# Choosing Well Your Opponents: How to Guide the Synthesis of Programmatic Strategies

**Rubens O. Moraes**[1,2] , **David S. Aleixo**[1] , **Lucas N. Ferreira**[2] and **Levi H. S. Lelis**[2]
[1] Departamento de Informática, Universidade Federal de Viçosa, Brazil
[2]Department of Computing Science, University of Alberta, Canada
Alberta Machine Intelligence Institute (Amii)
rubens.moraes@ufv.br, david.aleixo@ufv.br, lnferrei@ualberta.ca,levi.lelis@ualberta.ca

## Abstract

This paper introduces Local Learner (2L), an algorithm for providing a set of reference strategies to guide the search for programmatic strategies in two-player zero-sum games. Previous learning algorithms, such as Iterated Best Response (IBR), Fictitious Play (FP), and Double-Oracle (DO), can be computationally expensive or miss important information for guiding search algorithms. 2L actively selects a set of reference strategies to improve the search signal. We empirically demonstrate the advantages of our approach while guiding a local search algorithm for synthesizing strategies in three games, including MicroRTS, a challenging real-time strategy game. Results show that 2L learns reference strategies that provide a stronger search signal than IBR, FP, and DO. We also simulate a tournament of MicroRTS, where a synthesizer using 2L outperformed the winners of the two latest MicroRTS competitions, which were programmatic strategies written by human programmers.

## 1 Introduction

Programmatic strategies encode game strategies in human-understandable programs. Such programmatic encoding allows domain experts to interpret and modify computer-generated strategies, which can be valuable depending on the application domain (e.g., the games industry). Previous works have used Iterated-Best Response (IBR) [Lanctot *et al.*, 2017] as the learning algorithm for synthesizing programmatic strategies [Mariño *et al.*, 2021]. Given a game, IBR starts with an arbitrary strategy for playing the game and it approximates a best response to it; in the next iteration, it approximates a best response to the best response. This process is repeated a number of iterations and the programmatic strategy synthesized in the last iteration is returned.

The computation of the best responses in the IBR loop is performed by searching in the programmatic space defined by a domain-specific language. Given a target strategy, the algorithm searches for a program encoding a best response to it. Previous work used local search algorithms for searching in the programmatic space [Mariño *et al.*, 2021; Medeiros *et al.*, 2022; Aleixo and Lelis, 2023]. The target strategy IBR provides serves as a guiding function. In the context of local search, when considering the neighbors of a candidate solution, local search algorithms prefer to accept a program that achieves a higher utility value against the target strategy. Since IBR considers a single strategy as target, the search signal is often weak. This is because the neighbors of a candidate solution that performs poorly against the target strategy are also likely to perform poorly against it—small changes to a losing program will also generate a losing program. Moreover, IBR can loop around the strategy space in games with dynamics similar to Rock, Paper, and Scissors, without making progress toward strong solutions.

In this paper, we adapt Fictitious Play (FP) [Brown, 1951] and Double Oracle (DO) [McMahan *et al.*, 2003] to the context of programmatic strategies. FP and DO have been used in the context of neural strategies to overcome some of the weaknesses of IBR [Lanctot *et al.*, 2017]. Despite providing a better search signal than IBR, we show that FP and DO can still fail to provide relevant information for the search. We then introduce a novel learning algorithm, Local Learner (2L), that is designed specifically for guiding local search algorithms in the synthesis of programmatic strategies. 2L uses information gathered while computing best responses to decide the set of target strategies to be used in future iterations of the algorithm as a means of optimizing the search signal.

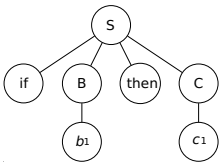
We evaluate 2L on three two-player zero-sum games: MicroRTS [Ontañón *et al.*, 2018], Poachers & Rangers, and Climbing Monkeys. Results show that 2L synthesized strategies that are never worse and often far superior to strategies synthesized with IBR, FP, and DO in all three domains. We also performed a simulated competition of MicroRTS with strategies synthesized with 2L IBR, FP, DO, as well as the programmatic strategies that won the last two MicroRTS competitions, which were written by programmers. 2L obtained the highest average winning rate in our tournament.

## 2 Problem Definition

We consider the synthesis of programmatic strategies assuming zero-sum two-player games $\mathcal{G} = (\mathcal{P}, \mathcal{S}, s_{\text{init}}, \mathcal{A}, \mathcal{T}, \mathcal{U})$. Let $\mathcal{P} = \{i, -i\}$ be the pair of players; $\mathcal{S}$ be the set of states, with $s_{\text{init}}$ in $\mathcal{S}$ being the initial state. Each player $i$ can perform an action from a legal set of actions $\mathcal{A}_i(s)$ in $\mathcal{A}$ for a given state $s$. The action of each player is given by a strategy, which is a function $\sigma_i$ that receives a state $s$ in $\mathcal{S}$ and returns

an action in $\mathcal{A}_i$ for $s$. A transition function $\mathcal{T}$ receives a state and an action for each player and deterministically returns the next state of the game, which could be a terminal state, where the utility of each player is determined. The utility function $\mathcal{U}$ returns the value of the game at a given state (terminal or not). For $s$, the value of the game is denoted by $\mathcal{U}(s, \sigma_i, \sigma_{-i})$ when player $i$ follows the strategy $\sigma_i$ and player $-i$, $\sigma_{-i}$. Considering that the game $\mathcal{G}$ is zero-sum, the utility function for $-i$ is $-\mathcal{U}(s, \sigma_i, \sigma_{-i})$. In this paper, we encode strategies for $\mathcal{G}$ as programs written in a domain-specific language (DSL).

A DSL can be defined as a context-free grammar $(M, \Omega, R, S)$, where $M$, $\Omega$, $R$, and $S$ are the sets of non-terminals, terminals, relations defining the production rules of the grammar, and the grammar's initial symbol, respectively. Figure 1 (right) shows an example of a DSL, where $M = \{S, C, B\}$, $\Omega = \{c_1, c_2, b_1, b_2 \text{ if, then}\}$, $R$ are the production rules (e.g., $C \rightarrow c_1$), and $S$ is the initial symbol.




$S \rightarrow C \mid \text{if}(B) \text{ then } C$
$C \rightarrow c_1 \mid c_2$
$B \rightarrow b_1 \mid b_2$

Figure 1: AST for "if $b_1$ then $c_1$" (left) and DSL in which the program is written (right).

The DSL in Figure 1 allows programs with a single command (e.g., $c_1$ or $c_2$) and programs with branching. We represent programs as abstract syntax trees (AST), where the root of the tree is $S$, the internal nodes are non-terminals, and the leaf nodes are terminals. Figure 1 (left) shows an example of an AST. We use a DSL $D$ to define the space of programs $[\![D]\!]$, where each program $p \in [\![D]\!]$ is a game strategy.

One solves the problem of synthesizing programmatic strategies by solving the following equation

$$\max_{\sigma_i \in [\![D]\!]} \min_{\sigma_{-i} \in [\![D]\!]} \mathcal{U}(s_{\text{init}}, \sigma_i, \sigma_{-i}) \,. \tag{1}$$

The strategies $\sigma_i$ and $\sigma_{-i}$ in $[\![D]\!]$ able to solve Equation 1 define a Nash equilibrium profile in the programmatic space. We consider a programmatic variant of PSRO [Lanctot *et al.*, 2017] to approximate a solution to Equation 1.

## 3 Programmatic PSRO (PPSRO)

Let $\lambda$ be a normal-form game defined by $(\Sigma, \mathcal{P}, U_\Sigma)$, where $\Sigma = \{\Sigma_i, \Sigma_{-i}\}$ represents a set of strategies for each player in $\mathcal{P} = \{i, -i\}$, and $U_\Sigma$ is the utility payoff table between each pair of strategies in $\Sigma$. A mixed strategy $\sigma$ is a probability distribution over strategies $\Sigma_i$ and $\Sigma_{-i}$ for players $i$ and $-i$, respectively. An empirical game of a normal-form game contains only a subset of the strategies of the original game.

Policy-Space Response Oracles (PSRO) is a framework for learning strategies that "grow" an empirical game [Lanctot *et al.*, 2017]. In PSRO, the empirical game starts with a single strategy in $\Sigma_i$ and $\Sigma_{-i}$ and it grows these sets by including a new strategy for each player in each iteration of the algorithm. Let a mixed strategy over the sets $\Sigma_i$ and $\Sigma_{-i}$ of the empirical game be called a meta-strategy. PSRO grows $\Sigma_i$ and $\Sigma_{-i}$ by adding best responses to meta-strategies. Once a best response is added to a set, a new meta-strategy is computed and the process is repeated. That is, given a meta-strategy $\sigma_{-i}$ (resp. $\sigma_i$), for player $-i$ (resp. $i$), the best response to $\sigma_{-i}$ (resp. $\sigma_i$) is added to $\Sigma_i$ (resp. $\Sigma_{-i}$).

**Algorithm 1** Programmatic PSRO

**Require:** Game $\mathcal{G}$, DSL $D$, learning algorithm $\Psi$.
**Ensure:** Strategy $\sigma_i$ for player $i$.
1: Initialize $\Sigma_i$ and $\Sigma_{-i}$ with one strategy each.
2: Compute utilities $U_\Sigma$ for $(\sigma_i, \sigma_{-i})$
3: **while** have not exhausted budget **do**
4:  **for** player $i$ in $\mathcal{P}$ **do**
5:  Compute a meta-strategy $\sigma_{-i}$ with $\Psi(\Sigma, U_\Sigma)$
6:  $\sigma_i' \leftarrow$ search($\sigma_i[-1]$, $\sigma_{-i}$)
7:  $\Sigma_i \leftarrow \Sigma_i \cup \{\sigma_i'\}$
8:  Compute entries in $U_\Sigma$ from $\Sigma$
9: **return** Last meta-strategy $\sigma_i$

PSRO generalizes algorithms such as IBR, FP, and DO depending on how the meta-strategies are computed. Let $\sigma_k = (p_1, p_2, \cdots, p_n)$ be a meta-strategy for player $k$ ($k$ can be either $i$ or $-i$). Here, $p_j$ in $\sigma_k$ represents the probability in which $\sigma_k$ plays the $j$-th strategy added to the empirical game for player $k$. PSRO generalizes IBR if the meta-strategies are of the form $(0.0, 0.0, \cdots, 1.0)$, i.e., the only strategy in the support of the meta-strategy is the last strategy added to the empirical game. If the meta-strategy $\sigma_{-i}$ with $n$ strategies is of the form $(1/n, 1/n, \cdots, 1/n)$, i.e., all the previous strategies added to the game are played with equal probability, then PSRO generalizes FP. PSRO also generalizes DO [McMahan *et al.*, 2003] when the meta-strategy is computed by solving the empirical game. We use a variant of PSRO, which we call Programmatic PSRO (PPSRO), to approximate a solution to Equation 1. PPSRO is shown Algorithm 1.

PPSRO starts by initializing the set of strategies, $\Sigma_i$ and $\Sigma_{-i}$, with two arbitrary strategies (line 1). PPSRO runs a number of iterations according to a given computational budget (e.g., the number of games played). In each iteration, PPSRO invokes a learning algorithm $\Psi$ (e.g., IBR) that receives the current empirical game and returns a meta-strategy $\sigma_{-i}$ (line 5). Then, it searches in the programmatic space of strategies for a best response $\sigma_i'$ to $\sigma_{-i}$. We consider local search algorithms for computing $\sigma_i'$. The search algorithm, described in Section 4, initializes its computation with the last strategy added to the empirical game for $i$, which is denoted as $\sigma_i[-1]$ (line 6). The best response $\sigma_i'$ is then added to $\Sigma_i$. At the end, PPSRO returns the last meta-strategy $\sigma_i$ as an approximate solution for player $i$ to Equation 1 (line 9).

The choice of meta-strategies across iterations of PPSRO determines how quickly it is able to approximate a Nash equilibrium profile for the game. Previous work investigated different approaches for defining meta-strategies in the context of PSRO and neural policies [Lanctot *et al.*, 2017; Anthony *et al.*, 2020; Muller *et al.*, 2020]. However, searching in programmatic space is different than searching in neural space, since the former does not have a gradient signal to guide the search. As we show in our experiments, meta-

strategies used with PSRO might not work well with PPSRO.

## 4 Hill Climbing for Synthesis of Strategies

Hill Climbing (HC) is a local search algorithm that starts with an arbitrary candidate solution to a combinatorial search problem and attempts to improve it with greedy changes to the candidate. We use HC to approximate best responses to strategies $\sigma_{-i}$ in the PPSRO main loop (line 6 of Algorithm 1). HC receives the last strategy added to the empirical game for player $i$, which is denoted as $\sigma_i[-1]$, and $\sigma_{-i}$. The algorithm returns an approximate best response to $\sigma_{-i}$. This is achieved by searching in the programmatic space defined by the DSL. The starting candidate solution $\sigma_0$ of the search is $\sigma_i[-1]$. HC attempts to approximate a best response to $\sigma_{-i}$ by evaluating neighbor strategies of $\sigma_0$. We update the current candidate solution $\sigma_0$ to a neighbor $\sigma_i'$ if the value $\mathcal{U}(s_{\text{init}}, \sigma_i', \sigma_{-i})$ is greater than $\mathcal{U}(s_{\text{init}}, \sigma_0, \sigma_{-i})$. Otherwise, HC generates and evaluates a new neighbor solution $\sigma_i'$ of $\sigma_0$. This process is repeated until we have exhausted the search budget. HC returns the strategy encountered in search with highest $\mathcal{U}$-value as its approximated best response to $\sigma_{-i}$.

Neighbor solutions are produced by applying a "mutation" in $\sigma_0$'s AST. A mutation is performed by uniformly sampling a non-terminal symbol $S$ in the AST, and replacing the subtree rooted at $S$ with a new subtree. The new subtree is generated by replacing $S$ with the right-hand side of a production rule for $S$ that is selected uniformly at random. The mutation process repeatedly replaces a non-terminal leaf node in the generated program with the right-hand side of a random production rule of the DSL until the program's AST contains only terminal symbols as leaves.

HC is initialized with a random program only in the first iteration of PPSRO; HC is initialized with the programmatic best response computed in PPSRO's previous iteration otherwise ($\sigma_i[-1]$ in line 6 of Algorithm 1).

## 5 Shortcomings of Existing Approaches

The effectiveness of the search algorithm, e.g., HC, for computing a best response, depends on the computational cost of $\sigma_{-i}$ and on the information $\sigma_{-i}$ encodes, as we explain next. The meta-strategy $\sigma_{-i}$ determines how fast we can approximate a Nash equilibrium profile for the game. This is because the utility function $\mathcal{U}(s_{\text{init}}, \sigma_i, \sigma_{-i})$ provides the search signal for the synthesis of a best response $\sigma_i$ to $\sigma_{-i}$ in the $[\![D]\!]$ space. For example, if the meta-strategy $\sigma_{-i}$ with $n$ strategies is of the form $(1/n, 1/n, \cdots, 1/n)$, i.e., all the previous strategies synthesized in the previous iterations are in $\sigma_{-i}$'s support, then $\sigma_{-i}$ is able to provide a richer guiding signal than IBR's meta-strategy, which accounts only for a single strategy. Note that PSRO (and PPSRO) with meta-strategies that accounts for all strategies with equal probability is equivalent to FP [Lanctot *et al.*, 2017]. Although FP provides a richer search signal, it incurs a higher computational cost as the guiding function $\mathcal{U}(s_{\text{init}}, \sigma_i, \sigma_{-i})$ requires one to evaluate all strategies in the support of the meta-strategy. Example 1 illustrates IBR's lack of information for guiding search in the game of Poachers and Rangers (P&R).

P&R is a simultaneous-move two-player zero-sum game without ties where *rangers* need to protect the gates of a national park to avoid *poachers* getting inside. In the game, poachers need to attack at least one unprotected gate to enter the park, and rangers succeed if they protect all gates attacked by poachers. Rangers receive the utility of 1 if they protect all attacked gates and -1 otherwise The game has a trivial Ranger's dominant strategy, where they protect all the gates. Despite having a trivial solution, the game is particularly hard as a program synthesis task. This difficulty is inherent to the size of the programmatic solution required to solve this game. If the number of gates is arbitrarily large, current synthesizers might struggle to synthesize such long programs. For example, for a game with $n$ gates, the optimal programmatic strategy is any permutation of the instructions in the following program: `defend`[1], `defend`[2], $\cdots$, `defend`[n], which we also denote as `defend`[1, 2, $\cdots$, n] for conciseness.

**Example 1.** *Let us consider a P&R instance with 2 gates. In the first iteration, IBR generates an arbitrary strategy for Rangers:* `defend[2]`. *In the next iteration, it computes a best response to* `defend[2]`: `attack[1]`. *Next, IBR computes a best response to the Poachers strategy,* `attack[1]`, *so it produces the strategy* `defend[1]`. *Then, IBR computes a best response to* `defend[1]`, *thus generating* `attack[2]` *for Poachers. In the next iteration, IBR computes* `defend[2]` *as a best response to* `attack[2]`. *Note that* `defend[2]` *is the strategy in which IBR started the learning procedure—IBR just looped back to the beginning of the process. Since IBR uses only the last synthesized strategy, it can loop over suboptimal strategies which could delay the convergence to the optimal strategy* `defend[1, 2]`.

*By contrast, in FP one considers all previous strategies synthesized in the learning process. Once the empirical game has the strategies* `attack[1]` *and* `attack[2]`, *the search algorithm is guided to synthesize the optimal* `defend[1, 2]`.

DO may strike a balance between computational cost and search guidance, i.e., it includes fewer strategies than FP, but more than IBR in the support of the meta-strategy. With DO, only the strategies in the empirical game that are deemed important, i.e., that are in the support of a Nash equilibrium strategy, will be considered in search. However, DO might still miss important information for guiding local search algorithms in the context of PPSRO, as we show in Example 2.

**Example 2.** *Let us consider a P&R instance with 5 gates. In the first iteration, DO generates two arbitrary strategies:* `defend[2]` *and* `attack[1]` *for Rangers and Poachers, respectively. Let us assume that PPSRO instantiated as DO generates the empirical game shown in Table 1 after a few iterations. In the following iteration, PPSRO adds a strategy for Rangers to the empirical game. This is achieved by solving the empirical game shown in Table 1 to generate a meta-strategy $\sigma_{-i}$ for Poachers and then approximating a best response $\sigma_i$ to $\sigma_{-i}$. The last row of Table 1 shows the strategy for $-i$ in the Nash equilibrium profile for the empirical game, which is used as the meta-strategy $\sigma_{-i}$. Any strategy $\sigma_i$ for Rangers that defends at least gates* 1, 2, *and* 5 *is a best response to $\sigma_{-i}$ since the support of $\sigma_{-i}$ only accounts for*

| Rangers | Poachers | | |
|---|---|---|---|
| | attack[1] | attack[1,2,3] | attack[1,2,5] |
| defend[2] | -1 | -1 | -1 |
| defend[1,2,4] | 1 | -1 | -1 |
| defend[1,2,3] | 1 | 1 | -1 |
| $\sigma_{-i}$ | 0.0 | 0.0 | 1.0 |

Table 1: The empirical game created by DO after a few iterations of the P&R. This matrix is calculated by evaluating row strategies against column strategies. A score of 1 means that the row strategy beats the column strategy, and -1, the row loses to the column.

*attack[1, 2, 5]. The best response $\sigma_i$ does not need to defend gate 3, despite it being part of the empirical game for Poachers (in strategy attack[1, 2, 3]). If both attack[1, 2, 3] and attack[1, 2, 5] were in the support of $\sigma_{-i}$, PPSRO would be forced to synthesize a strategy that defends gates 1, 2, 3, and 5. However, DO does not include attack[1, 2, 3] in the support of $\sigma_{-i}$, so PPSRO is only forced to synthesize a strategy that defends gates 1, 2, and 5, which could delay the convergence of the algorithm for missing gate 3.*

To address these limitations described for IBR, FP, and DO, we propose a new algorithm able to better guide the synthesis of programmatic strategies in the context of PPSRO.

## 6 Local Learner (2L)

We propose a new instance of PPSRO called Local Learner (2L), which can overcome the limitations of IBR, FP, and DO presented in the previous section. 2L defines meta-strategies that are "in between" those IBR and FP define in terms of the number of strategies in the meta-strategy's support. 2L can use more strategies than IBR to provide a better signal to the search algorithm, but it also attempts to use fewer strategies than FP to reduce the computational cost of the evaluation. The following P&R example illustrates how 2L works.

**Example 3.** *Let us consider a P&R instance with $n > 2$ gates. We initialize with an arbitrary strategy (attack[2]) for Poachers and compute a best response to it: defend[2]. Next iteration, we compute a best response to defend[2]: attack[1]. Next, 2L returns a meta-strategy $\sigma_{-i}$ for Poachers so we can compute a best response to it and add to the empirical game a new strategy for Rangers. Similarly to what FP would do, in this case, 2L returns a meta-strategy for Poachers that considers all strategies currently in the empirical game (attack[2] and attack[1]): $\sigma_{-i} = (0.5, 0.5)$. Let us suppose that the search returns the best response defend[1, 2] to $\sigma_{-i}$, which is added to the empirical game. 2L then returns a meta-strategy $\sigma_i = (0.5, 0.5)$ for Rangers that also considers all strategies currently in the empirical game (defend[2] and defend[1, 2]). While computing a best response to $\sigma_i$, 2L learns that the strategy defend[2] is redundant and can be dropped from the support of $\sigma_i$ in future iterations. Before finding a best response $\sigma_i$ (e.g., attack[3]), let us assume that the search evaluates strategies attack[1] and attack[2]. Note that defend[2] is a best response to only attack[2], while defend[1, 2] is a best response to both of them. Given the strategies evaluated in search and that defend[1, 2] is in the support of the meta-strategy, defend[2] does not add new information to the search and can thus be dropped.*

2L initially assumes that all strategies inserted in the empirical game is helpful to guide the search so it adds them to the support of its meta-strategy $\sigma_{-i}$. While computing a best response to $\sigma_{-i}$, it collects data about each strategy in $\sigma_{-i}$ and it removes from its support all "redundant strategies".

### 6.1 Formal Description

Let $\Sigma_k = \{\sigma_{1,k}, \cdots, \sigma_{n,k}\}$ be the set of strategies for player $k$ in the empirical game in an execution of PPSRO, where $k$ is either $i$ or $-i$ and $\sigma_{j,k}$ is the $j$-th strategy added for $k$ in the empirical game. Let $\sigma_k = (p_1, \cdots, p_n)$ be a meta-strategy over $\Sigma_k$ where $p_j$ in $\sigma_k$ indicates the probability in which $\sigma_k$ plays the $j$-th strategy in $\Sigma_k$. We denote $p_j$ in $\sigma_k$ as $\sigma_k[j]$. Let $\Sigma_{\sigma_k}$ be the subset of strategies in the support of $\sigma_k$, i.e., the strategies whose $p_j$-value is greater than zero in $\sigma_k$.

While computing a best response to a meta-strategy $\sigma_k$, 2L employs a search algorithm that evaluates a number of strategies as potential best responses to $\sigma_k$. Let $S$ be the set of strategies evaluated in search that are best responded by at least one strategy in $\Sigma_{\sigma_k}$. We call *helpful strategies*, denoted $\Sigma^h_{\sigma_k}$, the smallest subset of $\Sigma_{\sigma_k}$ that contains at least one best response to any strategy in $S$. We call *redundant strategies* the set $\Sigma_{\sigma_k}$ minus the helpful strategies $\Sigma^h_{\sigma_k}$.

**Example 4.** *In Example 3, when computing a best response to $\sigma_i = (0.5, 0.5)$ with $\Sigma_{\sigma_i} = \{$defend[2], defend[1, 2]$\}$ we have that $S = \{$attack[1], attack[2]$\}$ and $\Sigma^h_{\sigma_i} = \{$defend[1, 2]$\}$. 2L is then able to remove the redundant set $\{$defend[2]$\}$ from $\Sigma_{\sigma_i}$ for future iterations of the algorithm.*

In practice, we are unable to compute the smallest set $\Sigma^h_{\sigma_k}$ possible due to two reasons. First, the search might not encounter the strategies needed to prove a strategy helpful. In Example 3, if the synthesis algorithm encounters attack[1] but it does not encounter attack[2] during the search, then strategies defend[2] and defend[1, 2] would be "equally helpful" and either one could be selected depending on the tie-breaking procedure implemented. Second, finding the smallest set $\Sigma^h_{\sigma_k}$ given $S$ is equivalent to solving a set cover problem, which is NP-hard [Garey and Johnson, 1979]. 2L uses a polynomial-time greedy algorithm to approximate a solution to the set cover problem. Namely, we define an initially empty set $S'$. Then, in every iteration, we select the strategy $\sigma$ in $\Sigma_{\sigma_k}$ that is a best response to the largest number of strategies in $S \setminus S'$ and we add all strategies for which $\sigma$ is a best response to $S'$. We stop when $S = S'$. The strategies $\sigma$ selected from $\Sigma_{\sigma_k}$ in this procedure approximates to $\Sigma^h_{\sigma_k}$, which gives us an approximation to the redundant strategies.

2L works by executing the following steps:

1. Initialize $\Sigma_{-i}$ and $\Sigma_{\sigma_{-i}}$ with $\{\sigma_{1,-i}\}$ for some arbitrary strategy $\sigma_{1,-i}$; compute a best response $\sigma_{1,i}$ to $\sigma_{1,-i}$ and initialize $\Sigma_i$ and $\Sigma_{\sigma_i}$ with $\{\sigma_{1,i}\}$. Define meta-strategies $\sigma_i$ and $\sigma_{-i}$ as $(1.0)$.
2. While there is time for learning and alternating $k$ to be $-i$ in one iteration and $i$ in the next, execute:

(a) Compute a best response $\sigma$ to $\sigma_{-k}$ and add it to $\Sigma_k$ and to $\Sigma_{\sigma_k}$; set $\sigma_k[j] = 1.0/|\Sigma_{\sigma_k}|$ for all $\sigma_j$ in $\Sigma_{\sigma_k}$.

(b) Set $\sigma_{-k}[j] = 0$ and remove it from $\Sigma_{\sigma_{-k}}$ all $\sigma_{j,-k}$ in $\Sigma_{-k}$ that were estimated as redundant.

(c) Set $\sigma_{-k}[j] = 1.0/|\Sigma_{\sigma_{-k}}|$ for all $\sigma_j$ in $\Sigma_{\sigma_{-k}}$.

2L starts by initializing the set of strategies of the empirical game and the set of strategies in the support of the meta-strategy with an arbitrary strategy for one of the players ($-i$ in the pseudocode above). Then, it computes a best response to this arbitrary strategy and uses the best response to initialize $\Sigma_i$ and $\Sigma_{\sigma_i}$. The meta-strategies are of the form $(1.0)$ because the empirical game has a single strategy for each player (see Step 1 above). Step 2 refers to PPSRO's loop, where it computes best responses while alternating the players. Once a best response $\sigma$ is computed to strategy $\sigma_{-k}$, it is added to the support of $\sigma_k$ with uniform probability (see Step 2a).

2L estimates which strategies in the support of $\sigma_{-k}$ are redundant while computing the best response $\sigma$ to $\sigma_{-k}$. In Step 2b, 2L removes the redundant strategies from the support of $\sigma_{-k}$ and, in Step 2c, redistributes the probabilities such that each strategy in the support has equal probability.

**Example 5.** *In Example 2, we showed that DO fails to include both* `attack[1, 2, 3]` *and* `attack[1, 2, 5]` *in the support of the meta-strategy $\sigma_{-i}$, thus missing the guidance information* `attack[1, 2, 3]` *provides. Once the strategy* `attack[1, 2, 5]` *is added to the empirical game, the meta-strategy will automatically have both* `attack[1, 2, 3]` *and* `attack[1, 2, 5]` *in its support. In contrast with DO, 2L retains both strategies in the support of $\sigma_{-i}$ for the next iteration as long as strategies such as* `defend[1, 2, 3]` *and* `defend[1, 2, 5]` *are evaluated in search as both* `attack[1, 2, 3]` *and* `attack[1, 2, 5]` *will be flagged as helpful.*

A weakness of 2L as presented above is that it can flag as redundant a strategy that is helpful if it does not sample enough strategies in search. For example, if the meta-strategy for Rangers has both `defend`[1] and `defend`[2] in its support, but it never evaluates a strategy that attacks gate 1 in search, then `defend`[1] will mistakenly be removed from the meta-strategy's support. We implement the following enhancement to fix this weakness. Whenever the search returns a best response $\sigma$ to a meta-strategy $\sigma_{-i}$ (resp. $\sigma_i$), we evaluate $\sigma$ against all strategies in the empirical game, including those not in the support of $\sigma_{-i}$ (resp. $\sigma_i$). If there is a strategy $\sigma'$ in the empirical game that is a best response to $\sigma$, then it must be that 2L mistakenly removed $\sigma'$ from the support of the meta-strategy. In this case, we repeat the search for a best response with $\sigma'$ added to the support of the meta-strategy.

This enhancement can increase the number of times the search algorithm is invoked in each iteration of the PPSRO loop. While we perform a single search per iteration with IBR, FP, and DO, in the worst case, 2L can perform a number of searches that is equal to the number of strategies in the game. This is because, in the worst case, we add all strategies of the empirical game to the support of the meta-strategy. Despite the possible extra searches, preliminary experiments showed that this enhancement improves the sampling efficiency of 2L. All results of this paper use this enhancement.

In practice, we do not have the guarantee that the search algorithm used in PPSRO's main loop is able to return a best response to a meta-strategy. So we use whichever approximation the search returns as if it was a best response to the meta-strategy. Moreover, depending on the game, we might not be able to immediately recognize a best response to strategy once we see one as one would have to prove the strategy to be a best response. This could be problematic, for example, while implementing the enhancement that 2L re-runs the search if there is a strategy in the empirical game that is a best response to the strategy the search returns. We run our experiments in games with utilities of $-1, 0, +1$. If a best response cannot be easily verified (e.g., MicroRTS), then we consider that $\sigma$ is a best response to $\sigma'$ if $\mathcal{U}(s_{\text{init}}, \sigma, \sigma') = +1$.

Once 2L reaches a computational budget, it can return different strategies as its approximated solution to Equation 1. Similarly to IBR, it can return the last strategy added to the empirical game for each player. 2L can also return a mixed strategy that is given by the distribution of strategies added to the empirical game, as FP does. We can also solve the resulting empirical game with linear programming, like DO does, and return the resulting strategy. In this paper, we assume the games have a pure dominant strategy for which IBR's approach of returning the last strategy added to the empirical game is suitable; this is what we use in our experiments.

# 7 Empirical Evaluation

## 7.1 Problem Domains

In addition to P&R, we introduce Climbing Monkey (CM), another two-player zero-sum game with a trivial optimal strategy that is also challenging in the context of programmatic strategies. In CM, monkeys need to climb to a branch of a tree that is higher than the branch the opponent's monkey is able to reach. The branches need to be climbed one at a time, without skipping any branch. The monkey that climbs to a higher branch wins the game. The game ends in a draw if both monkeys climb to a branch of the same height. For a tree with $n$ branches, a dominant programmatic strategy is $\texttt{climb}[1], \texttt{climb}[2], \cdots, \texttt{climb}[n]$. Similarly to P&R, CM is challenging because, depending on the number of branches, it requires one to synthesize long programs.

In P&R, learning algorithms perform better if using a larger number of strategies in the support of meta-strategies as having many strategies helps Rangers converge to a strategy that protects all gates. CM is a game where all one needs to use is the last strategy added to the empirical game, i.e., the strategy that allows the monkey to climb to the highest branch. We hypothesize that 2L is able to detect which strategies are needed in the support of the meta-strategies for these two games.

We also evaluate 2L in MicroRTS, a real-time strategy game designed for research. There is an active research community using MicroRTS as a benchmark for evaluating intelligent systems.[1] MicroRTS is a game played with real-time constraints and very large action and state spaces [Lelis, 2021]. Each player can control two types of stationary units

[1] https://github.com/Farama-Foundation/MicroRTS/wiki

(Bases and Barracks) and four types of mobile units (Workers, Ranged, Light, and Heavy). Bases are used to store resources and train Worker units. Barracks can train Ranged, Light, and Heavy units. Workers can build stationary units, harvest resources, and attack opponent units. Ranged, Light, and Heavy units have different amounts of hit points and inflict different quantities of damage to the opponent units. Ranged units differ from each other by causing damage from long distances. In MicroRTS, a match is played on a grid, which represents the map. Different maps might require different strategies for playing well the game.

### 7.2 Empirical Methodology

The games of P&R and CM allow for a comparison of IBR, FP, DO, and `2L` that is easy to understand and analyze as they have trivial optimal strategies. The experiments with MicroRTS allow us to compare not only existing learning algorithms with `2L`, but also other methods for playing MicroRTS. Namely, we compare the programmatic strategies of IBR, FP, DO, and `2L` with programmatic strategies human programmers wrote to win the last two competitions: COAC[2] and Mayari.[3] We also include two programmatic strategies that have been used in the MicroRTS competition since 2017: WorkRush (WR) and LightRush (LR). LR was the winner of the 2017 competition. We use seven maps of different sizes: 8×8A BasesWorkers, 16×16 BasesWorkers, 24×24A BasesWokers, 24×24 DoubleGame, BWDistantResources 32×32, Chambers 32×32, and 32×32 BasesWorkers. We consider two starting locations (the location of player's base) on each map. When evaluating two strategies, to ensure fairness, each strategy plays an equal number of matches in both locations against the other strategy.

We are interested in evaluating the sample efficiency of the different approaches, i.e., the strength of the strategies they synthesize as a function of the number of games they need to play to synthesize the strategies. We present plots such as the one in Figure 2, where the x-axis shows the number of games played and the y-axis a metric of performance. We measure performance in P&R in terms of number of gates Rangers protect; for CM we measure how high a monkey climbs.

In the MicroRTS plots (Figure 3) we measure performance in terms of the winning rate of the strategy synthesized by each method in a tournament. The tournament is played among strategies synthesized by all systems after playing the same number of games. In the tournament, each strategy plays each other strategies 10 times, 5 in each starting location of the map. MicroRTS matches can finish in draws. Following previous work, we assign a score of 1.0 for each win and 0.5 for each draw. The winning rate is given by adding the number of wins with half the number of draws, divided by the total number of matches [Ontañón, 2017].

Since the mutation operation we use in the hill climbing algorithm is stochastic, we perform multiple independent runs of each experiment and report the average results and standard deviation. The number of runs performed in each experiment is specified below. We use Medeiros *et al.* [2022]'s DSL for MicroRTS. [4]

[2]https://github.com/Coac/coac-ai-microrts

[3]https://github.com/barvazkrav/mayariBot

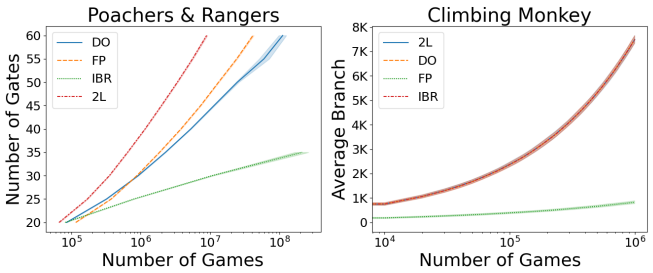



Figure 2: The average number of gates reached on P&R (left) and the average number of branches on CM (right) for different algorithms.

### 7.3 Empirical Results

**P&R.** Figure 2 (left) presents the results for P&R, where each line represents the average number of gates protected over 10,000 independent runs of the algorithms. The x-axis is in a log scale. `2L` derives strategies that protect more gates with many fewer games played than all the other approaches tested. IBR performs worst likely because it can cycle through strategies it has already seen in learning, as we illustrated in Example 1. FP performs best after `2L` as it is able to remember all previous strategies. However, FP uses more strategies than it needs to make progress, which explains the gap between the `2L` and FP lines.

**CM.** Figure 2 (right) presents the results for CM, where each line is an average of branches climbed over 300 independent runs; the x-axis is in log scale. IBR, DO, and `2L` perform equally well as they all use only the last strategy added to the empirical game in the meta-strategy, which in this domain is the true set of helpful strategies. FP is the worst-performing method in this domain as it unnecessarily uses all strategies from the empirical game in the support of the meta-strategy.

**MicroRTS.** Figure 3 presents the results for MicroRTS, with plots for four representative maps. Each line represents the average of 40 independent runs of each system. `2L` is never worse and it is often far superior to the other methods. DO also performs well in most maps, but it is outperformed by a large margin by `2L` and FP in BaseWorkers32x32A. IBR performs poorly on all maps.

### 7.4 Simulated Competition Results (MicroRTS)

Table 2 shows the average results for a set of simulated competitions using the seven maps mentioned in the empirical methodology section. Each entry of the table shows the average winning rate and standard deviation of the row method against the column method; the last column shows the average and standard deviation across a given row of the table. The numbers in Table 2 are the average winning rate computed by simulating 5 tournaments. The strategy we use in each tournament for IBR, FP, DO, and `2L`is generated as follows. We run each method 8 times, thus producing 8 different strategies each. Then, we run a round-robin evaluation among the 8 strategies of a given method and the winning strategy in

[4]Our code is at https://github.com/rubensolv/LocalLearnerIJCAI

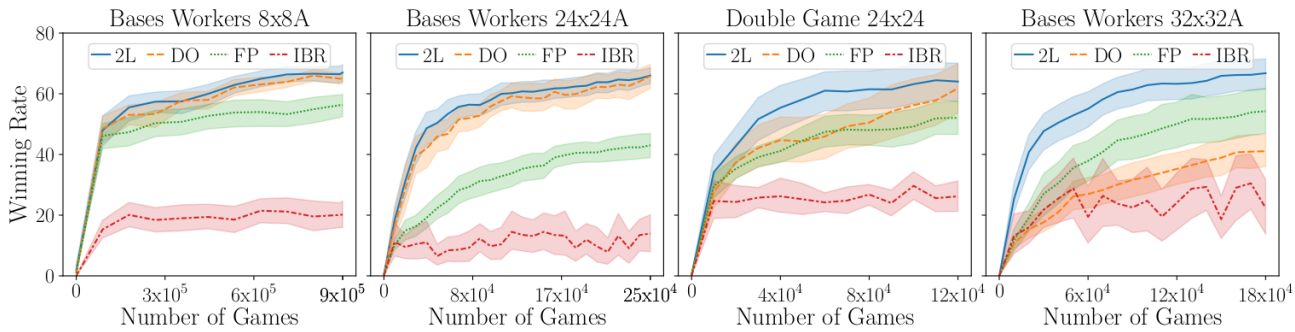



Figure 3: Results of each learning algorithm on four MicroRTS maps using Hill-Climbing local search. These curves represent the learning rate of each algorithm in comparison with their opponents using the same amount of games.

| Method | LR | WR | CoaC | IBR | Mayari | FP | DO | 2L | Total |
|---|---|---|---|---|---|---|---|---|---|
| **LR** | - | 0.71 ±.29 | 0.14 ±.14 | 0.20 ±.15 | 0.00 ±.00 | 0.23 ±.15 | 0.16 ±.18 | 0.13 ±.11 | 0.23 ±.21 |
| **WR** | 0.29 ±.29 | - | 0.36 ±.36 | 0.25 ±.13 | 0.26 ±.26 | 0.22 ±.10 | 0.14 ±.13 | 0.15 ±.06 | 0.24 ±.15 |
| **CoaC** | 0.86 ±.14 | 0.64 ±.36 | - | 0.52 ±.21 | 0.09 ±.09 | 0.37 ±.28 | 0.30 ±.26 | 0.24 ±.22 | 0.43 ±.26 |
| **IBR** | 0.80 ±.15 | 0.75 ±.13 | 0.48 ±.21 | - | 0.49 ±.25 | 0.23 ±.26 | 0.22 ±.26 | 0.22 ±.24 | 0.46 ±.13 |
| **Mayari** | 1.00 ±.00 | 0.74 ±.26 | 0.91 ±.09 | 0.51 ±.25 | - | 0.36 ±.18 | 0.45 ±.16 | 0.34 ±.19 | 0.62 ±.17 |
| **FP** | 0.77 ±.15 | 0.78 ±.10 | 0.63 ±.28 | 0.77 ±.26 | 0.64 ±.18 | - | 0.45 ±.29 | 0.41 ±.34 | 0.63 ±.13 |
| **DO** | 0.84 ±.18 | 0.86 ±.13 | 0.70 ±.26 | 0.78 ±.26 | 0.55 ±.16 | 0.55 ±.29 | - | 0.43 ±.30 | 0.67 ±.15 |
| **2L** | 0.87 ±.11 | 0.85 ±.06 | 0.76 ±.22 | 0.78 ±.24 | 0.66 ±.19 | 0.59 ±.34 | 0.57 ±.30 | - | 0.72 ±.13 |

Table 2: Average winning rate and standard deviation of each learning method (column method) against agents used in the MicroRTS competition. The row values represent the percentage against the column enemy.

this evaluation is used as the method's strategy in our tournament. For a given tournament, the winning rate is computed by having the strategy of each method play the other strategies 10 times in each map, 5 for each starting location.

2L is the only method to obtain an average winning rate above 0.50 against all other opponents; 2L also obtained the highest average winning rate when considering all opponents: 0.72 (column "Total"). In particular, it obtains average winning rates of 0.76 and 0.66 against COAC and Mayari, respectively, the winners of the two latest competitions. A Welch's t-test shows that the difference between 2L and the competition winners COAC and Mayari, in terms of total average winning rate, is statistically significant with $p < 10^{-5}$.

These results on P&R, CM, and MicroRTS show that 2L's approach for defining its meta-strategy can be quite effective in guiding a synthesizer that uses HC search.

## 8 More Related Works

In addition to PSRO [Lanctot *et al.*, 2017], this work is related to programmatic policies [Verma *et al.*, 2018], where the goal is to synthesize human-readable programs encoding policies for solving reinforcement learning problems [Bastani *et al.*, 2018; Verma *et al.*, 2019]. Generalized planning (GP) is also related as it deals with the synthesis of programs for solving classical planning problems [Bonet *et al.*, 2010; Srivastava *et al.*, 2011; Hu and De Giacomo, 2013; Aguas *et al.*, 2018]. 2L differs from these works because it learns how to solve two-player games, while the latter focus on single-agent problems.

Mariño *et al.* [2021; 2022] also use local search algorithms to synthesize programmatic strategies and they also evaluate their system in MicroRTS. In terms of learning algorithms, they only use IBR, so the IBR version in our experiments can be seen as a representation of their work. Medeiros *et al.* [2022] presents a system for learning sketches with imitation learning as a means of speeding up the computation of programmatic best responses. They focus on the computation of best responses, so their solution can in theory be combined with any of the learning algorithms we evaluated in this paper.

## 9 Conclusions

In this paper, we introduced Local Learner, a learning algorithm based on the PSRO framework to guide local search algorithms on the task of synthesizing programmatic strategies. 2L uses information collected from the computation of best responses to approximate a set of helpful strategies to have in the support of 2L's meta-strategy, which serves as a guiding function for the search. We show empirically in three games the advantages of 2L over adaptations of the learning algorithms IBR, FP, and DO to programmatic strategies. The empirical results show that 2L's approach of using information collected during search to determine its own guiding function can be quite effective in practice. 2L is never worse than the other learning algorithms and is often far superior. In particular, in the game of MicroRTS, we simulated a competition with the last two winners of the annual MicroRTS competition, and the strategies 2L synthesized obtained the highest winning rate across all evaluated systems.

## Acknowledgments

This research was supported by Canada's NSERC and the CIFAR AI Chairs program and Brazil's CAPES. The research was carried out using computational resources from Compute Canada. We thank the anonymous reviewers for their feedback.

## References

[Aguas *et al.*, 2018] Javier Segovia Aguas, Sergio Jiménez, and Anders Jonsson. Computing hierarchical finite state controllers with classical planning. *Journal of Artificial Intelligence Research*, 62:755–797, 2018.

[Aleixo and Lelis, 2023] David S. Aleixo and Levi H. S. Lelis. Show me the way! Bilevel search for synthesizing programmatic strategies. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 2023.

[Anthony *et al.*, 2020] Thomas Anthony, Tom Eccles, Andrea Tacchetti, János Kramár, Ian Gemp, Thomas Hudson, Nicolas Porcel, Marc Lanctot, Julien Pérolat, Richard Everett, et al. Learning to play no-press diplomacy with best response policy iteration. *Advances in Neural Information Processing Systems*, 33:17987–18003, 2020.

[Bastani *et al.*, 2018] O. Bastani, Y. Pu, and A. Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2499–2509, 2018.

[Bonet *et al.*, 2010] Blai Bonet, Héctor Palacios, and Héctor Geffner. Automatic derivation of finite-state machines for behavior control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, page 1656–1659. AAAI Press, 2010.

[Brown, 1951] GW Brown. Iterative solution of games by fictitious play, 1951. *Activity Analysis of Production and Allocation (TC Koopmans, Ed.)*, pages 374–376, 1951.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Hu and De Giacomo, 2013] Yuxiao Hu and Giuseppe De Giacomo. A generic technique for synthesizing bounded finite-state controllers. *Proceedings of the International Conference on Automated Planning and Scheduling*, 23(1):109–116, 2013.

[Lanctot *et al.*, 2017] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[Lelis, 2021] Levi H. S. Lelis. Planning algorithms for zero-sum games with exponential action spaces: A unifying perspective. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 4892–4898, 2021.

[Mariño and Toledo, 2022] Julian R. H. Mariño and Claudio Toledo. Evolving interpretable strategies for zero-sum games. *Applied Soft Computing*, page 108860, 2022.

[Mariño *et al.*, 2021] Julian R. H. Mariño, Rubens O. Moraes, Tassiana C. Oliveira, Claudio Toledo, and Levi H. S. Lelis. Programmatic strategies for real-time strategy games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 381–389, 2021.

[McMahan *et al.*, 2003] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 536–543, 2003.

[Medeiros *et al.*, 2022] Leandro C. Medeiros, David S. Aleixo, and Levi H. S. Lelis. What can we learn even from the weakest? Learning sketches for programmatic strategies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7761–7769. AAAI Press, 2022.

[Muller *et al.*, 2020] P Muller, S Omidshafiei, M Rowland, K Tuyls, J Pérolat, S Liu, D Hennes, L Marris, M Lanctot, E Hughes, et al. A generalized training approach for multiagent learning. In *ICLR*, pages 1–35. ICLR, 2020.

[Ontañón *et al.*, 2018] Santiago Ontañón, Nicolas A. Barriga, Cleyton R. Silva, Rubens O. Moraes, and Levi H. S. Lelis. The first microrts artificial intelligence competition. *AI Magazine*, 39(1), 2018.

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

[Srivastava *et al.*, 2011] Siddharth Srivastava, Neil Immerman, Shlomo Zilberstein, and Tianjiao Zhang. Directed search for generalized plans using classical planners. In *Proceedings of the International Conference on Automated Planning and Scheduling*. AAAI, 2011.

[Verma *et al.*, 2018] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. Programmatically interpretable reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 5052–5061, 2018.

[Verma *et al.*, 2019] Abhinav Verma, Hoang Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 32, pages 1–12. Curran Associates, Inc., 2019.