A = \mathcal{Z}$ and B is a much smaller subset, then despite player i having access to a larger set of actions in the game, $V(s_{init},A,B)$ might still be negative. This is because B allows the search procedure deriving a strategy for $-i$ to focus on a promising set of actions, while i's search might spend all its time available for planning evaluating unpromising actions.

Our formulation of the SSG is simplified in the sense that we assume the existence of a solution with pure strategies (i.e., subsets of $\mathcal{Z}$), although one might need to play a mixed strategy to solve simultaneous-move games. We further observe that although we use the maxmin value in Equation 2 as the value of the SSG, we could have used the minmax value, as the maxmin and minmax values are equal in finite, two-player, zero-sum games (v. Neumann 1928).

An Evolutionary Algorithm for SSG

In this section we describe an evolutionary algorithm for solving the SSG. A high-level description of the approach is presented in Algorithm 1. The algorithm receives as input a set $\mathcal{Z}$ of pure strategies from which one can induce abstractions, a number of generations l, a fitness function Ψ, a population size n, an elitist parameter e, a tournament parameter t, and a mutation parameter $u \in [0,1]$. The algorithm returns a subset of $\mathcal{Z}$. In our evolutionary approach, subsets of $\mathcal{Z}$ are called individuals and a population P is a set of individuals. In our implementation, individuals are treated as totally-ordered subsets of $\mathcal{Z}$, as the order in which the pure strategies are placed in an individual affects the result of the genetic operators we describe below. Function INIT$(\mathcal{Z},n)$ returns n random subsets of $\mathcal{Z}$ of random size (line 2) as the initial population of the procedure.

Algorithm 1 Evolutionary Algorithm for Solving SSG

Require: Set $\mathcal{Z}$ of strategies, number of generations l, fitness function Ψ, population size n, elitist parameter e, tournament parameter t, mutation parameter u.
Ensure: Individual A.

```
1: k ← 0
2: P ← INIT(Z, n)
3: while k < l do
4:     EVALUATE(P, Ψ)
5:     M ← SELECT(P, n, e, t)
6:     M ← CROSSOVER(M)
7:     M ← MUTATION(M, Z, u)
8:     P' ← ELITISM(P, e)
9:     P ← P' ∪ M
10:    k ← k + 1
11: EVALUATE(P, Ψ)
12: return argmax_{A∈P} Ψ(A)
```

A generation is defined by an iteration of the while loop (line 3). In every generation, a set M of individuals are selected to produce another set of individuals through crossover and mutation operators (lines 5–7). Function SELECT(P,n,e,t) selects a set M of individuals with a tournament (Talbi 2009). That is, t individuals from P are randomly selected, and amongst them, the one with largest Ψ-value is added to M; this process is repeated until $|M| = n-e$, which ensures that the next population has size n, as later in the process, ELITISM adds e individuals to M.

Function CROSSOVER(M) implements a recombination procedure in which two individuals p_1 and p_2 of M are randomly selected to generate novel individuals. p_1 and p_2 are randomly divided in two disjoint parts: $p_{1,1}$ and $p_{1,2}$, and $p_{2,1}$ and $p_{2,2}$. Individuals p_1' and p_2' are generated by appending $p_{1,1}$ to $p_{2,2}$ and $p_{2,1}$ to $p_{1,2}$, respectively. Individuals p_1' and p_2' are then added to an initially empty set M'. This process is repeated until $|M'| = |M|$, when M' is returned and attributed to M (line 6). Function MUTATION$(M,\mathcal{Z},u)$ iterates through each strategy σ in each individual in M and replaces σ by a random $\sigma' \in \mathcal{Z}$ with probability u.

Function ELITISM(P,e) selects the subset P' of P with e individuals with the largest Ψ-value (line 8). The population of the next generation is determined by the union of M and P'. Since $|M| = n-e$ and $|P'| = e$, the population is always of size n. Once the procedure has completed l generations, it

returns the individual in P with the largest Ψ-value (line 12).

The Ψ-value of an individual A in P is computed as,

$$\Psi(A, P) = \sum_{B \in P \wedge B \neq A} V(s_{init}, A, B)\,.$$

The value of $V(s_{init}, A, B)$ is computed by playing the actual game ∇ (e.g., RTS match) with a search algorithm using the abstraction induced by A against the same search algorithm using an abstraction induced by B. The V-value is the final score of the game; in our implementation we use 1, 0, and -1 if the player controlled by the search algorithm using the abstraction induced by A wins, draws, or loses the match, respectively. The Ψ-value of A is the sum of the V-values of A with each $B \neq A$ in the population.

Evolutionary Stable Strategies

In our experiments we use the concept of Evolutionary Stable Strategies (ESS) for symmetric games, introduced by Maynard Smith and Price (1973), to evaluate the solutions encountered by our approach for SSG. A two-player game is symmetric if it has the same set of strategies for both players and if the utilities of the game are symmetric, i.e., players payoffs are the same if they are "player 1" or "player 2". Our SSGs have the same set of strategies for both players (the powerset of $\mathcal{Z}$), but there could be asymmetries in the utility function of the SSG, i.e., the result of a match played by the same search algorithm using different abstractions. For example, in RTS games one of the players could be in advantage depending on the location she starts the game. Instead of computing the value of V directly with a single match, we mitigate possible asymmetries by computing V as the average result of two matches in which we swap the roles of "player 1" and "player 2" in the matches. This procedure allows us to ensure that our SSG is symmetric.

A pure strategy A is an ESS if a population composed of A (population in the sense of set P in Algorithm 1) cannot be "invaded" by another strategy B during an evolutionary process. That is, if B appears in the population, then it is quickly eliminated through the evolutionary process for being "less fit to survival" than A. Formally, we have the following definition adapted from Maynard Smith and Price (1973).

Definition 1 (ESS) *Let ∇ be a zero-sum extensive-form game with start state s_{init}. Let A be a strategy for the SSG defined over ∇ and a set of strategies $\mathcal{Z}$ for ∇. Strategy A is an Evolutionary Stable Strategy (ESS) if the following conditions hold, for any strategy $B \neq A$ of the SSG.*

$$V(A, A) \geq V(B, A) \tag{3}$$

$$V(A, A) = V(B, A) \Rightarrow V(A, B) > V(B, B) \tag{4}$$

If Equation 3 is satisfied, then profile (A, A) is a Nash equilibrium profile for the SSG. Equation 4 is a stability condition that guarantees that newly inserted strategies (mutants) are repealed by the ESS. That is, even if a mutant B is as good as the A against A, then B is still repealed for being strictly worse than A against B. In our experiments we are particularly interested in observing if the populations in our evolutionary approach present the characteristics of an ESS, i.e., if the population is composed solely of a single strategy and if other strategies are quickly eliminated.

Generation of Novel Strategies from Expert-Designed Rule-Based Strategies

Here we define rule-based strategies and how a small number of them can be used to generate a large set of strategies.

Definition 2 *A rule-based strategy σ^r for ∇ is defined by a totally-ordered set of rules $\mathcal{E} = (\mathcal{B}, \mathcal{A}^r)$, where,*

- *$\mathcal{B} : \mathcal{S} \rightarrow \{0, 1\}$ is a function that returns, for state s, the result of a Boolean expression that depends on a set of parameters $\{\mathcal{C}_1, \cdots, \mathcal{C}_h\}$. Here, each $\mathcal{C}_j$ can assume K_j different values $\mathcal{C}_j = \{v_j^1, \cdots, v_j^{K_j}\}$, and $\mathcal{C}^T = \{(v_1, \cdots, v_n) \in \mathcal{C}_1 \times \cdots \times \mathcal{C}_n\}$ is the set of possible combinations of value assignments for the parameters.*
- *$\mathcal{A}^r : \mathcal{S} \rightarrow \mathcal{A}$ is a function that receives a state s as input and returns a legal action a to be played in s.*

σ^r is used to play ∇ by iteratively invoking the $\mathcal{B}$ function of each rule $\mathcal{E}$ in σ^r, for each decision point s in the game. One then executes in s the action returned by $\mathcal{A}^r(s)$ for the first rule $\mathcal{E}$ whose $\mathcal{B}$ function returns true. The order in which the $\mathcal{B}$ functions are invoked is defined by σ^r's total ordering.

A rule-based strategy σ^r can be used to define a possibly large set of strategies by using different parameter values $(v_1, \cdots, v_n)$ in $\mathcal{C}^T$ in each rule $\mathcal{E}$ of σ^r. For example, a rule-based strategy for an RTS game could have the following rule $\mathcal{E}$: "if controlling 2 combat units, then attack the enemy". Here, $\mathcal{B}$ is "controlling 2 combat units", where the number of units is a parameter $\mathcal{C}$ that can assume any value $v > 0$. The action a of $\mathcal{E}$ is "attack the enemy". By changing the parameter value from 2 to 3, one generates another instantiation of the rule-based strategy to play the game.

In our experiments we use a small set of expert-designed rule-based strategies to automatically generate a large set of strategies by changing the parameter values of the initial strategies. The resulting strategies compose our set $\mathcal{Z}$.

Empirical Methodology

Problem Domain All our experiments are run on μRTS, an RTS game developed for research (Ontañón 2013). μRTS allows one to test algorithms without having to deal with engineering problems encountered in commercial video games. Moreover, there is an active community using μRTS as research testbed, with competitions being organized (Ontañón et al. 2018), which helps organizing all methods in a single codebase.[1] In μRTS each player starts the game controlling a set of worker units on a gridded map. Workers can be used to collect resources, build structures, and battle the opponent. In some of the maps, players also start with a structure called base, which is used to train workers and to store resources. In addition to the base, another structure workers can build is called barracks, which can be used to train combat units. μRTS has the following combat units: light, ranged, and heavy. Combat units differ in how much damage they can take before being removed from the game, how much damage they can inflict to other units, and how close to an opponent unit u they must be to attack u.

[1] https://github.com/santiontanon/microrts

A player wins a μRTS match if she is able to remove all the other player's units from the game. Every algorithm is allowed 100 milliseconds for planning in each decision point.

Set $\mathcal{Z}$ of Strategies We use an initial set $\mathcal{I}$ of rule-based strategies for μRTS as defined by the strategies of Silva et al. (2018). Their strategies can be described as 3 rule-based strategies with different parameter sets. We call their strategies Rush, Defense, and Military. As an example, the Boolean expression of a rule of the Rush strategy verifies if the player has a minimum number n of units of a given type before attacking the opponent. The parameters for this rule are the type of units trained and the number of units required to trigger an attack. Silva et al. used 4 strategies derived from Rush, all of them with $n = 1$. Their Rush strategies differed only on the type of unit trained, with one strategy training each of the types: worker, light, ranged, and heavy—these strategies are named Worker Rush (WR), Light Rush (LR), Ranged Rush (RR), and Heavy Rush (HR). One can generate novel rush strategies by changing not only the type of unit trained, but also the value of n. Similarly, one can define novel Defense and Military strategies by changing their parameters. In this paper we vary the parameters of Rush, Defense, and Military to generate 100 strategies of each type, which results in a $\mathcal{Z}$ set of 300 strategies.[2]

Search Algorithms We use Portfolio Greedy Search (PGS) (Churchill and Buro 2013) and Stratified Strategy Selection (SSS) (Lelis 2017) in our experiments. We use PGS and SSS because both algorithms were developed to search in action-abstracted state spaces induced by strategies. Note, however, that one could use other algorithms such as A3N (Moraes et al. 2018) to search in our action-abstracted spaces. We denote as PGS* and SSS* the search algorithms using an action abstraction derived by our evolutionary procedure. When deriving PGS*'s action abstraction, we use PGS to compute the values of V in the evolutionary procedure. Similarly, we use SSS to compute the values of V when deriving SSS*'s action abstraction.

As evaluation function for PGS, PGS*, SSS, and SSS*, we perform two random and independent play-outs with 200 game cycles of length from the state being evaluated. The evaluated value is the average value of the states reached through the play-outs according to the built-in *SimpleSqrtEvaluationFunction3* function of μRTS (Ontañón 2017).

Maps μRTS matches can be played in maps of different sizes. We test our action abstractions in maps of size $x \times x$ with $x \in \{8, 24, 32, 64\}$, where the map of 64×64 is a map from Blizzard's StarCraft. All maps we use are symmetric, meaning that the players are indifferent to being assigned the role of "player 1" or "player 2". In addition to being algorithmic dependent, all action abstractions are map dependent. That is, we run our evolutionary approach to generate an action abstraction for a given search algorithm and map. Every match played is limited by a number of game cycles and gameplay time. We use $(4000, 20)$, $(6000, 20)$, $(7000, 25)$, and $(12000, 30)$ as the limit values in our experiments, where the first number specifies the limit in game cycles and the second number the limit in minutes for maps of size x of 8, 24, 32, and 64, respectively.

Experiment Sets Our empirical evaluation is divided into two parts. In the first part we apply our evolutionary algorithm for solving the SSG in a μRTS map of size 8×8, which is small enough for us to compute a Nash equilibrium profile for the SSG. This experiment allows us to verify that our approach is able to converge to an ESS. We use SSS as the search algorithm and sets $\mathcal{Z}$ of size 8, 10, and 15. The sets $\mathcal{Z}$ are also derived from the Rush, Defense, and Military rule-based strategies and all sets include the WR strategy.

In the second part we test the performance of PGS* and SSS* against state-of-the-art methods. We use in our experiments the MCTS version of Puppet Search (PS) (Barriga, Stanescu, and Buro 2017b), Strategy Tactics (STT) (Barriga, Stanescu, and Buro 2017a), which was the winner of the 2017 μRTS competition (Ontañón et al. 2018), Adversarial Hierarchical Task Network (AHT) (Ontañón and Buro 2015), NaïveMCTS (Ontañón 2017), Strategy Creation Through Voting (SCV) (Silva et al. 2018), the four instantiations of the Rush strategy explained above, WR, LR, HR and RR (Stanescu et al. 2016), which are known to perform well in μRTS (Ontañón et al. 2018). Also, we include in our comparison versions of PGS and SSS that use an action abstraction induced by the set {WR, LR, HR, RR}, which we denote as PGS^r and SSS^r, where "r" stands for Rush. We also use versions of PGS and SSS that use an action abstraction induced by 9 strategies, which are derived from the same Rush, Defense, and Military rule-based strategies used to generate our set $\mathcal{Z}$. We denote the variants of PGS and SSS that use the action abstraction induced by these 9 strategies as PGS^s and SSS^s, where "s" refers to SCV, as these 9 strategies are those used by SCV (Silva et al. 2018). Namely, PGS^s and SSS^s use the same 4 Rush strategies used with PGS^r and SSS^r, 4 Defense strategies, and 1 Military strategy; see Silva et al. (2018) for details. As explained before, the action abstractions we use with PGS* and SSS* are defined by our evolutionary approach with a $\mathcal{Z}$ set with 300 strategies, for all maps tested.

We compare an algorithm to another by simulating several matches between the two algorithms. Although all our maps are symmetric, to guarantee fairness, in our experiments a system always plays half of the simulated matches as "player 1" and the other half as "player 2".

Evolutionary Parameters In the first experiment we use a mutation probability u of 0.05, size of the population n of 100, the number of generations l of 20, and the size e of the elite population is set to 25. Since the maps are much larger in the second experiment, we set $n = 20$, $l = 30$, and $e = 5$.

Empirical Results

First Part: Convergence to an ESS

Due to the real-time constraints, the order in which the strategies are used to induce an abstraction matters to search

[2] See our codebase for details of how the 300 strategies are generated: https://github.com/julianmarino/evolutionary-action-abstractions.git

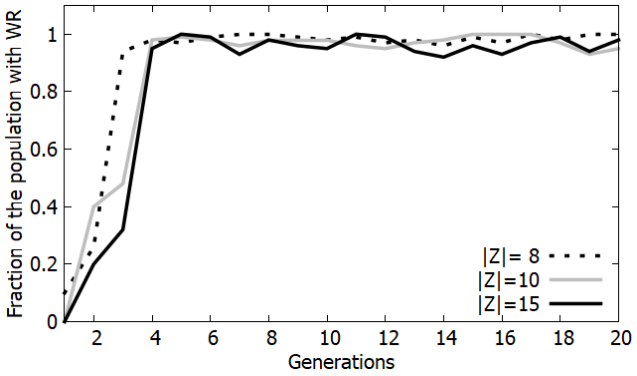

Figure 1: Fraction of the population representing WR.

algorithms. For example, let σ_1^r and σ_2^r be two rule-based strategies used to induce an abstraction. The abstraction induced with $\{\sigma_1^r, \sigma_2^r\}$ could be different from the abstraction induced with $\{\sigma_2^r, \sigma_1^r\}$ as, depending on the game state, the search algorithm might be able to evaluate only the action given by the first strategy, which could be the action given by strategy σ_1^r or σ_2^r, depending on the abstraction's ordering. This means that, instead of having the powerset of $\mathcal{Z}$ as the strategy set for the SSG, one has all permutations of all subsets of $\mathcal{Z}$ as the strategy set for the SSG. Our evolutionary procedure already deals with this combinatorial explosion as the individuals in the population are treated as totally ordered subsets of $\mathcal{Z}$.

In this first experiment we show that WR is likely to be a strictly dominant strategy for the SSG defined by the set $\mathcal{Z}$ of size 8. For that, we tested SSS with an abstraction induced by each permutation of subsets of the set $\mathcal{Z}$ against SSS with an abstraction induced by WR. Since SSS uses a stochastic evaluation function, we repeated each match 50 times. Ignoring matches that finish in a draw (i.e., matches that reach the map's maximum number of game cycles or time limit), SSS with an abstraction induced by WR is able to win more matches than lose against SSS with any other permutation, with $p < .001$, for the largest p, according to two-sided binomial tests. Thus, with high probability, (WR, WR) is a Nash equilibrium profile for the SSG with $|Z| = 8$.

Figure 1 shows the fraction of WR in the population in 20 generations for the three SSGs tested. The plot suggests that WR is stable after generation 4 for all SSGs. We repeated this experiment 5 times for $|\mathcal{Z}| = 8$ and the fraction of WR in the population quickly approached one in all runs. Figure 1 shows a representative run of our system for that SSG.

Second Part: Comparison with the State of the Art

Since our action abstractions are generated by a stochastic approach, we perform 5 independent runs of the experiment described in this section and report the average percentage of victories. In this experiment we follow a previous work (Ontañón 2017) and draws are counted as a 0.5 victory for both systems. In each run of this experiment we have each system play every other system 100 times on a given map.

Table 1 shows the average percentage of victories of the column player against the row player for the three maps used in our experiment. As an example of how to read the table, PGS* wins on average 98.4 percent of its matches against SSS^s in the 24 × 24 map. We highlight the background of cells showing the results of both PGS* and SSS*, the two search methods using action abstractions derived by our evolutionary method. We also present as column players the four baselines, which are the same search algorithms

		PGS^r	PGS^s	PGS*	SSS^r	SSS^s	SSS*
Map 24 × 24	PS	81.8	81.6	99.8	57.1	75.2	99.2
	AHT	100.0	99.8	96.8	100.0	99.2	95.6
	STT	97.4	91.8	99.8	44.9	87.6	98.6
	NS	98.9	98.9	100.0	86.5	98.0	100.0
	RR	100.0	100.0	100.0	81.2	100.0	100.0
	HR	100.0	100.0	100.0	97.0	99.8	100.0
	WR	60.4	90.3	78.8	44.8	82.6	66.6
	LR	92.9	88.6	100.0	19.8	84.8	99.2
	SCV	99.8	99.2	98.2	95.0	98.2	97.4
	PGS^r	-	42.4	99.6	34.7	40.7	93.5
	PGS^s	57.6	-	97.0	43.0	45.0	91.4
	PGS*	0.4	3.0	-	5.4	1.6	41.7
	SSS^r	65.3	57.0	94.6	-	40.7	90.5
	SSS^s	59.3	55.0	98.4	59.3	-	93.2
	SSS*	6.5	8.6	58.3	9.5	6.8	-

		PGS^r	PGS^s	PGS*	SSS^r	SSS^s	SSS*
Map 32 × 32	PS	37.9	48.4	99.8	52.1	47.8	99.6
	AHT	100.0	100.0	100.0	100.0	100.0	100.0
	STT	84.8	22.7	100.0	88.6	23.4	100.0
	NS	99.2	99.3	100.0	99.6	98.9	100.0
	RR	100.0	100.0	100.0	100.0	100.0	100.0
	HR	98.8	99.2	100.0	99.8	98.0	100.0
	WR	100.0	100.0	94.2	100.0	100.0	100.0
	LR	1.8	8.2	96.8	16.2	7.2	99.0
	SCV	11.9	29.4	86.5	56.5	27.8	91.5
	PGS^r	-	55.2	99.6	96.0	55.7	99.6
	PGS^s	44.8	-	99.6	97.2	51.0	100.0
	PGS*	0.4	0.4	-	1.0	0.0	51.6
	SSS^r	4.0	2.8	99.0	-	2.4	99.8
	SSS^s	44.3	49.0	100.0	97.6	-	100.0
	SSS*	0.4	0.0	48.2	0.2	0.0	-

		PGS^r	PGS^s	PGS*	SSS^r	SSS^s	SSS*
Map 64 × 64	PS	0.3	0.0	83.8	0.0	0.0	77.5
	AHT	100.0	100.0	99.5	100.0	100.0	100.0
	STT	25.2	0.3	95.2	20.7	2.0	91.0
	NS	100.0	100.0	100.0	100.0	100.0	100.0
	RR	87.7	100.0	100.0	94.7	98.3	100.0
	HR	99.7	100.0	100.0	99.3	100.0	100.0
	WR	60.5	82.0	98.8	75.7	72.3	97.7
	LR	54.3	78.0	100.0	59.0	66.0	100.0
	SCV	1.5	1.5	86.0	9.3	1.8	73.8
	PGS^r	-	50.3	100.0	68.0	50.7	100.0
	PGS^s	49.7	-	100.0	67.0	45.3	100.0
	PGS*	0.0	0.0	-	0.0	0.0	36.8
	SSS^r	32.0	33.0	100.0	-	39.5	100.0
	SSS^s	49.3	54.7	100.0	60.5	-	100.0
	SSS*	0.0	0.0	62.8	0.0	0.0	-

Table 1: Average percentage of matches won by the column player against the row player for maps of different sizes.

but using handcrafted actions abstractions: PGS^r, PGS^s, SSS^r, and SSS^s. The numbers are truncated to one decimal place. All PGS* and SSS* results are significant with $p < .05$ according to pairwise comparisons performed with unpaired t-tests. Each t-test compares the average percentage of matches won by a column player against a row player.

Comparison with Baselines We start by comparing PGS* and SSS* with their baselines: PGS^r, PGS^s, SSS^r, and SSS^s. One observes that PGS* and SSS* are able to substantially outperform the baselines in direct matches. This can be observed in the table of results by looking at PGS*'s and SSS*'s highlighted rows. SSS^r is the baseline that wins the largest number of matches against either PGS* and SSS*, and that is only 9.5% of the matches played against SSS* in the 24×24 map. The percentage of matches won by a baseline against one of our methods reduces as one increases the size of the map. In particular, PGS* and SSS* win all their matches against the baselines in maps of size 64×64.

PGS* and SSS* can also be compared with the baselines by observing the average winning percentage of the PGS and SSS variants against the other approaches. In general, PGS* and SSS* win more matches against other approaches than the baselines. A notable exception are the results against WR in the 24×24 map, where the baselines PGS^s and SSS^s win more matches than PGS* and SSS*. This is because the strategies used by PGS^s and SSS^s are quite effective against WR, but less effective in general—PGS* and SSS* are much stronger than PGS^s and SSS^s against the other methods. These results highlight that the action abstractions derived by our approach can be far superior to those derived from existing strategies. This might be because existing strategies (e.g., those used in SCV) are designed to play the game directly, and not to be combined with other strategies through action abstractions. Since we use a large set of strategies, our approach might encounter strategies that if combined to induce an action abstraction, provide a search algorithm with a set of promising and complementary actions.

Comparison with the State of the Art STT was the winner of the 2017 μRTS competition (Ontañón et al. 2018). Both PGS* and SSS* substantially outperform STT in all maps tested, with 91% being the smallest average percentage of victories obtained by one of our approaches, which occurred in matches played by SSS* in the 64×64 map. Our approaches also outperform by a large margin PS, which is a previous version of STT. Our scheme for generating our set $\mathcal{Z}$ is similar to the search performed by PS and STT as they also alter the parameter values of expert-designed strategies. One possible explanation for these results is that our approach of using a set of strategies to induce an action abstraction as a preprocessing step can be more effective than the online searching for parameter values. We believe this to be true because our evolutionary method filters out weak strategies while solving the SSG, thus allowing the planning systems to focus their search on more promising actions. By contrast, PS and STT might use the limited time available for planning to evaluate weak strategies.

SCV (Silva et al. 2018) and LR also performed quite well in the 2017 μRTS competition (Ontañón et al. 2018). PGS* and SSS* are able to outperform both of them also by a large margin. The comparison with SCV is important because the rule-based strategies we use to generate our set $\mathcal{Z}$ is derived from the strategies used in SCV. Moreover, SCV also generates a large set of strategies from a small set of options. The superior performance of our approaches against SCV suggests that our system for generating a large set of strategies and thus selecting a subset to induce action abstractions can be superior to other schemes that generate a large set of strategies but use them to play the game directly.

Limitations

Although our evolutionary method for generating action abstractions allowed existing algorithms to substantially outperform state-of-the-art methods in the problem domain of μRTS, our approach still has limitations, which point to interesting directions of future research. Although we are able to generate a large number of strategies from a small number of existing ones, the strategies we generate to compose $\mathcal{Z}$ are still somewhat restricted to the initial set of expert-designed strategies (e.g., the strategies we generate by changing the parameters of rule-based strategies are still restricted to the actions from the initial set of strategies). We expect to generate stronger action abstractions with our evolutionary approach should the set $\mathcal{Z}$ contained a larger diversity of strategies. Another limitation of our approach is due to the fact that our approach considers only pure strategies for the SSG (i.e., a single subset of strategies from $\mathcal{Z}$ is returned after the last generation of our method). Since the SSG is a simultaneous-move game, one might need to play a mixed strategy to optimize her payoffs. We intend to investigate the use of mixed strategies for the SSG in future works.

Conclusions

In this paper we showed that the problem of generating action abstractions for extensive-form games can be cast as a problem of selecting a subset of strategies from a pool $\mathcal{Z}$ of options. We then modeled the subset selection problem as a two-player game, which we named SSG. We presented an evolutionary procedure to solve SSGs. Another contribution of this paper was to show that a large set $\mathcal{Z}$ of strategies can be generated by a small set of what we called rule-based strategies, by changing the parameter values of the strategies. Empirical results on a small map of μRTS showed that our method is able to converge to what is likely to be an optimal solution to the SSG. Results on larger maps showed that PGS and SSS using action abstractions generated by our approach was able to outperform versions of the same algorithms using existing action abstractions. Our results also showed that PGS and SSS with our action abstractions outperformed all state-of-the-art systems tested, including the best performing methods from the 2017 μRTS competition.

Acknowledgements This research was partially supported by CNPq, Capes, and FAPEMIG. The research was carried out using the computational resources of the Center for Mathematical Sciences Applied to Industry (CeMEAI)

funded by FAPESP (grant 2013/07375-0), and the cluster Jupiter from Universidade Federal de Viçosa. We thank the anonymous reviewers for great suggestions.

References

Barriga, N. A.; Stanescu, M.; and Buro, M. 2017a. Combining strategic learning and tactical search in real-time strategy games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 9–15. AAAI.

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* 69–77.

Brown, N., and Sandholm, T. 2014. Regret transfer and parameter optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 594–601.

Canaan, R.; Shen, H.; Torrado, R.; Togelius, J.; Nealen, A.; and Menzel, S. 2018. Evolving agents for the hanabi 2018 cig competition. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 1–8. IEEE.

Churchill, D., and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *Proceedings of the Conference on Computational Intelligence in Games*, 1–8. IEEE.

Gintis, H. 2000. *Game Theory Evolving: A Problem-centered Introduction to Modeling Strategic Behavior*. Economics / Princeton University Press. Princeton University Press.

Hawkin, J. A.; Holte, R.; and Szafron, D. 2011. Automated action abstraction of imperfect information extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 681–687.

Hawkin, J. A.; Holte, R.; and Szafron, D. 2012. Using sliding windows to generate action abstractions in extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1924–1930.

Justesen, N.; Tillman, B.; Togelius, J.; and Risi, S. 2014. Script- and cluster-based UCT for StarCraft. In *IEEE Conference on Computational Intelligence and Games*, 1–8.

Lelis, L. H. S. 2017. Stratified strategy selection for unit control in real-time strategy games. In *International Joint Conference on Artificial Intelligence*, 3735–3741.

Maynard Smith, J., and Price, G. R. 1973. The logic of animal conflict. *Nature* 246:15–18.

Moraes, R. O., and Lelis, L. H. S. 2018. Asymmetric action abstractions for multi-unit control in adversarial real-time games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 876–883. AAAI.

Moraes, R. O.; Mariño, J. R. H.; Lelis, L. H. S.; and Nascimento, M. A. 2018. Action abstractions for combinatorial multi-armed bandit tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 74–80. AAAI.

Ontañón, S., and Buro, M. 2015. Adversarial hierarchical-task network planning for complex real-time games. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1652–1658.

Ontañón, S.; Barriga, N. A.; Silva, C. R.; Moraes, R. O.; and Lelis, L. H. S. 2018. The first microrts artificial intelligence competition. *AI Magazine* 39(1):75–83.

Ontañón, S. 2013. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 58–64. AAAI.

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

Ponsen, M.; Munoz-Avila, H.; Spronck, P.; and Aha, D. W. 2006. Automatically generating game tactics through evolutionary learning. *AI Magazine* 27(3):75–84.

Ross, S. M. 1971. Goofspiel – the game of pure strategy. *Journal of Applied Probability* 8(3):621–625.

Sandholm, T. 2015. Abstraction for solving large incomplete-information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 4127–4131.

Silva, C. R.; Moraes, R. O.; Lelis, L. H. S.; and Gal, Y. 2018. Strategy generation for multi-unit real-time games via voting. *IEEE Transactions on Games*.

Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2006. Adaptive game ai with dynamic scripting. *Machine Learning* 63(3):217–248.

Stanescu, M.; Barriga, N. A.; Hess, A.; and Buro, M. 2016. Evaluating real-time strategy game states using convolutional neural networks. In *Proceedings IEEE Conference on Computational Intelligence and Games*, 1–7. IEEE.

Talbi, E.-G. 2009. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.

Tavares, A. R.; Anbalagan, S.; Marcolino, L. S.; and Chaimowicz, L. 2018. Algorithms or actions? a study in large-scale reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2717–2723.

v. Neumann, J. 1928. Zur theorie der gesellschaftsspiele. *Mathematische Annalen* 100(1):295–320.

Wang, C.; Chen, P.; Li, Y.; Holmgård, C.; and Togelius, J. 2016. Portfolio online evolution in StarCraft. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment*, 114–120. AAAI.