Proceedings of the Fourteenth Artificial Intelligence and
Interactive Digital Entertainment Conference (AIIDE 2018)

Nested-Greedy Search for Adversarial Real-Time Games

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

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

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

Abstract

Churchill and Buro (2013) launched a line of research through Portfolio Greedy Search (PGS), an algorithm for adversarial real-time planning that uses scripts to simplify the problem's action space. In this paper we present a problem in PGS's search scheme that has hitherto been overlooked. Namely, even under the strong assumption that PGS is able to evaluate all actions available to the player, PGS might fail to return the best action. We then describe an idealized algorithm that is guaranteed to return the best action and present an approximation of such algorithm, which we call Nested-Greedy Search (NGS). Empirical results on μRTS show that NGS is able to outperform PGS as well as state-of-the-art methods in matches played in small to medium-sized maps.

Real-time strategy (RTS) games are challenging for artificial intelligence (AI) methods. A chief difficulty faced by AI methods is the large action space encountered in such games. Churchill and Buro (2013) launched a line of research for dealing with a game's large action space by using expert-designed scripts. Scripts are designed to play RTS games by following simple rules such as *"do not attack an enemy unit u if an ally unit will already cause enough damage to eliminate u from game"*. Instead of playing the game directly with a script, Churchill and Buro used a set of scripts to define which actions should be considered during search. This way, instead of considering all legal actions during search, Churchill and Buro's Portfolio Greedy Search (PGS) considers only the actions returned by the set of scripts.

Several researchers were inspired by Churchill and Buro's work and developed other search algorithms that use the same principle of employing a set of scripts to reduce the action space in RTS games (Justesen et al. 2014; Wang et al. 2016; Lelis 2017; Moraes and Lelis 2018). In this paper we present a problem in PGS's search scheme that has hitherto been overlooked. Namely, even under the strong assumption that PGS is able to evaluate all actions considered by its set of scripts, the algorithm is not guaranteed to return the best available action at a given state. We call this issue the non-convergence problem. The non-convergence problem is related to how PGS handles the responses of the player's opponent and it might cause the algorithm to present pathological results. That is, the algorithm can produce worse results if allowed more computation time. We show empirically in the context of μRTS, a minimalist RTS game for research purposes, that PGS's pathology is very common in practice.

In this paper we also present a search algorithm called Nested-Greedy Search (NGS) to overcome PGS's non-convergence problem. NGS is similar to PGS, with the only difference being how the algorithm handles the enemy responses during search. In contrast with PGS, NGS approximates how the opponent could best respond to different actions of the player and returns the action that yields the largest payoff for the player, assuming the opponent will play an approximated best response. We evaluated NGS in μRTS matches. Our empirical results show that NGS is able to outperform not only PGS, but all state-of-the-art methods tested in matches played in small to medium-sized maps.

In addition to presenting the non-convergence problem as well as a search algorithm to overcome the problem, another contribution of this work is to show that PGS and NGS can be used to play entire RTS matches. This is important because PGS was developed to control units in combat scenarios that arise in RTS games, and not to play entire RTS matches, which requires one to deal with the economical side of the game in addition to the military side of the game. Our work suggests that other researchers should consider PGS, NGS, and other algorithms derived from PGS as competing methods for their planning systems for RTS games.

Related Work

After PGS, several researchers developed search algorithms that also used scripts to filter the set of actions considered during search. Justesen et al. (2014) introduced two variations of UCT (Kocsis and Szepesvári 2006) for searching in the action space filtered by scripts. Wang et al. (2016) introduced Portfolio Online Evolution (POE) a local search algorithm also designed for searching in the script-reduced action spaces. Lelis (2017) introduced Stratified Strategy Selection, a greedy algorithm that uses a type system to search in the action space given by a set of scripts. Moraes and Lelis (2018) introduced search algorithms that search in asymmetrically action-abstracted spaces, which were induced by scripts. Moraes et al. (2018) extended combinatorial multi-armed bandit tree search algorithms (Ontañón 2017) to also search in asymmetrically action-abstracted spaces induced by scripts. Although all these works built di-

rectly on the work of Churchill and Buro (2013), they overlooked PGS's non-convergence problem.

Other works have used expert-designed scripts differently. For example, Puppet Search (Barriga, Stanescu, and Buro 2017b) defines a search space over the parameter values of scripts. Similarly to Puppet Search, Strategy Tactics (STT) (Barriga, Stanescu, and Buro 2017a) also searches in the space of parameter values of scripts. However, Strategy Tactics balances the search over the space of parameters with a search in the actual state space with NaïveMCTS (Ontañón 2017). Silva et al. (2018) introduced Strategy Creation via Voting, a method that uses a set of scripts with a voting system to generate novel scripts that can be used to play RTS games. We show empirically that NGS is able to outperform these approaches in small to medium-sized maps.

Before the adoption of scripts to guide search algorithms to play RTS games, state-of-the-art methods included search algorithms that accounted for the entire action space, such as Monte Carlo (Chung, Buro, and Schaeffer 2005; Sailer, Buro, and Lanctot 2007; Balla and Fern 2009; Ontañón 2013) and Alpha-Beta (Churchill, Saffidine, and Buro 2012). However, in contrast with methods that use scripts to reduce the action space, Alpha-Beta and Monte Carlo methods perform well only in very small RTS matches in which ones controls a small number of units.

Background

Definitions and Notation

An RTS match can be described as a finite zero-sum two-player simultaneous-move game, and be denoted as $(\mathcal{N}, \mathcal{S}, s_{init}, \mathcal{A}, \mathcal{R}, \mathcal{T})$, where,

- $\mathcal{N} = \{i, -i\}$ is the set of players, where i is the player we control and $-i$ is our opponent.
- $\mathcal{S} = \mathcal{D} \cup \mathcal{F}$ is the set of states, where $\mathcal{D}$ denotes the set of non-terminal states and $\mathcal{F}$ the set of terminal states. Every state $s \in \mathcal{S}$ includes the joint set of units $\mathcal{U}^s = \mathcal{U}^s_i \cup \mathcal{U}^s_{-i}$, for players i and $-i$, respectively. We write $\mathcal{U}$, $\mathcal{U}_i$, and $\mathcal{U}_{-i}$ whenever the state s is clear from the context.
- $s_{init} \in \mathcal{D}$ is the start state of a match.
- $\mathcal{A} = \mathcal{A}_i \times \mathcal{A}_{-i}$ is the set of joint player-actions. $\mathcal{A}_i(s)$ is the set of legal player-actions i can perform at state s. Each player-action $a \in \mathcal{A}_i(s)$ is denoted by a vector of n unit-actions $(m_1, \cdots, m_n)$, where $m_k \in a$ is the unit-action of the k-th ready unit of player i. We write "action" instead of "player-action" or "unit-action" if it is clear from the context that we are referring to a player-action or unit-action. A unit u is not ready at s if u is performing an action (e.g., a worker might be constructing a base and is unable to perform another action). We denote the set of ready units of players i and $-i$ at state s as $\mathcal{U}^{r,s}_i$ and $\mathcal{U}^{r,s}_{-i}$ and write $\mathcal{U}^r_i$ and $\mathcal{U}^r_{-i}$ if the state is clear from the context. For unit u, we write $a[u]$ to denote the action of u in a.
- $\mathcal{R}_i : \mathcal{F} \to \mathbb{R}$ is a utility function with $\mathcal{R}_i(s) = -\mathcal{R}_{-i}(s)$, for any $s \in \mathcal{F}$, as matches are zero-sum games.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A}_i \times \mathcal{A}_{-i} \to \mathcal{S}$ is the transition function, which determines the sucessor of a state s for a set of joint actions taken at s.

Algorithm 1 PORTFOLIO GREEDY SEARCH (PGS)

Require: state s, ready units $\mathcal{U}^r_i = \{u^1_i, \cdots, u^{n_i}_i\}$ and $\mathcal{U}^r_{-i} = \{u^1_{-i}, \cdots, u^{n_{-i}}_{-i}\}$ in s, set of scripts $\mathcal{P}$, evaluation function Ψ, integers I and R, and time limit t.
Ensure: action a for player i's units.
1: $\bar{\sigma}_i \leftarrow$ choose a script from $\mathcal{P}$ *//see text for details*
2: $\bar{\sigma}_{-i} \leftarrow$ choose a script from $\mathcal{P}$ *//see text for details*
3: $a_i \leftarrow \{\bar{\sigma}_i(u^1_i), \cdots, \bar{\sigma}_i(u^{n_i}_i)\}$
4: $a_{-i} \leftarrow \{\bar{\sigma}_{-i}(u^1_{-i}), \cdots, \bar{\sigma}_{-i}(u^{n_{-i}}_{-i})\}$
5: $a_i \leftarrow$ IMPROVE($s, \mathcal{U}^r_i, \mathcal{P}, a_i, a_{-i}, \Psi, I, t$)
6: **for** $r \leftarrow 0$ to R **do**
7: $\quad a_{-i} \leftarrow$IMPROVE($s, \mathcal{U}^r_{-i}, \mathcal{P}, a_{-i}, a_i, \Psi, I, t$)
8: $\quad a_i \leftarrow$ IMPROVE($s, \mathcal{U}^r_i, \mathcal{P}, a_i, a_{-i}, \Psi, I, t$)
9: **return** a_i

A pure strategy is a function $\sigma : \mathcal{S} \to \mathcal{A}_i$ for player i mapping a state s to an 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), similarly to other RTS methods (Churchill, Saffidine, and Buro 2012; Churchill and Buro 2013; Wang et al. 2016; Ontañón 2017; Barriga, Stanescu, and Buro 2017b; Lelis 2017), we consider only pure strategies in this paper. A script $\bar{\sigma}$ is a function mapping a state s and a unit u in s to an action for u. A script $\bar{\sigma}$ allows one to define a strategy σ by applying $\bar{\sigma}$ to every ready unit in the state. We write $\bar{\sigma}$ instead of $\bar{\sigma}(s, u)$ whenever s and u are clear from the context. At every state s, search algorithms such as PGS assign a script $\bar{\sigma}$ from a collection of scripts, denoted $\mathcal{P}$, to every ready unit u in s. Unit u then performs the action returned by $\bar{\sigma}(s, u)$.

Portfolio Greedy Search (PGS)

Algorithm 1 and 2 show the pseudocode of PGS. PGS receives as input player i's and $-i$'s set of ready units for a given state s, denoted $\mathcal{U}^r_i$ and $\mathcal{U}^r_{-i}$, a set of scripts $\mathcal{P}$, and an evaluation function Ψ, which receives a state s' as input and estimates the end-game utility for player i if the game continues from s'. PGS also receives as input two integers, R and I. R controls PGS's search effort for computing player $-i$'s best response to player i's action and I controls PGS's search effort for computing a best response for player $-i$'s action. Finally, PGS receives as input a time limit t, which caps the algorithm's running time. PGS returns an action vector a for player i to be executed in s. PGS can be divided in two steps, the configuration of the *seeds* of the two players and an improvement process. Next, we describe these steps.

Configuration of the Seeds PGS starts by selecting the script $\bar{\sigma}_i$ (resp. $\bar{\sigma}_{-i}$) from $\mathcal{P}$ that yields the largest Ψ-value when i (resp. $-i$) executes a player-action composed of unit-actions computed with $\bar{\sigma}_i$ (resp. $\bar{\sigma}_{-i}$) (see lines 1 and 2 of Algorithm 1) for all units in $\mathcal{U}^r_i$ (resp. $\mathcal{U}^r_{-i}$). While evaluating these Ψ values, PGS assumes that player $-i$ (resp. i) performs in s a player-action in which all ready units perform a unit-action given by a default script from $\mathcal{P}$. Player-action a_i and a_{-i} are initialized with the unit-actions provided by $\bar{\sigma}_i$ and $\bar{\sigma}_{-i}$ (lines 3 and 4 of Algorithm 1).

Algorithm 2 IMPROVE

Require: state s, ready units $\mathcal{U}_i^r = \{u_i^1, \cdots, u_i^{n_i}\}$ in s, set of scripts $\mathcal{P}$, action vector a_i for player i, action vector a_{-i} for player $-i$, evaluation function Ψ, integer I, and time limit t.
Ensure: action vector a_i for player i

```
1: for j ← 0 to I do
2:     if if time elapsed is larger than t then
3:         return a_i
4:     for k ← 1 to |U_i^r| do
5:         for each σ̄ ∈ P do
6:             a'_i ← a_i; a'_i[k] ← σ̄(s, u_i^k)
7:             if Ψ(T(s, a'_i, a_{-i})) > Ψ(T(s, a_i, a_{-i})) then
8:                 a_i ← a'_i
9: return a_i
```

The Improve Procedure Once a_i and a_{-i} have been initialized, PGS iterates through all units u_i^k in $\mathcal{U}_i^r$ and tries to greedily improve the move assigned to u_i^k in a_i, denoted by $a_i[k]$ (see Algorithm 2). PGS evaluates a_i while replacing $a_i[k]$ by each possible action for u_i^k, where the actions are defined by the scripts in $\mathcal{P}$. PGS keeps in a_i the action vector found during search with the largest Ψ-value. Procedure IMPROVE approximates a best response for a_{-i}. R determines how many times PGS alternates between approximating a best response for i's action and then $-i$'s action. The search procedure is capped by time limit t (line 2 of Algorithm 2).

PGS in Practice Churchill and Buro (2013) and Wang et al. (2016) used PGS with $R = 0$ in their experiments. In addition to using $R = 0$, Lelis (2017) and Moraes and Lelis (2018) removed parameter I and their PGS variant runs its IMPROVE procedure while the time elapsed is smaller than the limit t. In practice, by having $R = 0$, PGS is used to compute a best response to a fixed opponent, the one defined in the seeding process. As we show below, PGS tends to encounter weaker strategies if $R > 0$.

Non-Convergence Problem

The process of alternating between improving the actions of players i and $-i$, as described in Algorithms 1 and 2, might fail to retrieve the best action amongst those evaluated. Figure 1 shows a hypothetical game that highlights this problem, which we call the non-convergence problem. In this example player i and $-i$ can choose from actions a, b and c, and e and f, respectively. In a simultaneous move game, player $-i$ would not be able to distinguish the three states at the second level of tree (i.e., $-i$ would not know which action i will play). However, as was done in previous works (Kovarsky and Buro 2005; Churchill, Saffidine, and Buro 2012), we simplify the game and assume throughout this paper that one player acts after the other; in this example $-i$ acts after i. The squared nodes in the tree represent terminal states, with the numbers inside the squares representing player i's payoffs. Here, i is trying to maximize their payoff, while $-i$ is trying to minimize it.

Action c is the best action for player i as i is guaranteed a utility of 1, independently of player $-i$'s action. Next, consider the following possible run of PGS for the game shown in Figure 1. Let us suppose that in its seeding process PGS chooses action a for player i, hoping to reach the terminal state with utility of 2, and action e for player $-i$, hoping to reach the terminal state with utility of -2. In its improvement step for player i, PGS chooses action b, as b maximizes i's payoff given that $-i$ plays action e. After that, PGS's improvement for player $-i$ chooses action f, as f minimizes i's payoff given player i's action. Notice that PGS indefinitely alternates between actions a and b for player i and between actions e and f for player $-i$, thus failing to return the best action c. This example shows that, even if IMPROVE performed a systematic search in which all legal actions for both players were evaluated, PGS could still fail to return the best action—in the example action c is not returned by PGS even if it is evaluated in every call to IMPROVE for i.

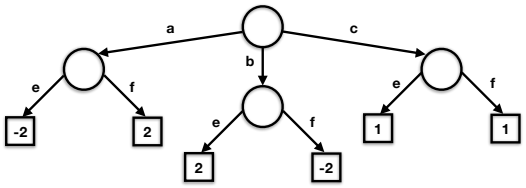

Figure 1: A hypothetical game where player i acts first by playing actions a, b, or c; player $-i$ acts second by playing actions e or f. Squared nodes are terminal states where the numbers represent the utility values for player i.

The non-convergence problem poses a serious limitation to the applicability of PGS. This is because, in practice, as we show below, PGS with $R > 0$ tends to be outperformed by PGS with $R = 0$. Thus, the practitioner has to define a priori an opponent strategy for which PGS will compute a best response (if $R = 0$, then a_{-i} is fixed throughout PGS execution, making PGS approximate a best response to a_{-i}). Wang et al. (2016), Lelis (2017), and Moraes and Lelis (2018) fixed $\bar{\sigma}_{-i}$ of PGS (see line 2 of Algorithm 1) to a strategy called NOKAV. However, NOKAV is specialized for combats and is unable to play an RTS match. It is unclear which strategy to use in other domains such as μRTS.

Another negative consequence of using $R = 0$ is that the player controlled by PGS might become highly exploitable. This is because the strategy derived by PGS considers that the opponent plays a pre-defined strategy, while in reality the opponent could be playing a different strategy.

An obvious solution to the non-convergence problem explained above is to run a minimax search to retrieve an optimal action. However, a minimax search might require one to visit a large number of states before finding an optimal solution, which is not feasible due to the games' real-time constraints. Next, we introduce NGS, a novel search algorithm that uses a procedure that is similar to PGS's greedy search to approximate the minimax value of the game.

Nested-Greedy Search (NGS)

Algorithm 3 Nested-Greedy Search (NGS)

Require: state s, ready units $\mathcal{U}_i^r = \{u_i^1, \cdots, u_i^{n_i}\}$ and $\mathcal{U}_{-i}^r = \{u_{-i}^1, \cdots, u_{-i}^{n_{-i}}\}$ in s, set of scripts $\mathcal{P}$, evaluation function Ψ, and time limit t.
Ensure: action a for player i's units.

```
1: σ̄_i ← choose a script from P
2: σ̄_{-i} ← choose a script from P
3: a_i ← {σ̄_i(u_i^1), ··· , σ̄_i(u_i^{n_i})}
4: a_{-i} ← {σ̄_{-i}(u_{-i}^1), ··· , σ̄_{-i}(u_{-i}^{n_{-i}})}
5: while time elapsed is not larger than t do
6:    for k ← 1 to |U_i^r| do
7:       for each σ̄ ∈ P do
8:          a'_i ← a_i; a'_i[k] ← σ̄(s, u_i^k)
9:          if GS(s, a_{-i}, a'_i, Ψ) > GS(s, a_{-i}, a_i, Ψ) then
10:            a_i ← a'_i
11:      if time elapsed is larger than t then
12:         return a_i
13: return a_i
```

Similarly to PGS, NGS uses a greedy search to decide which actions a_i will be evaluated during search. Each a_i considered by NGS's greedy procedure is evaluated by another greedy search that approximates the opponent's best response to a_i. This is in contrast with PGS, which evaluates each a_i as a best response to the opponent's current action a_{-i}. NGS returns the action a_i evaluated during search with highest estimated payoff. The name "nested greedy" comes from the fact that NGS uses a greedy search to evaluate each action a_i considered by the algorithm's main greedy search.

Algorithm 3 shows NGS's pseudocode. NGS receives as input the sets of ready units for state s, denoted $\mathcal{U}_i^r$ and $\mathcal{U}_{-i}^r$, a set of scripts $\mathcal{P}$, an evaluation function Ψ, and a time limit t. NGS returns an action vector a for player i to be executed in s. NGS also starts by setting seeds for both players (see lines 1–4), exactly as is done by PGS. Similarly to PGS, NGS evaluates a set of actions a_i as defined by the set of scripts $\mathcal{P}$ (lines 6–8). NGS evaluates each a_i according to the approximated best response of player $-i$ to a_i, as computed by a greedy search (GS), shown in Algorithm 4. GS iterates through all units u_{-i}^k in $\mathcal{U}_{-i}^r$ while greedily improving the action assignment to u_{-i}^k in a_{-i}, denoted by $a_{-i}[k]$ (see lines 2 and 3), while assuming i's action to be a_i. GS approximates the players' payoffs while $-i$ best responds to a_i. Note that i tries to maximize its payoff by changing the assignment of a_i only if that results in a larger value returned by GS (lines 9 and 10 in Algorithm 3), and player $-i$ tries to minimize i's payoff by changing a_{-i} only if that results in a reduction in i's payoff (lines 5 and 6 of Algorithm 4).

Non-Convergence Example Revisited

If NGS evaluates all actions for player i in the hypothetical game shown in Figure 1 and GS is able to correctly compute the best response for each a_i, then NGS will return action c for player i. This is because when evaluating action a, GS returns the value of -2, as $-i$ is able to best respond with e; GS returns -2 for b and 1 to c, which is returned by NGS.

Note that, in general, NGS is not guaranteed to find the best legal action amongst those considered by the set of scripts $\mathcal{P}$. This is because NGS uses a greedy search to decided which actions a_i will be evaluated during search, which may leave legal actions without being evaluated, and it uses another greedy search to approximate the best response of the opponent. However, in contrast with PGS, if the greedy search used to evaluate the opponent's best response is exact, NGS is guaranteed to return the best action for player i amongst the set of actions evaluated in search.

Algorithm 4 GREEDY SEARCH (GS)

Require: state s, ready units $\mathcal{U}_{-i}^r = \{u_{-i}^1, \cdots, u_{-i}^{n_{-i}}\}$ in s, set of scripts $\mathcal{P}$, action vector a_{-i} for player $-i$, action vector a_i for player i, and evaluation function Ψ.
Ensure: the best action value by player $-i$ in response a action a_i.

```
1: B ← ∞
2: for k ← 1 to |U_{-i}^r| do
3:    for each σ̄ ∈ P do
4:       a'_{-i} ← a_{-i}; a'_{-i}[k] ← σ̄(s, u_{-i}^k)
5:       if Ψ(T(s, a_i, a'_{-i})) < B then
6:          a_{-i} ← a'_{-i} ; B ← Ψ(T(s, a_i, a'_{-i}))
7: return B
```

Another source of error for NGS is its inability to evaluate a large number of actions due to its time complexity. The number of calls of Ψ grows linearly with the size of $\mathcal{P}$ and with the number of units for PGS. By contrast, the number of calls of Ψ grows quadratically with the size of $\mathcal{P}$ and with the number of units for NGS. Specifically, each iteration of the outer for loop of PGS (see Algorithm 2) performs $O(|\mathcal{U}_i^r| \times |\mathcal{P}|)$ calls of Ψ. By contrast, each iteration of the outer while loop of NGS (see Algorithm 3) performs $O(|\mathcal{U}_i^r| \times |\mathcal{U}_{-i}^r| \times |\mathcal{P}|^2)$ calls of Ψ. Due to the real-time constraints, in scenarios with a large set of scripts and/or with many units, PGS might be able to evaluate a much larger number of actions, which could outweigh NGS's advantage of approximating a best response to the player's action.

Finally, another source of error for both PGS and NGS is an imperfect function Ψ. An imperfect Ψ can make NGS's GS compute the wrong best response a_{-i}. Due to all these factors, we evaluate empirically in the domain of μRTS if NGS can be more effective than PGS's search procedure.

Empirical Evaluation

Our empirical evaluation of NGS is divided into two parts. In the first part we show the results of PGS with $I = 1$ and $R = 0$, PGS with $I = 1$ and $R = 1$ (PGS$_R$), and NGS. In the first part we do not limit the running time of the algorithms and allow PGS and PGS$_R$ complete their iterations as defined by the values of I and R. NGS is allowed to run a complete iteration of the outer while loop shown in Algorithm 3. The goal of this first experiment is to show that even if allowed "more search", likely due to the non-convergence problem, PGS$_R$ can be outperformed by PGS. We also intend to show NGS performance if not limited by running time constraints.

In the second part we test PGS, PGS$_R$, and NGS against state-of-the-art search methods for RTS games. Namely,

Map 8 × 8					Map 12 × 12					Map 16 × 16					Map 24 × 24				
	PGS	PGS_R	NGS	Avg.		PGS	PGS_R	NGS	Avg.		PGS	PGS_R	NGS	Avg.		PGS	PGS_R	NGS	Avg.
PGS	-	75.0	43.8	59.4	PGS	-	87.5	46.2	66.9	PGS	-	78.7	32.5	55.6	PGS	-	73.8	47.5	60.6
PGS_R	25.0	-	12.5	**18.8**	PGS_R	12.5	-	3.8	**8.1**	PGS_R	21.3	-	12.5	**16.9**	PGS_R	26.3	-	3	**18.8**
NGS	56.2	87.5	-	71.9	NGS	53.8	96.2	-	75.0	NGS	67.5	87.5	-	77.5	NGS	52.5	88.8	-	70.6

Table 1: Results of PGS, PGS_R, and NGS without running time constraints. Entries in bold indicate pathological cases in which PGS_R performs on average worse than PGS (see column "Avg.").

	NGS	STT	NAV	SCV	PGS	AHT	PS	PGS_R
Total	866	735	560	546	493	475	339	331

Table 2: Total number of victories of each approach; maximum possible number of victories is 1,120.

we test the following algorithms: Adversarial Hierarchical Task Network (AHT) (Ontañón and Buro 2015), an algorithm that uses Monte Carlo tree search and HTN planning; NaïveMCTS (Ontañón 2017) (henceforth referred as NAV), an algorithm based on combinatorial multi-armed bandit algorithm; the MCTS version of Puppet Search (PS) (Barriga, Stanescu, and Buro 2017b) and Strategy Tactics (STT) (Barriga, Stanescu, and Buro 2017a). In these experiments all algorithms are allowed 100 milliseconds of planning time.

All our experiments are run on μRTS, a minimalist RTS game developed for adversarial real-time planning research (Ontañón 2013). μRTS allows one to test algorithms without having to deal with engineering problems normally 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]

We use maps of size $x \times x$ with $x \in \{8, 12, 16, 24\}$. Every match is limited by a number of game cycles and the match is considered a draw once the limit is reached. We present the percentage of matches won by each algorithm, the matches finishing in draws are counted as 0.5 for both sides. The maximum number of game cycles is map dependent. We use the limits defined by Barriga et al. (2017b): 3000, 4000, 4000, 5000 game cycles for maps of size 8, 12, 16, and 24. Each tested algorithm plays against every other algorithm 40 times in each map tested. To ensure fairness, the players switch their starting location on the map an even number of times. For example, if method 1 starts in location X with method 2 starting in location Y for 20 matches; we switch the starting positions for the remaining 20 matches.

The Ψ function we use for PGS, PGS_R, and NGS is a random play-out of 100 game cycles of length (approximately 10 actions for each player in the game). The random play-out evaluates state s by simulating the game forward from s for 100 game cycles with both players choosing random actions, until reaching a state s'. Then, we have that $\Psi(s) = \Phi(s')$, where Φ is μRTS's evaluation function introduced by Ontañón (Ontañón 2017). Φ computes a score for each player—$score(i)$ and $score(-i)$—by summing up the cost in resources required to train each unit controlled by the player weighted by the square root of the unit's hit points. The Φ value of a state is given by player i's score minus player $-i$'s score. Φ is then normalized to a value in $[-1, 1]$ through the following formula $\frac{2*score(i)}{score(-i)+score(i)} - 1$.

The set of scripts we use with PGS, PGS_R, and NGS is composed by Worker rush (WR) (Stanescu et al. 2016), NOKAV, and Kiter (Churchill and Buro 2013). WR trains a large number of workers which are immediately sent to attack the enemy; NOKAV 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 back in combat. Although traditionally used with units that can attack from far, Kiter may still give units that have to be near the enemy to be able to attack a strategic advantage by allowing them to move away from the enemy. The default script we use in the seeding process of PGS, PGS_R, and NGS is WR. All experiments were run on 2.1 GHz CPUs.

First Experiment: No Time Limit

Table 1 presents the results for PGS, PGS_R, and NGS. Each entry of the table shows the percentage of wins of the row approach against the column approach (out of 40 matches). We highlight in bold the pathological results, i.e., the cases in which PGS_R or NGS win fewer matches than PGS (see column "Avg.", which shows the average results). We call it pathological because PGS_R and NGS are expected to defeat PGS for being granted more "search time" than PGS. Recall that PGS_R performs one improve for the player, one for the opponent, and finally, a last improvement for the player. By contrast, PGS performs a single improvement for the player. PGS_R presented pathological results in all maps tested. For example, PGS wins on average 60.6% of the matches played in the 24×24 map, while PGS_R wins only 18.8%. Overall, NGS outperforms both PGS and PGS_R. For example, NGS wins on average 77.5% of the matches played in the 16×16 map, while PGS wins 55.6%.

Second Experiment: Against State-of-the-Art

Table 2 presents the number of matches won by each approach tested in all 4 maps; matches finishing in draws are not included in these results. The maximum possible number of victories is 1,120. Overall, NGS wins more matches than any approach tested, suggesting that NGS's search scheme is able to find good actions by accounting for the opponent's possible response. PGS also performs well, being competitive with NAV and SCV and outperforming AHT, PS, and

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

Map 8 × 8

	PS	AHT	STT	NAV	SCV	PGS_R	PGS	NGS	Avg.
PS	-	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.7
AHT	100.0	-	25.0	12.5	2.5	57.5	33.8	25.0	36.6
STT	95.0	75.0	-	46.3	21.3	92.5	91.3	26.3	63.9
NAV	100.0	87.5	53.8	-	35.0	97.5	96.3	20.0	70.0
SCV	100.0	97.5	78.8	65.0	-	92.5	68.8	12.5	73.6
PGS_R	100.0	42.5	7.5	2.5	7.5	-	63.8	10.0	33.4
PGS	100.0	66.3	8.8	3.8	31.3	36.3	-	25.0	38.8
NGS	100.0	75.0	73.8	80.0	87.5	90.0	75.0	-	83.0

Map 12 × 12

	PS	AHT	STT	NAV	SCV	PGS_R	PGS	NGS	Avg.
PS	-	0.0	2.5	70.0	40.0	7.5	0.0	0.0	17.1
AHT	100.0	-	23.8	10.0	81.3	78.8	46.3	50.0	55.7
STT	97.5	76.3	-	45.0	100.0	100.0	100.0	53.8	81.8
NAV	30.0	90.0	55.0	-	57.5	82.5	72.5	60.0	63.9
SCV	60.0	18.8	0.0	42.5	-	5.0	0.0	0.0	18.0
PGS_R	92.5	21.3	0.0	17.5	95.0	-	21.3	0.0	35.4
PGS	100.0	53.8	0.0	27.5	100.0	78.8	-	17.5	53.9
NGS	100.0	50.0	46.3	40.0	100.0	100.0	82.5	-	74.1

Map 16 × 16

	PS	AHT	STT	NAV	SCV	PGS_R	PGS	NGS	Avg.
PS	-	0.0	65.0	100.0	0.0	12.5	0.0	0.0	25.4
AHT	100.0	-	45.0	0.0	60.0	75.0	70.0	12.5	51.8
STT	35.0	55.0	-	76.3	48.8	65.0	65.0	31.3	53.8
NAV	0.0	100.0	23.8	-	25.0	82.5	53.8	17.5	43.2
SCV	100.0	40.0	51.3	75.0	-	77.5	62.5	0.0	58.0
PGS_R	87.5	25.0	35.0	17.5	22.5	-	17.5	0.0	29.3
PGS	100.0	30.0	35.0	46.3	37.5	82.5	-	5.0	48.0
NGS	100.0	87.5	68.8	82.5	100.0	100.0	95.0	-	90.5

Map 24 × 24

	PS	AHT	STT	NAV	SCV	PGS_R	PGS	NGS	Avg.
PS	-	27.5	52.5	100.0	100.0	85.0	87.5	92.5	77.9
AHT	72.5	-	0.0	5.0	16.3	62.5	50.0	0.0	29.5
STT	47.5	100.0	-	81.3	67.5	86.3	77.5	45.0	72.1
NAV	0.0	95.0	18.8	-	2.5	52.5	40.0	30.0	34.1
SCV	0.0	83.8	32.5	97.5	-	55.0	43.8	50.0	51.8
PGS_R	15.0	37.5	13.8	47.5	45.0	-	20.0	0.0	25.5
PGS	12.5	50.0	22.5	60.0	56.3	80.0	-	8.8	41.4
NGS	7.5	100.0	55.0	70.0	50.0	100.0	91.3	-	67.7

Table 3: Percentage winning rate of all methods tested; draws are counted as 0.5 to both sides before the percentage is computed.

PGS_R. PGS is only outperformed by NGS and STT. The difference between PGS and PGS_R helps explaining why researchers use PGS with $R = 0$ in their experiments.

Table 3 shows the results of our experiments for each map. Each cell shows the percentage of wins of the row method against the column method; the numbers are truncated to one decimal place. We highlight the background of cells showing the percentage of wins of PGS, PGS_R, or NGS if that was greater or equal to 50%. We also highlight the cell with the highest average percentage of wins (column "Avg.").

By comparing the lines of PGS and PGS_R one can see that the latter is never better than the former, but often substantially worse. For example, while PGS wins 53.8% of the matches played in a 12 × 12 map against NAV, PGS_R wins only 21.3% of the matches against the same opponent. Overall, NGS not only performs better than PGS and PGS_R, but it also performs better than most of the state-of-the-art approaches tested. For example, NGS only does not directly outperform all approaches in the map of size 12 × 12—this can be observed by the highlighted cells across NGS's rows.

One notices a decrease in the performance of NGS against some of the methods as the size of the map increases. For example, against SCV, NGS wins 80%, 100%, and 100% of the matches played in maps of size 8, 12, and 16, respectively. However, NGS wins only 50% of matches played in a map of size 24 against the same opponent. This happens likely because NGS's time complexity grows quadratically with the number of units. Thus, other approaches might be preferred in matches played in larger maps. In addition to RTS games played in small to medium-sized maps, NGS might be a valuable option for games such Prismata (Churchill and Buro 2015), which also impose time constraints, but the constraints are on the order of seconds instead of milliseconds.

Another interesting observation from the positive results shown in Tables 2 and 3 is the fact that PGS and NGS can be used to effectively play full RTS games. PGS was developed to predict the results of combat scenarios that arise in RTS matches, and not to play RTS matches. Our results suggest that researchers should consider PGS and NGS, as well as all other algorithms based on the same ideas such as POE (Wang et al. 2016) and SSS (Lelis 2017), as competing schemes for search-based systems for RTS games.

Conclusions

In this paper we have presented a problem with PGS's search scheme. Namely, even under the strong assumption that PGS is able to evaluate all actions available to the player at a given state, the algorithm might fail to return the best action. We showed empirically in μRTS matches that this problem might cause PGS to present pathological results, i.e., PGS performs worse if allowed more planning time. We then introduced NGS, a search algorithm to overcome PGS's problem. Empirical results in μRTS matches played in small to medium-sized maps showed that NGS is able to outperform not only PGS but all state-of-the-art algorithms tested. A secondary contribution of our work was to show that, despite PGS being developed to control units in RTS combats, PGS and NGS can be used to effectively play entire RTS matches. Thus, other researchers should also consider PGS and the algorithms that followed PGS as competing schemes for search-based systems for RTS games.

Acknowledgements

This research was supported by FAPEMIG, CNPq and CAPES, Brazil. The authors thank the great suggestions provided by the anonymous reviewers.

References

Balla, R.-K., and Fern, A. 2009. Uct for tactical assault planning in real-time strategy games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 40–45.

Barriga, N. A.; Stanescu, M.; and Buro, M. 2017a. Combining strategic learning and tactical search in real-time strategy games. *Thirteenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

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*.

Chung, M.; Buro, M.; and Schaeffer, J. 2005. Monte Carlo planning in RTS games. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*.

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.

Churchill, D., and Buro, M. 2015. Hierarchical portfolio search: Prismata's robust AI architecture for games with large search spaces. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 16–22.

Churchill, D.; Saffidine, A.; and Buro, M. 2012. Fast heuristic search for RTS game combat scenarios. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

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

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.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proceedings of the European Conference on Machine Learning*, 282–293. Springer-Verlag.

Kovarsky, A., and Buro, M. 2005. Heuristic search applied to abstract combat games. In *Advances in Artificial Intelligence: Conference of the Canadian Society for Computational Studies of Intelligence*, 66–78. Springer.

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.

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

Moraes, R. O.; no, J. R. H. M.; Lelis, L. H. S.; and Nascimento, M. A. 2018. Action abstractions for combinatorial multi-armed bandit tree search. *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

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. 2018. The first microrts artificial intelligence competition. *AI Magazine* 39(1).

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.

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

Sailer, F.; Buro, M.; and Lanctot, M. 2007. Adversarial planning through strategy simulation. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 80–87.

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

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

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.