A Reinforcement Learning Approach to Domain-Knowledge Inclusion Using Grammar Guided Symbolic Regression

Laure Crochepierre, Lydia Boudjeloud-Assala, Vincent Barbesant

A Reinforcement Learning Approach to Domain-Knowledge Inclusion Using Grammar Guided Symbolic Regression

Laure Crochepierre[1,2*], Lydia Boudjeloud-Assala[1] and Vincent Barbesant[2]

[1*]Université de Lorraine, CNRS, LORIA, F-57000, Metz, France.
[2]Réseau de Transport d'Electricité (Rte) R&D, Paris, France.

*Corresponding author(s). E-mail(s): laure.crochepierre@{rte-france.com, univ-lorraine.fr};

Abstract

In recent years, symbolic regression has been of wide interest to provide an interpretable symbolic representation of potentially large data relationships. Initially circled to genetic algorithms, symbolic regression methods now include a variety of Deep Learning based alternatives. However, these methods still do not generalize well to real-world data, mainly because they hardly include domain knowledge nor consider physical relationships between variables such as known equations and units. Regarding these issues, we propose a Reinforcement-Based Grammar-Guided Symbolic Regression (RBG2-SR) method that constrains the representational space with domain-knowledge using context-free grammar as reinforcement action space. We detail a Partially-Observable Markov Decision Process (POMDP) modeling of the problem and benchmark our approach against state-of-the-art methods. We also analyze the POMDP state definition and propose a physical equation search use case on which we compare our approach to grammar-based and non-grammar-based symbolic regression methods. The experiment results show that our method is competitive against other state-of-the-art methods on the benchmarks and offers the best error-complexity trade-off, highlighting the interest of using a grammar-based method in a real-world scenario.

Keywords: Symbolic Regression, Reinforcement Learning, Probabilistic Context Free Grammar, Domain-Knowledge

1 Introduction

Finding a generic symbolic representation from an observation set has been of a long interest in the Physics community. Long before Newton's fallen apple, scientists have been working on discovering symbolic relationships that satisfy observations, current knowledge, and already known equations. In astronomy, for example, the search for a closed-form solution has been an ongoing interest notably for Kepler's study of planetary motion and still today for the modeling of albedos (Heng et al, 2021). However, in recent years, the number of observations available has surged thanks to the increase of sensor monitoring systems in Big Data environments and, finding the best possible equation to model complex systems is now a tedious task without computer assistance.

This automatic equation search task is known as Symbolic Regression (SR). It aims at finding a symbolic function f that matches the relationship $f(X) = y$ between an observation set $X \in \mathbb{R}^n$ described with n variables and a target variable $y \in \mathbb{R}$ to explain y from X. SR was, until recently, mainly performed using Genetic Programming (GP) (Koza, 1990), a family of techniques that draws inspiration from Darwinian evolution to search for solutions automatically. However, the initial GP formulation is not suited for large search spaces (Ebner, 1999). To cope with this limitation, since the early works on GP (Koza, 1990), multiple extensions have been proposed to restrict the search space, such as Strongly Typed Genetic Programming (Montana, 1995) or Grammar Guided Genetic Programming (G3P) (Koza et al, 2006). These methods share the interesting property of providing a way to enforce domain knowledge into the learning process (Ratle and Sebag, 2000) and are thus applied to various real-world problems (Cherrier et al, 2019a).

Meanwhile, Deep Learning (DL) methods have progressively become ubiquitous because of their high representational capacity when trained on large datasets (Russakovsky et al, 2015). However, in the general case, DL often has a "black box" behavior and lack of interpretability. Thus, combining DL and SR by taking advantage of the computational capacity of DL and the expressiveness of SR would help provide more human-readable results. Along with the rapid development of DL, there have also been a growing interest in Reinforcement Learning (RL) methods to solve decision-making problems. RL is a Machine Learning paradigm inspired by behavioral psychology concerned with how an agent should act in an environment to maximize a cumulative reward (Sutton and Barto, 2018). Deep reinforcement learning (Deep RL) approaches especially show great power in solving complex sequential problems. Recent work in these both domains also offers viable alternatives to GP on SR tasks (Petersen et al, 2021; Udrescu and Tegmark, 2020). Still, unlike GP-based methods, they are not yet able to include sophisticated domain-related knowledge or constraints.

In this work, we propose a method called Reinforcement Based Grammar Guided Symbolic Regression (RBG2-SR) to tackle SR with a Deep Reinforcement Learning (Deep RL) approach using a Backus-Naur Form (BNF) (Knuth, 1964) Context Free Grammar action space which constrains the solution space

only to domain-viable solutions. Our method enforces domain-related constraints as grammatical rules in a human-readable manner. These constraints are then translated into directly interpretable symbolic outputs. We also propose a general Partially-Observable Markov Decision Process (POMDP) (Kaelbling et al, 1998) modeling of the SR problem. We show that our approach performs better than other tested methods with the same grammar on the tested benchmarks. We also offer comparative results on a real-world scenario to show how our approach could use a BNF grammar to take advantage of expert knowledge.

The rest of this paper is organized as follows. First, Section 2 summarizes related state-of-the-art works. In Sections 3 and 4, we respectively describe the proposed method and its corresponding experimental results. Finally, Section 5 offers concluding remarks and perspectives.

2 Related Works

2.1 Symbolic Regression

Symbolic Regression (SR) is the process of searching for a symbolic relationship, also called *expression*, that accurately matches a given dataset. Found expressions can be represented as trees where nodes contain operations and variables. SR was early investigated with Genetic Programming (GP) approaches (Koza, 1990, 1992). They iteratively evolve a population of individuals (each individual representing an *expression*) across multiple generations through evolutionary operations. Many works today are descendants of these works, and propose to overcome the problems of GP, such as *bloat* (Silva, 2008) (the individual's complexity explosion over the generations) or convergence (Rosca, 1996) issues, for example, with multi-objective strategies (Tamaki et al, 1996) or a partial derivative-based error fitness (Schmidt and Lipson, 2009).

Driven by the current need for more interpretable models, non-GP-based methods have been developed to tackle SR. These methods propose to take advantage of the computational capacity of neural networks (Hornik et al, 1989) while providing an interpretable solution. For instance, they offer to encode the expression in the neural network structure and activation functions (Sahoo et al, 2018; Kim et al, 2020), to predict a string expression (Anjum et al, 2019) with Recurrent Neural Networks, or to use Deep Reinforcement Learning (Deep RL) as a search engine (Petersen et al, 2021). By combining partial derivative and neural networks, AI Feynman method (Udrescu and Tegmark, 2020) propose to make use of simplifying properties (such as units, symmetry, separability.. etc.) intrinsically present in physical expressions to repeatedly cut the global SR problem into simpler ones with fewer variables. Other methods that do not use neural networks rely on bayesian optimization (Jin et al, 2019) or perform nonlinear basis function expansion (McConaghy, 2011).

However, as designed, these methods do not insert custom knowledge and expertise into the symbolic expression construction.

2.2 Knowledge Insertion by Constraints

Because the search space in SR is very large and can leads to local optima, it is relevant to restrict the function space by removing sub-optimal search regions. In line with Koza's work on GP, a preliminary solution called Strongly Typed GP (STGP) (Montana, 1995) proposed to enforce data types constraints for computer program search, which decreases the search time and improves the generalizability of the found solutions. Regarding SR for physical laws (re)discovery, other kinds of constraints can be considered, such as physical units. More precisely, as each variable comes with its physical unit, arbitrarily combined variables can produce illegal unit combinations. To this end, dimensionally awareness GP (Keijzer and Babovic, 1999) was initially proposed to take into account unit knowledge in GP by minimizing the distance to a legal unit in the fitness function. Note that, more than just constraining the search space, these constraints enforce structured knowledge and expertise about the problem within the learning. This knowledge can also take the form of ontologies (Prieschl et al, 2019) to include prior knowledge as additional input features. However, both dimensionally-aware GP (Keijzer and Babovic, 1999) and ontology-guided GP (Prieschl et al, 2019) do not ensure to produce dimensionally valid expressions.

Toward this goal, the definition of explicit constraints can guarantee to produce only legal expressions with respect to the constraints. These constraints can be grammatical (Whigham et al, 1995) or ontological (Lucena-Sánchez et al, 2021). In Grammar-Guided Genetic Programming (G3P) (Whigham et al, 1995), also called Grammar-Based GP, a Context-Free Grammar (CFG) (Cremers and Ginsburg, 1975) is used to define constraint rules. Grammatical rules allow defining physical units, thanks to which G3P has found a variety of industrial applications (Crochepierre et al, 2021; Cherrier et al, 2019b). CFGs are often written in Backus-Naur form (BNF) (Knuth, 1964), which is made of:

- *Non-Terminals* also called *symbols* (ex. <s>). One of the non-terminals is called *start symbol*.
- *Terminals*, character strings to replace non-terminals (ex. "a", "b", "+"). The set of terminals can contain input features and operators to combine features.
- *Rules* which defines how the terminals and non-terminals are connected. A set of rules for a specific symbol is called a *production rule* where rules are separated by a vertical line |, and ::= means "defined as" (for example <s> ::= "a" | "b" | <s>+<s>).

Eventually, a grammar contains multiple production rules, one per symbol. To select which rule will replace a given symbol <symbol>, uniform sampling is made in <symbol> production. McKay et al. (McKay et al, 2010) provides a detailed survey of G3P strategies and genetic operations. Because of its ability to restrict the symbol space, CFG structures have been a privileged topic

of study. Outside SR, CFGs found other applications, such as in Grammar Variational Autoencoder (Kusner et al, 2017) where they are used for symbolic data representation along with molecule representation.

Nowadays, probabilistic CFG (noted PCFG) (Sakakibara, 2017) are preferred as they allow to weigh the importance of a rule. In addition to CFG rules, they assign a probability to each rule in a production rule so that all probabilities for a given symbol add up to 1. At the end of a production rule, a list preceded by a keyword `probs` and a double vertical line defines the probabilities associated to the rules. The production rule structure now becomes:

```
<symbol> ::= rule1 | rule2 |...|| probs [prob_r1,prob_r2,...]
```

PCFGs are of particular importance as they allow estimating the probabilities associated with each rule and discard unuseful rules. Probability distributions can be updated according to sampled expressions using Linear Genetic Programming (Sotto and de Melo, 2017) or Monte Carlo sampling (Brence et al, 2021).

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning approach to solving Markov Decision Processes (MDPs), where the MDP is mainly defined by its *state* space, *action* space, *state transitions probabilities*, and *reward*. In the RL paradigm (Sutton and Barto, 2018), an agent learns to achieve a task by interacting with its environment at discrete time steps. At each time step, the agent chooses an action among available actions according to a given policy. Next, the action is sent to the environment, which gives back a reward feedback and a new state to the agent. Then, the agent can learn from the received reward signals to improve its policy. RL strategies are mainly valued-based like Deep Q-Network (Mnih et al, 2015), or policy-based like REINFORCE algorithm (Williams, 1992). Value-based RL learns a value function and deduce a policy from values. In contrast, policy-based RL explicitly learns a policy π and keeps it in memory during learning. In Deep Reinforcement Learning (Deep RL), value and policy functions are approximated using neural networks. For example, the REINFORCE algorithm (Williams, 1992) at its core uses the policy gradient theorem to update the probability distribution of actions. Actor-Critic algorithms (Konda and Tsitsiklis, 2000) are inheriting from both strategies by trying to learn alongside a policy and its value to reduce variance and improve converge.

RL has been applied to a variety of domains in pattern recognition (Piñol et al, 2012; Khurana et al, 2018; Bertsekas, 2019), from feature construction (Khurana et al, 2018) to feature-based aggregation (Bertsekas, 2019). Including knowledge in RL is of paticular importance especially for safety issues (Alshiekh et al, 2018) where shielding strategy is used to correct actions if the chosen one causes a violation of some specified sort. However, few works have focused on building a symbolic representation for data. Recent work on this topic includes the use of RL to search among a library of operators and

features for SR (Petersen et al, 2021) or the creation of symbolic computer programs (Verma et al, 2018).

Regarding interpretability, Deep Learning and RL approaches mostly produce complex solutions that are often hard to interpret. It is especially true in environments where most agents perform according to a black-box policy. To make these black-box approaches more grey and learn more interpretable RL policies, recent work proposes to learn symbolic policies either with GP (Hein et al, 2018) or Deep RL (Landajuela et al, 2021). However, even if they have interpretable outputs, most of these solutions do not include domain-related knowledge within the learning to ensure that the output solutions follow the prior knowledge.

3 Reinforcement Based Grammar Guided Symbolic Regression (RBG2-SR)

3.1 Definition of the Reinforcement Learning Environment

In this work, we adopt a Markov Decision Process (MDP) (Sutton and Barto, 2018) modeling of the SR problem. More precisely, we choose to consider a Partially-Observable Markov Decision Process (POMDP)(Kaelbling et al, 1998) in a finite episodic setting with maximum horizon H. This section is dedicated to the definition of the main components of the MDP in a RL setting, namely: *state* and *action* spaces and *reward*. In Section 3.1.1, we define State and action spaces for SR and Section 3.1.2 details the reward definition along with its properties. Section 3.1.3 combine the definitions from Sections 3.1.1 and 3.1.2 to define the whole POMDP. Finally, Section 3.1.4 details how to learn a policy over action, used to generate the action probabilities given the current state.

3.1.1 Grammatical state and action space

Let us consider a RL setting where the overall task is to find an optimal symbolic function f^* so that $f^* = \text{argmin}_{f \in F_G} ||y - f(X)||$, with F_G being the function space accessible from a given grammar G. The grammar is defined by the tuple $(\sigma_{start}, (\sigma_{nt})_{nt \in NT}, (\sigma_t)_{t \in T}, \rho, \Psi)$ with σ_{start} the start symbol of the grammar, $(\sigma_{nt})_{nt \in NT}$ a set of non-terminals, $(\sigma_t)_{t \in T}$ a set of terminals, ρ the production rules to combine terminals/non terminals, and Ψ the probability associated to each production rule. Figures 1 and 4 in Section 4 show example BNF Grammar inspired by the work of Sotto and Melo (Sotto and de Melo, 2017).

Given this notation, we propose to define the construction of f^* as a sequential decision making problem where an agent sequentially chooses rules in the grammar to build up the function f. In this grammatical space, we first specify the maximal number of steps to create f, called the maximal horizon H. We then define for each step $h \in [0, ..., H]$ an *action* a_h as the *selection of a*

rule in the production rule accessible from the current state. We also propose to define the *state* s_h at step h by $s_h = (a_h^{past}, a_h^{parent}, a_h^{siblings}, d_h, \sigma_h, m_h, \eta_h)$, with $a_h^{past} = [a_0, ..., a_{h-1}]$ all previously selected action, a_h^{parent} the action taken by the parent in the parse tree, $a_h^{siblings}$ the action taken by each already computed siblings in the parse tree, d_h the depth of the expression tree at step h, σ_h the current type of symbol to find at step h, m_h a mask over accessible actions from the current state symbol σ_h and η_h hidden information about the current state. We call sibling nodes those nodes with the same depth as the current node, and parent node the node above the current node in the parse tree. Alternative state definitions will be tested in Section 4.1.3. Initially, we define the state by the start symbol σ_{start}, previously selected actions is an empty list, m_0 masks out inaccessible actions from the initial symbol, and hidden information η_0 is randomly initialized. A *trajectory* of actions $\tau_k = (a_0^k, ..., a_h^k), h \in [H]$ is associated to each symbolic function f_k. We consider a case where the grammar is chosen and constructed so that there is always at least one action accessible at each step.

Algorithm 1 Single episode sampling, returns one function f per episode.

Require: maximal horizon H, policy π_θ, grammar G

function SAMPLE EPISODE(H, π_θ, G)
  $queue, a^{past}, a^{parents}, a^{siblings}, f \leftarrow$ empty
  $d \leftarrow 0$
  $\sigma \leftarrow$ `get_start_symbol`(G)
  $m \leftarrow$ `get_mask`(σ, G)
  $\eta \leftarrow$ `random_initialisation`$()$
  for h in H **do**
    $state \leftarrow ((a^{past}, a^{parent}, a^{siblings}, d, \sigma, m, \eta)$
    $action\_probs, \eta \leftarrow \pi_\theta(state)$ ▷ π_θ is described in Figure 3(a)
    $action \leftarrow$ `sample`$(action\_probs)$ ▷ Corresponds to "Action Sampling" blue box in Figure 3
    $a_{past} \leftarrow$ `append`$(a_{past}, action)$
    $\sigma_{NT}^{child}, \sigma_T^{child} \leftarrow$ `get_child_symbols`$(action, G)$
    $f \leftarrow$ `translate`$(f, \sigma_{NT}^{child}, \sigma_T^{child})$
    $queue \leftarrow$ `extend`$(\sigma_{NT}^{child}, queue)$ ▷ Put σ_{NT} at the start of the queue
    $\sigma \leftarrow$ `pop`$(queue)$
    $a^{parent}, a^{siblings} \leftarrow$ `get_parent_and_siblings`(σ, a_past)
    $m \leftarrow$ `get_mask`(G, σ)
    $d \leftarrow d + 1$
  end for
  return f
end function

The Algorithm 1 details the episode sampling procedure. A visual example of this expression sampling is provided in Figure 1. From a given grammar with

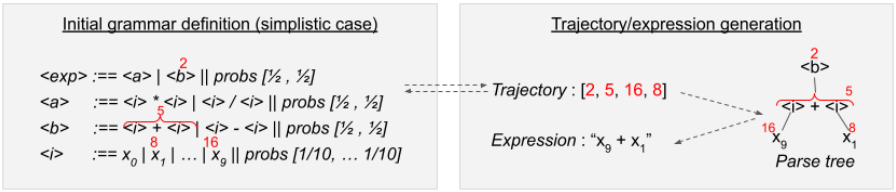

Fig. 1 "$x_9 + x_1$" Expression and trajectory generation from a given grammar. A simplistic grammar is given (left), with actions numbered from 1 to 16 and start symbol $< exp >$. On the right, a *trajectory* is sampled from this grammar, from which we define the corresponding *parse tree* and symbolic *expression*

16 actions and <exp> as start symbol, we generate a trajectory of actions. The trajectory construction goes as follow:

1. We begin by selecting the first action among accessible actions from the start symbol $< exp >$: either actions 1 or 2. Given the probabilities associated with these actions, we perform a weighted sampling with the probabilities listed in `probs` as weights. Action 2 is sampled with the rule "$< b >$". As this rule contains the non-terminal symbol $< b >$ we need to replace it with a rule from the grammar "accessible" for the symbol $< b >$.
2. The 3^{rd} row in the grammar defines actions accessible from $< b >$: actions 5 or 6. Given the weights, we sample action 5, "$< i > + < i >$". It contains two non-terminal symbols ($< i >$ and $< i >$) that need to be replaced in the next steps.
3. The non-terminal symbol replacement is performed on one symbol at a time in a depth-first search manner, by looping over the first symbol until reaching a terminal symbol. We iterate this procedure until either the maximal trajectory length is reached or all non-terminal symbols encountered in all action selections have been replaced by a terminal value.

3.1.2 Reward definition

The standard metric to minimize in SR is the Mean Squared Error (MSE). To match the RL definitions where the reward function is a monotonically increasing function, we use the squashing function $\frac{1}{1+x}$.

$$r_h = \begin{cases} 0 & if\ h < H \\ R = \frac{1}{1+MSE(y,\hat{y})} & if\ h = H \end{cases} \tag{1}$$

As the function f can only be evaluated at the end of the episode when the function is complete, the reward r equals 0 until the final step is reached and values $\frac{1}{1+MSE(y,\hat{y})}$ at step H, where $\hat{y} = f(X)$ as shown in Equation 1. We also note the expected cumulative reward R and highlight that $R = \sum_h r_h = r_H$. The sparsity property is used in Section 3.2.3 to simplify the REINFORCE algorithm (Williams, 1992) loss function.

3.1.3 Partially-Observable Markov Decision Process

Given the previous space, action and reward definitions, the POMDP we consider here is defined by a tuple $(S, A, r, P, H, \Omega, O)$ with S the state space, A the action space, $r : S \times A \to [0, 1]$ the reward function, $P : S \times A \to [0, 1]$ the transition kernel, $\Omega = (o_1, o_2, ..., o_K)$ a set of observations and O a set of conditional observation probabilities $O(o|s', a)$. We write as $P(s'|s, a)$ the probability of having a transition to state $s \in S$ when taking action a in state s. A POMDP setting is here considered to mitigate the fact that action effects are uncertain until the final state is reached and that the state might be partially observable. Each episode k builds up a trajectory of actions $\tau_k = (a_k^h)_{h \in [H]}$ that ends either when the maximum horizon is reached or when the function f_k constructed by τ_k is complete and can be evaluated.

3.1.4 Policy optimization

Given the grammatical action space defined in Section 3.1.1, there exists an optimal policy π^* capable of generating a trajectory as close as possible to the symbolic function f^*. Depending on the choice of a judiciously constructed grammar, it is even possible to generate an optimal trajectory exactly corresponding to f^*.

To search for this optimal symbolic function, we propose to learn a policy π_θ, parametrized by a vector θ to generate the weights of grammatical action rules at each step $h \in [H]$ of the trajectory, from which we sample the next action. More precisely, to build f, the stochastic policy π_θ assigns a probability vector to accessible actions from a given state. At each step h, the action is then sampled according to the probability vector given by $\pi_\theta(s_h, o_h)$. Using the reward defined in Section 3.1.2, we iteratively update the parameters of π_θ to sample, in the future, more relevant trajectories with respect to the defined reward.

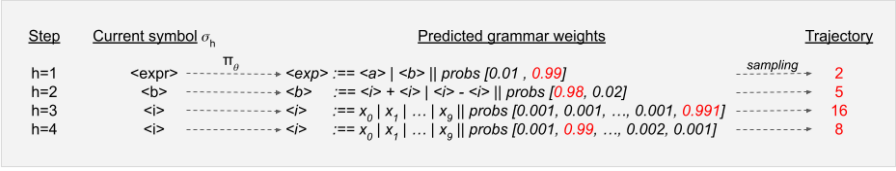

Fig. 2 Weights and trajectory generation with the grammar from Figure 1 after training. The red elements define the results of the sampling on the grammar. Grammar weights are updated by searching for the target expression "$x_9 + x_1$"

Using the same grammar example from Figure 1, we show in Figure 2 how this grammar could be updated when trained on the search of the expression "$x_9 + x_1$". After training, all unnecessary grammatical rules now have a low or zero probability, and it is easier to generate the correct target expression with this grammar.

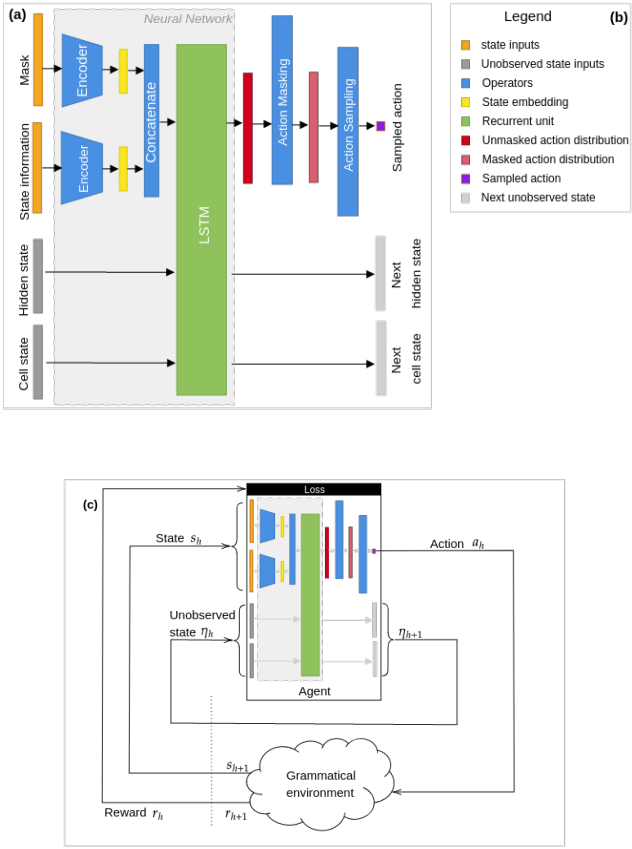

Fig. 3 Neural Network architecture to learn π_θ (Better seen in color). **(a)** shows the recurrent architecture to predict an action from a given state as described in Algorithm 1 lines 9 and 10. **(b)** is the color legend we use in **(a)** and **(c)**. **(c)** represents the POMDP agent-environment interaction

3.2 Learning π_θ and finding f^*: Implementation Details

3.2.1 POMDP Modeling with a Recurrent Neural Network

In order to find an optimal function f^*, we propose to use π_θ as an exploration tool, trained to emphasize exploration on the most relevant regions of the grammatical space. To do so, we learn π_θ with the policy-gradient (PG) algorithm called REINFORCE (Williams, 1992) on the neural network architecture described in Figure 3. In Figure 3(a), we describe (using the legend in Figure 3(b)) the recurrent architecture to predict an action a_h from a given state s_h at step h. To handle recurrency we choose Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) networks. They are a purposely designed structure to capture long term temporal dependencies and they are also able to model the partial observability of the above-mentioned MDP (Wierstra et al, 2007).

The neural network takes as input the state s_h (in orange), and hidden state η_h (in grey). All state-input features are encoded either using convolutions or feed-forward layers depending on their shape. The encoded state segments are then concatenated and handed to a LSTM cell along with the hidden state η_h. This recurrent cell both outputs an estimation of the observations for the next hidden state η_{h+1}, and an encoding of all actions in the grammar. This actions encoding is then masked using the state mask m_h (see Section 3.2.2) and outputs the distribution over actions *accessible* from the current state. Then, we sample action s_h according to the distribution over accessible action.

3.2.2 Invalid action Masking

To handle grammatical constraints on the action space, we propose to use at each step h a mask over inaccessible actions m_h in the current state s_h. Similarly to what is done by Huang and Ontañón (2020), invalid actions are masked through element-wise multiplication with a large negative value and followed by `softmax` function.

3.2.3 Exploration Driven Cost Function

REINFORCE algorithm (Williams, 1992) objective function, simplified with Equation 1, is $J_\theta = \mathbb{E}_\pi\big[R \sum_h \ log\ \pi_\theta(a_h|s_h)\big]$ from which we specify the optimal policy $\pi^*_\theta = \text{argmax}_\theta J_\theta$. As PG methods such as REINFORCE tends to have a large variance (Chung et al, 2021), it is common practice to subtract a baseline to the batch, such as a moving average of rewards across batches. However, SR is only interested in building a policy that maximizes the best-performing trajectories found during training, and these strategies might have a slow convergence because many sampled trajectories are irrelevant and have a low or zero final reward. Based on these comments, Petersen et al (2021) proposed a *risk-seeking policy gradient*, which only compute the cost function based on the top$-\epsilon$ quantile of the expected rewards R_ϵ, i.e., the most relevant trajectories of the batch:

$$J_\theta^{risk}(\epsilon) = \mathbb{E}_{\tau\sim\pi_\theta}\Big[(R(\tau|\theta) - R_\epsilon)\sum_{h=0}^{H}\ log\ \pi_\theta(a_h|s_h) \ \Big|\ R(\tau|\theta) > R_\epsilon\Big] \tag{2}$$

They also added a entropy term $\mathcal{H}$, weighted by $\lambda_\mathcal{H}$, to encourage exploration:

$$J_\theta^{entropy}(\epsilon) = \mathbb{E}_{\tau\sim\pi_\theta}\Big[\mathcal{H}(\tau|\theta) \ \Big|\ R(\tau|\theta) > R_\epsilon\Big] \tag{3}$$

Eventually, our final cost function becomes:

$$\begin{aligned} J_\theta^{tot}(\epsilon) &= J_\theta^{risk}(\epsilon) + \lambda_\mathcal{H} J_\theta^{entropy}(\epsilon) \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\begin{matrix}(R(\tau|\theta) - R_\epsilon)\sum_{h=0}^{H}\ log\ \pi_\theta(a_h|s_h) \\ +\lambda_\mathcal{H}\mathcal{H}(\tau|\theta)\end{matrix} \ \middle|\ R(\tau|\theta) > R_\epsilon\right] \end{aligned} \tag{4}$$

Given this cost function, the Neural network architecture is trained using the Algorithm 2.

Algorithm 2 Training procedure

Require: X, target y, maximal horizon H, batch size B, number of training iterations N, quantile threshold ϵ

$\pi_\theta \leftarrow \pi_{\theta_0}$
for n in N **do**
  $episodes \leftarrow$ `sample_episodes`(H, π_θ, G, B) ▷ Use Algorithm 1 B times
  $rewards \leftarrow$ `evaluate_episodes`$(episodes, X, y)$
  $best\_episodes\_and\_rewards \leftarrow$ `filter_episodes`$(episodes, rewards, \epsilon)$
  $\pi_\theta \leftarrow$ `update_policy`$(\pi_\theta, best\_episodes\_and\_rewards)$
end for

3.2.4 Brief summary of RBG2-SR method

To summarize, our proposed RBG2-SR method performs constrained SR where a grammatical structure restricts symbolic expressions creation. We adopt a POMDP setting with a finite horizon H, for which the associated reward r_h is zero until the full expression f has been generated. A grammar defines a set of constraint rules used for the expression construction, which masks the non-accessible actions at each step $h \in [H]$. The weights of the actions accessible at the current time step are generated by π_θ, a neural network learned with the REINFORCE algorithm, using a risk-seeking with entropy term loss.

4 Experiments

In this section, we describe two experiments. The first one compares the proposed method with several state-of-the-art (Whigham et al, 1995; Sotto and de Melo, 2017) SR solutions on reference benchmarks (Uy et al, 2011; Keijzer, 2003; Vladislavleva et al, 2009; Pagie and Hogeweg, 1997). In the second experiment, we present a feature exploration use case where we show an application of our approach on a real-world dataset with an unknown relationship to uncover. Both data and code for this benchmark are freely available on Github [1].

4.1 Experiment 1: Benchmark evaluation

4.1.1 Benchmarked methods and datasets

To evaluate our method and compare it to other state-of-the-art works, we consider 34 functions gathered from the Nguyen (Uy et al, 2011) (noted N1 to N10), Keijzer (Keijzer, 2003) (K1-15), Vladislavleva (Vladislavleva et al, 2009)

[1] https://github.com/laure-crochepierre/reinforcement-based-grammar-guided-symbolic-regression

(V1-8), and Pagie (Pagie and Hogeweg, 1997) benchmark suites with varying levels of difficulty. The Nguyen benchmark suite is known to be the easiest one because it is mainly using one input feature and does not require optimizing for constant values in the expressions. Keijzer and Vladislavleva benchmarks are more complex as they contain functions with up to 5 inputs variables and also require to represent scaling constants. We also used Pagie (P1) (Pagie and Hogeweg, 1997) function, which has the reputation of being more challenging (McDermott et al, 2012). We applied several guidelines identified for Symbolic Regression benchmarking as provided by McDermott et al (2012). Our data generation procedure uses their functions and sampling intervals for train and test sets (McDermott et al, 2012).

The selected symbolic functions are benchmarked against the following grammar based methods:

Grammar Guided Genetic Programming (G3P) (Whigham et al, 1995) is a grammar guided genetic algorithm build using the Deap library. (Fortin et al, 2012)

Probabilistic Model Building Genetic Programming (GB-LGP) (Sotto and de Melo, 2017) updates the probability distribution of a grammar according to selected individual from an evolutionary population in gradient-descent like algorithm.

As these methods are not directly available with our expression representation, we have re-implemented both methods. The following code is available on our Github repository.

```
<e> ::= (<e><dop><e>) | (<etw1><dopw1><etw1>) | <sop>(<e>) | <et> || probs [1/4,1/4,1/4,1/4]
<et> ::= (- x[x.columns<varidx>]]) | x[<varidx>] || probs [0.5, 0.5]
<etw1> ::= ( - x[<varidx>]) |  x[<varidx>] | 1 || probs [1/3,1/3,1/3]
<dopw1> ::=  + |  - || probs [1/2, 1/2]
<dop> ::= + | - | * | / || probs [1/4, 1/4, 1/4, 1/4]
<sop> ::= cos | sin | exp | log || probs [0.25, 0.25, 0.25, 0.25]
<varidx> ::= 1... nvar || probs [1/nvar ... 1/nvar]
```

Fig. 4 Grammar example inspired by (Sotto and de Melo, 2017). <e> is the start symbol, T = {<e>, <et>, <etw1>, ..., <varidx>} NT={x[], +,-,*,/,cos,sin, exp, log,1,..,nvar}

To have comparable results between these methods, we propose to use the grammar described in Figure 4 for G3P, GB-LGP, and RBG2-SR (ours). It describes the transitions between 7 symbols and defines the action space to search into, made of at least $19+n_{var}$ actions (with n_{var} the number of features in the dataset). Datasets are generated using the drawing process detailed in Appendix A.

After hyperparameters search using a grid search method for all three algorithms, we choose the following best perfoming parameters for RBG2-SR: $\lambda_H = 0.005$ and a learning rate $\alpha = 0.001$. All methods are compared on a maximal horizon of 50 actions and each run is performed on a population/batch of 1000 expressions with a total of 2 millions expressions tested at most (corresponding to a 2000 iterations: batch size $\times$ nb training steps$= 1000 \times 2000 = 2M$).

4.1.2 Benchmark results

Expressions found by these methods on 30 independent runs are compared using Mean Squared Error (MSE) averages and standard deviations between the exact expression to uncover and best in-run solution of each algorithm. The last row of the table corresponds to the result of the Mann–Whitney U test for independent samples (Mann and Whitney, 1947). U-test results are summarised by counting the number of times each method performs better (symbol +), equivalently (∼), or worse (−) than other methods. "Equivalently" refers to the case where two (or more) methods are equally good, and have the same best results. A method is said to perform "worse" if at least one of the two other methods is performing "better" or "equivalently". Results for these benchmarks are shown in Table 1 with best results per function in bold and equivalent scores in italic.

Table 1 Mean Squared Error and Standard Deviation scores for benchmarked methods, averaged over 30 runs (best results in bold). The symbol − is used when unable to compute a solution or when the solution error is larger than 10^{10}. On the last two rows, is shown first the average MSE across all valid runs (out of 30 runs), and all benchmarks and then the count of times where each method is performing better (symbol +), equivalently (∼) or worse (−) than others performed using the Mann–Whitney U test

Name	GB-LGP (Sotto and de Melo, 2017)	G3P (Whigham et al, 1995)	RBG2-SR (Ours)
N1	5.71×10^{-2} ($\pm 7.6 \times 10^{-2}$)	2.35×10^{-3} ($\pm 2.6 \times 10^{-3}$)	**0.00 (±0.0)**
N2	9.29×10^{-2} ($\pm 1.7 \times 10^{-1}$)	2.19×10^{-2} ($\pm 5.8 \times 10^{-2}$)	**0.00 (±0.0)**
N3	2.24×10^{-1} ($\pm 2.7 \times 10^{-1}$)	1.82×10^{-2} ($\pm 2.4 \times 10^{-2}$)	**0.00 (±0.0)**
N4	2.24×10^{-1} ($\pm 4.1 \times 10^{-1}$)	***1.48 × 10⁻²*** ***(±1.6 × 10⁻²)***	*1.62 × 10⁻²* *(±1.6 × 10⁻²)*
N5	6.16×10^{-3} ($\pm 1.2 \times 10^{-2}$)	1.31×10^{-3} ($\pm 1.9 \times 10^{-3}$)	$\mathbf{5.87 \times 10^{-4}}$ **(±8.6 × 10⁻⁴)**
N6	1.92×10^{-2} ($\pm 1.4 \times 10^{-2}$)	1.70×10^{-3} ($\pm 1.4 \times 10^{-3}$)	$\mathbf{2.78 \times 10^{-4}}$ **(±7.9 × 10⁻⁴)**
N7	1.48×10^{-2} ($\pm 1.5 \times 10^{-2}$)	***3.97 × 10⁻⁴*** ***(±3.0 × 10⁻⁴)***	*3.30 × 10⁻⁴* *(±3.1 × 10⁻⁴)*
N8	2.11×10^{-2} ($\pm 4.1 \times 10^{-2}$)	5.64×10^{-2} ($\pm 2.9 \times 10^{-1}$)	$\mathbf{1.24 \times 10^{-4}}$ **(±6.6 × 10⁻⁴)**
N9	1.41×10^{-1} ($\pm 1.2 \times 10^{-1}$)	***3.93 × 10⁻²*** ***(±1.1 × 10⁻¹)***	*1.66 × 10⁻²* *(±2.3 × 10⁻²)*
N10	1.81×10^{-2} ($\pm 3.1 \times 10^{-2}$)	6.67×10^{-3} ($\pm 1.2 \times 10^{-2}$)	$\mathbf{1.38 \times 10^{-3}}$ **(±1.5 × 10⁻³)**
K1	3.38×10^{-2} ($\pm 5.2 \times 10^{-3}$)	1.11×10^{-2} ($\pm 9.7 \times 10^{-3}$)	$\mathbf{2.35 \times 10^{-3}}$ **(±3.0 × 10⁻³)**
K2	4.48×10^{-2} ($\pm 1.1 \times 10^{-3}$)	3.72×10^{-2} ($\pm 4.5 \times 10^{-3}$)	$\mathbf{2.28 \times 10^{-2}}$ **(±7.4 × 10⁻³)**
K3	4.50×10^{-2} ($\pm 1.1 \times 10^{-4}$)	4.03×10^{-2} ($\pm 6.5 \times 10^{-3}$)	$\mathbf{3.15 \times 10^{-2}}$ **(±7.7 × 10⁻³)**
K4	8.71×10^{-2} ($\pm 2.0 \times 10^{-2}$)	2.44×10^{-2} ($\pm 1.7 \times 10^{-2}$)	$\mathbf{1.19 \times 10^{-2}}$ **(±8.9 × 10⁻³)**
K5	−	−	−
K6	2.56×10^{-2} ($\pm 7.8 \times 10^{-2}$)	$\mathbf{1.46 \times 10^{-3}}$ **(±2.0 × 10⁻³)**	2.32×10^{-3} ($\pm 1.9 \times 10^{-3}$)
K7	***4.70 × 10⁻⁴*** ***(±1.8 × 10⁻³)***	6.59×10^{-7} ($\pm 3.0 \times 10^{-6}$)	*0.00* *(±0.0)*
K8	5.05×10^{-1} (± 1.0)	1.94×10^{-1} ($\pm 2.1 \times 10^{-1}$)	$\mathbf{2.92 \times 10^{-2}}$ **(±8.9 × 10⁻²)**
K9	1.03×10^{-3} ($\pm 3.8 \times 10^{-3}$)	$\mathbf{3.11 \times 10^{-6}}$ **(±4.3 × 10⁻⁶)**	4.08×10^{-6} ($\pm 3.4 \times 10^{-6}$)
K10	2.45×10^{-3} ($\pm 2.3 \times 10^{-3}$)	6.94×10^{-4} ($\pm 7.5 \times 10^{-4}$)	$\mathbf{1.79 \times 10^{-4}}$ **(±2.0 × 10⁻⁴)**
K11	8.49×10^{-1} (± 1.9)	$\mathbf{5.52 \times 10^{-1}}$ **(±2.8 × 10⁻¹)**	2.42 (± 9.5)
K12	$3.40 \times 10^{+2}$ ($\pm 4.4 \times 10^{+2}$)	$6.33 \times 10^{+4}$ ($\pm 3.5 \times 10^{+5}$)	**2.36 (±1.2)**
K13	−	3.04 (± 3.6)	$\mathbf{5.23 \times 10^{-1}}$ **(±1.2)**
K14	5.65×10^{-1} ($\pm 7.5 \times 10^{-2}$)	4.21×10^{-1} ($\pm 1.9 \times 10^{-1}$)	$\mathbf{1.49 \times 10^{-1}}$ **(±1.7 × 10⁻¹)**
K15	2.41 (± 1.5)	$2.03 \times 10^{+8}$ ($\pm 1.1 \times 10^{+9}$)	$\mathbf{9.22 \times 10^{-1}}$ **(±1.6 × 10⁻¹)**
P1	2.14×10^{-1} ($\pm 2.5 \times 10^{-1}$)	***1.66 × 10⁻¹*** ***(±1.2 × 10⁻¹)***	*1.11 × 10⁻¹* *(±9.2 × 10⁻²)*
V1	***5.74 × 10⁻²*** ***(±2.7 × 10⁻²)***	−	*5.13 × 10⁻²* *(±1.7 × 10⁻²)*
V2	$\mathbf{8.39 \times 10^{-2}}$ **(±1.9 × 10⁻²)**	−	1.47×10^{-1} ($\pm 6.4 \times 10^{-1}$)
V3	−	4.52 ($\pm 1.7 \times 10^{+1}$)	$\mathbf{8.31 \times 10^{-1}}$ **(±2.9 × 10⁻¹)**
V4	***3.83 × 10⁻²*** ***(±3.6 × 10⁻³)***	***3.87 × 10⁻²*** ***(±4.7 × 10⁻³)***	*3.71 × 10⁻²* *(±5.3 × 10⁻³)*
V5	2.77×10^{-1} ($\pm 1.2 \times 10^{-1}$)	1.54×10^{-1} ($\pm 9.1 \times 10^{-2}$)	$\mathbf{4.10 \times 10^{-2}}$ **(±3.3 × 10⁻²)**
V6	4.76 (± 5.3)	−	$\mathbf{8.64 \times 10^{-1}}$ **(±7.6 × 10⁻¹)**
V7	$2.60 \times 10^{+1}$ ($\pm 7.6 \times 10^{+1}$)	$1.13 \times 10^{+1}$ (± 3.8)	**9.94 (±1.0)**
V8	4.12 ($\pm 2.9 \times 10^{-1}$)	3.93 (± 1.9)	**2.06 (±5.7 × 10⁻¹)**
Avg MSE	$1.14 \times 10^{+1}$	1.70	$\mathbf{6.09 \times 10^{-1}}$
U test	+1 / ∼ 3 / − 29	+3/ ∼ 5/ − 25	**+22/ ∼ 7/ − 4**

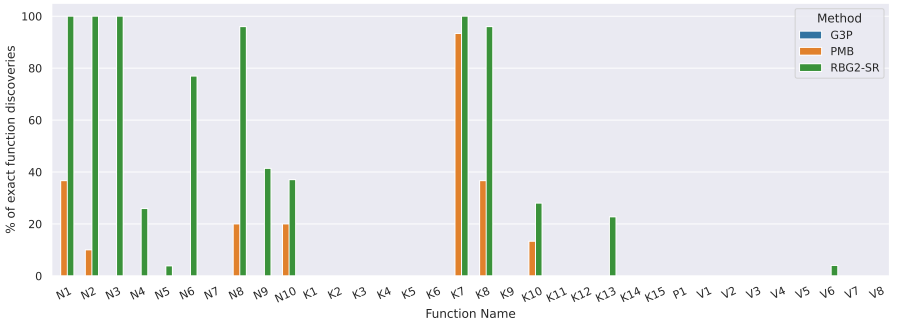

Fig. 5 Percentage of exact solutions found on the four benchmarks for the three methods RBG2-SR (blue), GB-LGP (orange), G3P (green). Better seen in color. 100% means that all 30 runs uncover the solution, 50% that 15 out of the 30 were able to match the right results

As shown on the antepenultimate row in Table 1, our method lower the average MSE by one order of magnitude and performs statistically better than other methods on 22 out of 33 benchmarks according to the Mann–Whitney U test. The K5 benchmark is not taken into account in testing because all methods perform poorly and produce a high MSE error on this benchmark (above 10^{14}). We also note that we perform similarly to other methods for seven benchmarks, leading to an *at least* similar error in more than 90% of the tested benchmarks (30 out of 33 benchmarks).

More precisely, looking at the Nguyen benchmark, we show that our proposed method outperforms other methods for all functions, except for N4, N6, and N9, where the error is similar between RBG2-SR and G3P. Then, for Keijzer and Vladislavleva benchmarks, our method proposes solutions with a lower (resp. equivalent) error for 10 out of 15 (resp. 1/15) and 5 out of 8 (resp. 3/8) benchmarks. We also highlight that our method seems to complement the evolutionary G3P method, especially on the Keijzer benchmark, as G3P has a lower error on the few benchmarks where our method is weaker.

Let us now analyse the ability of our algorithm to precisely retrieve the exact target symbolic expression. It is also worth noting that our method reaches a zero error for several benchmarks (N1-3 and K7), meaning that we recover all the time the exact solution. More details on this result is described in Figure 5. In this Figure, we represent for each benchmark and method the number of times (in percentage) where we can recover the exact solution up to the numerical precision. First, we see that two of the three methods recover functions mostly on the Nguyen and Keijzer benchmarks. As we detailed above, unlike other methods for most benchmarks, the RBG2-SR method can often recover the exact solution for 9 out of 10 expressions of the Nguyen benchmark. On the same benchmark, G3P only recovers approximate solutions, and GB-LGP can only find four functions with a lower percentage. While GB-LGP recovers 3 out of 15 expressions from the Keijzer benchmark, our RBG2-SR method recovers 4 functions with a higher recovery rate. RGB2-SR also recover sometimes one function on the Vladislavleva benchmark. Note that these

results are tied to the grammar used. The GB-LGP and G3P methods could potentially find more exact solutions by choosing another grammar. Moreover, among other unfound functions, some are difficult or even impossible to construct with the chosen grammar. This is especially true for functions such as K1-3 or V1 that require finding decimal constants, which is not supported in this version of the grammar.

4.1.3 Ablation study

In order to identify which elements of the RBG2-SR method are essential to the success of the expression search, we performed an ablation study on elements of the state definition s_h and on the algorithm. In the algorithm itself, we try removing the risk-seeking objective J_θ^{risk} (keeping all trajectories) and the entropy loss term $J_\theta^{entropy}$. Regarding the state definition, we compare state defined with and without: the current symbol σ_h, the current mask m_h, the current depth d_h, the parent node a_h^{parent}, the siblings nodes $a_h^{siblings}$ and the previously selected actions a_h^{past}. This ablation study uses the ten functions of the Nguyen benchmark and was run 10 times for each ablation/function combination. All results are compared to the *baseline* case where no element is occluded. The results are summarized in Table 2.

Table 2 Ablation Study. Averaged MSE scores (± Standard Deviation) and percentage of variation over 10 runs on the Nguyen benchmark. All results are to be compared to the baseline case: $Variation(\%) = 100\frac{baseline-ablation}{baseline}$

Type	Ablation	MSE	Variation(%)
	Baseline	1.58×10^{-3} ($\pm 4.92 \times 10^{-3}$)	-
Algorithm	No entropy	1.51×10^{-2} ($\pm 4.41 \times 10^{-2}$)	860%
	No risk-seeking	5.16×10^{-2} ($\pm 8.68 \times 10^{-2}$)	3200%
State	No parent	1.51×10^{-3} ($\pm 4.68 \times 10^{-3}$)	−4%
	No siblings	2.57×10^{-3} ($\pm 8.75 \times 10^{-3}$)	63%
	No past actions	2.97×10^{-3} ($\pm 1.01 \times 10^{-2}$)	87%
	No depth	3.73×10^{-3} ($\pm 1.18 \times 10^{-2}$)	140%
	No symbol	3.83×10^{-3} ($\pm 1.23 \times 10^{-2}$)	140%

First, when looking at the algorithm learning itself, we show in Table 2 that the combination of the two loss terms is relevant to our problem. The MSE significantly increases when removing one of these terms. The risk-seeking policy is crucial to this type of learning: removing the risk-seeking policy increases the error by 3200% when compared to the proposed method with risk-seeking. The entropy term is also of great importance, with an error increase of 860%.

We performed a second type of ablation on the state definition. In this part of the ablation study, we tried removing elements from *state* inputs from the Neural Network (the orange block called "state information"): either parent action information a^{parent}, siblings action information $a^{siblings}$, previously selected actions a^{past}. Results from Table 2 indicate that these siblings, past actions, depth, and symbols are highly beneficial to the expression search since

removing one of these terms increases the error by at least 63%. Depth and symbol information seems to be of almost equal importance for a successful expression search. Regarding parent information, it seems that removing this information tends to reduce the error by a small 4%. However, by looking more carefully at each benchmark, removing the parent information is only beneficial to the search for benchmark N9. Except for N1-3 and N7 (where both configurations always recover the exact expression), the complete proposed algorithm (*Baseline*) outperforms the parent ablation. For the remaining benchmarks, the corresponding increase lies between +50 and +55%. With these results, we choose to keep parent information in the state definition.

4.2 Experiment 2: Interpretability analysis of a use case

This work aims at describing an algorithm that provides interpretable symbolic solutions directly readable by humans. From the first experiment, we also see a potential application of our approach to more complex datasets with an unknown relationship between a set of observations X and a target variable y, where X and y variables may be of different physical units. In this scenario, the use of a grammar is particularly important to restrict outputs to realistic solutions in terms of physical units. For example it is not physically feasible to add a speed (measured in meters per second) with a distance (in meters).

Toward this physical interpretability goal, we designed a second experiment on the Airfoil Self-Noise dataset (Brooks et al, 1989), to compare the solutions proposed by our algorithm with the ones given by four state-of-the-art (Whigham et al, 1995; Sotto and de Melo, 2017; McConaghy, 2011; Petersen et al, 2021) methods. The dataset is accessible on the UCI Machine Learning Repository[2]. The tested methods are GB-LGP, G3P from the previous experiment, and two other non-grammar based methods:

Fast Function Extraction (FFX) (McConaghy, 2011) which applies pathwise learning to a large set of nonlinear functions, and exploits the path structure to generate models that trade-off error/complexity.

Deep Symbolic Regression (DSR from (Petersen et al, 2021)) a deep learning algorithm which uses a RL approach to search the solution space without grammatical constraints.

4.2.1 Dataset Description

The problem we focus on in this experiment is predicting scaled sound pressure level (SSPL) on different size NACA 0012 airfoils. The estimation is made using the following variables: frequency (unit Hz), angle of attack (degree °), chord length (meters m), free-stream velocity (meters per second $m.s^{-1}$), suction side displacement thickness (meters m).The measurements are obtained using airfoils taken at various wind tunnel speeds and angles of attack. The span of the airfoil and the observer position is fixed for all measurements. The dataset is split between 70% in the train set and 30% in the test set.

[2] https://archive.ics.uci.edu/ml/datasets/airfoil+self-noise (Accessed February 8, 2022)

4.2.2 Grammar construction

The first preprocessing step, before methods comparison, is the definition of a constrained grammar that contains premice of knowledge on the studied topic, such as the one used in Figure 6. The start symbol is <exp>. This symbol describes the dimensions (units) and structures the algorithm allows as a return. From the previous studies (Lau et al, 2009) on this data, we want to find an expression in the form: `exp` = `constant` − 10 ∗ `log10`(`child_expression`). We also add several other structures to leave to the algorithm the freedom to explore.

```
<exp>      ::=  <unit> * const  | <no_unit> * const
           | const-10*log10(<no_unit>/const)*<no_unit>
           | const-10*log10(<unit>/const)*<no_unit>
           ||probs [0.25, 0.25, 0.25, 0.25]
<unit>     ::= <distance> | <velocity> | <time> |(<no_unit> * <unit>)
           || probs [0.25,0.25,0.25,0.25]
<no_unit>  ::= <no_unit>*<no_unit>| cos(x.alpha) | sin(x.alpha) | <distance>/<distance>
            | <velocity>/<velocity> | <time>/<time> ||  probs [0.16,0.16,...,0.16]
<velocity> ::= (<velocity><dop><velocity>) | (<distance>/<time>) | x.U_infinity
            || probs [0.33,0.33,0.33]
<distance> ::= (<distance><dop><distance>) | (<velocity>*<time>) | abs(<distance>)
            | x.delta | x.c || probs [0.2,...,0.2]
<time>     ::=(<distance>/<velocity>)| (1/x.f) || probs [0.5,0.5]
<dop>      ::= - | + || probs [0.5,0.5]
```

Fig. 6 Grammar used in the second experiment on the Airfoil dataset. The start symbol is <exp>

To sum up the grammar from Figure 6, the three first lines describe what the units and non-units (composition of units) of the problem are. In the four final lines, the grammar constrains the operations on each dimension (or unit) to only physically consistent combinations. These lines also define how to go from one unit to another by using physical properties (such as velocity law). This part of the grammatical description is of particular importance to describe the expertise and knowledge we want to include to constrain the search space during the SR resolution.

4.2.3 Experiments and results

In this experiment, all algorithms are run 10 times (except for FFX (McConaghy, 2011) as it is deterministic), and the best expression of all runs is shown in Table 3. Their results are compared on the test set based on the MSE error, determination coefficient $\mathcal{R}^2$ and complexity $\mathcal{C}$. The complexity $\mathcal{C}$ is defined by the sum of operations and input features used in the formula. From Table 3, we can note that the best performing algorithm on this dataset is FFX, with an error of 10.3. However, this method also produces the expression with the highest complexity (around 140), above the maximal horizon H of 50. As seen in the column *Expression*, FFX solution is complex, not directly readable by a human and combine variables with different units. Among other four methods, we highlight that our proposed RBG2-SR method produces the second lowest error and highest determination coefficient of 0.72, while keeping

Table 3 Analysis of the Airfoil Self-Noise benchmark. Best expression found are presented along with their MSE, determination coefficient $\mathcal{R}^2$, and complexity $\mathcal{C}$ scores

Method	Expression	MSE	$\mathcal{R}^2$	$\mathcal{C}$	$\mathcal{C} - H$ (< 0)
DSR	$-\alpha + U_{infinity} - \frac{U_{infinity}}{\sin(\log_{10}(U_{infinity}))}$	239.15	−4.07	10	−40 (<<)
FFX	$0.001U_{infinity}^2 - 0.026U_{infinity} + 14.4\alpha\delta$ $-0.492\alpha + 12.8c^2 + \ldots + \log_{10}(f)$ $+18.8\log_{10}(\delta) - 71.3\log_{10}(f) + 272$	10.3	**0.78**	140	90 (>>)
G3P	$101.38 - 10\log_{10}\left(\frac{1}{f} + \frac{\delta f^2 c^2}{U_{infinity}^3}\right)$	19.5	0.58	22	−28 (<)
GB-LGP	$127.36 - 10\log_{10}\left(\frac{c\delta f}{\frac{U_{infinity}{}^{\delta}}{c} + 2U_{infinity}}\right)\cos^2(\alpha)$	35.6	0.24	24	−26 (<)
RBG2-SR (ours)	$\mathbf{83.85 - \frac{10\log_{10}\left(\frac{U_{infinity}}{cf}\right) + \frac{U_{infinity}^2}{f^2 c\delta}}{1 + \frac{c\delta f^2}{U_{infinity}^2}}}$	13.0	0.72	**32**	**−18 (<)**

an acceptable complexity of 36, close to the threshold H but lower. Moreover, it is worth noticing that all grammar-based methods used a expression which uses the format : `constant` $-$ `10` $*$ `log10(child_exp)`. For example, the DSR method finds a simple solution, largely under the threshold H. However, this solution is too simplistic and doesn't respect dimensional consistency. These results tend to advocate for the usage of grammatical constraints for equation discovery. The expression found by our method, could for example be used to estimate the value of the `constant` in the above-mentioned equation.

Eventually, regarding unit or dimensional consistency, all non-grammar based methods constructed forbidden combinations of different units, showing that their are not yet able to compete with grammar-based methods to build physically-relevant expressions.

5 Conclusion and perspectives

This study proposes a new algorithm (RBG2-SR) for Grammar Guided Symbolic Regression using a Reinforcement Learning search approach that allows the inclusion of domain knowledge within the learning process and in the format of the solutions. We describe a POMDP modeling of the SR task in a grammatical action space. The proposed method is benchmarked against grammar-based state-of-the-art algorithms and shows significant improvements over other algorithms regarding the error metric and exact expression discovery. We performed an ablation study of the blocks of our algorithm and state definition. The results show that parent, sibling, past actions, depth and symbol informations are all important elements of the state definition. In the second experiment, we also show how the use of a grammar based approach could be useful and interpretable when working on a dataset with physical constraint between input features.

From the obtained results, we also foresee different perspectives to this work. First, when comparing G3P and RGB2-SR results, we envision improvements by doing cross-learning (Zhang and Zhou, 2021) between G3P to encourage exploration and our method for learning and sampling. Moreover, as the grammar construction process can be a time-consuming task, we could draw inspiration from techniques that automatically build ontologies (Emani et al, 2019) to automatically create and improve grammar. We also foresee application perspectives for our method to find interpretable policies that follow expert grammatical rules. From the second experiment, we also want to explore the behavior of our algorithm when dealing either with longer horizons up to 100 actions to create more expressive expressions or with shorter horizons for more concise and interpretable expressions.

Acknowledgments. This work was supported by the French Association Nationale de la Recherche et de la Technologie (ANRT) grant number 2018/1466. We also thank Benjamin Donnot for helpful discussions, Remy Clément and Baltazar Donon for their comments and suggestions.

Appendix A Benchmark generation information

SR benchmarks used in this paper: Nguyen (Uy et al, 2011), Keijzer (Keijzer, 2003), Vladislavleva (Vladislavleva et al, 2009), and Pagie (Pagie and Hogeweg, 1997) (respectilvely noted N1-10, K1-15, V1-8 and P1). Variables (column *Vars*) are x, y, z, v, w and their corresponding representation in the grammar is $x[1]$ to $x[5]$. $U[a, b, c]$ is a uniform sampling of c samples between a to b. $E[a, b, c]$ samples in a grid of evenly spaced points with an interval of c, from a to b. Table 4 is an extended version of the generation information presented by McDermott et al (2012).

Declarations

Funding. This work was supported by the French Association Nationale de la Recherche et de la Technologie (ANRT) [CIFRE convention between Université de Lorraine and Rte, grant number 2018/1466].

Conflict of interest/Competing interests. The authors declare that they have no conflict of interest.

Ethics approval. Not applicable

Consent to participate. Not applicable

Consent for publication. Not applicable

Availability of data and materials. The experiment benchmarks are generated following the procedure detailed in Appendix A and data used in the second experiment can be found on the UCI Machine Learning Repository. Data generation code is available on our Github repository.

Table 4 Nguyen (respectilvely noted N1-10), Keijzer (respectilvely noted K1-15) Vladislavleva and Pagie (respectilvely noted V1-8 and P1) benchmarks

Name	Function	Vars	Train set	Test Set
N1	$x^3 + x^2 + x$	1	$U[0, 2, 20]$	$U[0, 2, 20]$
N2	$x^4 + x^3 + x^2 + x$	1	$U[-1, 1, 20]$	$U[-1, 1, 20]$
N3	$x^5 + x^4 + x^3 + x^2 + x$	1	$U[-1, 1, 20]$	$U[-1, 1, 20]$
N4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	1	$U[-1, 1, 20]$	$U[-1, 1, 20]$
N5	$sin(x^2)cos(x) - 1$	1	$U[-1, 1, 20]$	$U[-1, 1, 20]$
N6	$sin(x) + sin(x + x^2)$	1	$U[-1, 1, 20]$	$U[-1, 1, 20]$
N7	$ln(x + 1) + ln(x^2 + 1)$	1	$U[0, 2, 20]$	$U[0, 2, 20]$
N8	$\sqrt{(x)}$	1	$U[0, 4, 20]$	$U[0, 4, 20]$
N9	$sin(x) + sin(y)$	2	$U[0, 2, 100]$	$U[0, 2, 100]$
N10	$2sin(x)cos(y)$	2	$U[0, 2, 100]$	$U[0, 2, 100]$
K1	$0.3xsin(2\pi x)$	1	$E[-1, 1, 0.1]$	$E[-1, 1, 0.001]$
K2	$0.3xsin(2\pi x)$	1	$E[-2, 2, 0.1]$	$E[-2, 2, 0.001]$
K3	$0.3xsin(2\pi x)$	1	$E[-3, 3, 0.1]$	$E[-3, 3, 0.001]$
K4	$x^3e^{-x}cos(x)sin(x)(sin^2(x)cos(x) - 1)$	1	$E[0, 10, 0.05]$	$E[0.05, 10.05, 0.05]$
K5	$\frac{30xz}{(x-10)y^2}$	3	$x, z : U[-1, 1, 1000]$ $y : U[1, 2, 1000]$	$x, z : U[-1, 1, 10000]$ $y : U[1, 2, 10000]$
K6	$\sum_i^x \frac{1}{x}$	1	$E[1, 50, 1]$	$E[1, 120, 1]$
K7	$ln(x)$	1	$E[1, 100, 1]$	$E[1, 100, 0.1]$
K8	$\sqrt{(x)}$	1	$E[0, 100, 1]$	$E[0, 100, 0.1]$
K9	$arcsinh(x)$	1	$E[0, 100, 1]$	$E[0, 100, 0.1]$
K10	x^y	2	$U[0, 1, 100]$	$E[0, 1, 0.01]$
K11	$xy + sin((x - 1)(y - 1))$	2	$U[-3, 3, 20]$	$E[0, 1, 0.01]$
K12	$x^4 - x^3 + \frac{y^2}{2} - y$	2	$U[-3, 3, 20]$	$E[0, 1, 0.01]$
K13	$6sin(x)cos(y)$	2	$U[-3, 3, 20]$	$E[0, 1, 0.01]$
K14	$\frac{8}{2+x^2+y^2}$	2	$U[-3, 3, 20]$	$E[0, 1, 0.01]$
K15	$\frac{x^3}{5} + \frac{y^3}{2} - y - x$	2	$U[-3, 3, 20]$	$E[0, 1, 0.01]$
V1	$\frac{e^{-(x-1)^2}}{1.2+(y-2.5)^2}$	2	$U[0.3, 4, 100]$	$E[-0.2, 4.2, 0.1]$
V2	$e^{-x}x^3cos(x)sin(x)(sin^2(x)cos(x) - 1)$	1	$E[0.05, 10, 0.1]$	$E[-0.5, 10.5, 0.05]$
V3	$e^{-x}x^3cos(x)sin(x)(sin^2(x)cos(x) - 1)(y - 5)$	2	$x : E[0.05, 10, 0.1]$ $y : E[0.05, 10.05, 2]$	$x : E[0.05, 10, 0.1]$ $y : E[-0.5, 10.5, 0.5]$
V4	$\frac{10}{5+(x-3)^2+(y-3)^2+(z-3)^2+(v-3)^2+(w-3)^2}$	5	$U[0.05, 6.05, 1024]$	$U[-0.25, 6.35, 5000]$
V5	$30\frac{(x-1)(z-1)}{y^2(x-10}$	3	$x : U[0.05, 2, 300]$ $y : U[1, 2, 300]$	$x : E[-0.05, 2.1, 0.15]$ $y : E[0.95, 2.05, 0.1]$
V6	$6sin(x)cos(y)$	2	$U[0.1, 5.9, 30]$	$E[-0.05, 6.05, 0.02]$
V7	$(x - 3)(y - 3) + 2sin((x - 4)(y - 4))$	2	$U[0.05, 6.05, 300]$	$U[-0.25, 6.35, 1000]$
V8	$\frac{(x-3)^4+(y-3)^3-(y-3)}{(y-2)^4+10}$	2	$U[0.05, 6.05, 50]$	$E[-0.25, 6.35, 0.2]$
P1	$\frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$	2	$E[-5, 5, 0.4]$	$E[-5, 5, 0.4]$

Code availability. The code is freely available on the Github repository whose link is shared in this document.

Authors' contributions. All authors contributed to the study conceptualization, design, and investigation. Data collection and analysis were performed by Laure Crochepierre. The first draft of the manuscript was written by Laure Crochepierre and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

References

Alshiekh M, Bloem R, Ehlers R, et al (2018) Safe reinforcement learning via shielding. In: Thirty-Second AAAI Conference on Artificial Intelligence

Anjum A, Sun F, Wang L, et al (2019) A novel neural network-based symbolic regression method: Neuro-encoded expression programming. In: International Conference on Artificial Neural Networks, Springer, pp 373–386

Bertsekas DP (2019) Feature-based aggregation and deep reinforcement learning: a survey and some new implementations. IEEE/CAA Journal of Automatica Sinica 6(1):1–31

Brence J, Todorovski L, Džeroski S (2021) Probabilistic grammars for equation discovery. Knowledge-Based Systems 224:107,077

Brooks TF, Pope DS, Marcolini MA (1989) Airfoil self-noise and prediction

Cherrier N, Poli J, Defurne M, et al (2019a) Consistent feature construction with constrained genetic programming for experimental physics. In: IEEE Congress on Evolutionary Computation, CEC, pp 1650–1658

Cherrier N, Poli JP, Defurne M, et al (2019b) Consistent feature construction with constrained genetic programming for experimental physics. In: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp 1650–1658

Chung W, Thomas V, Machado MC, et al (2021) Beyond variance reduction: Understanding the true impact of baselines on policy optimization. In: Meila M, Zhang T (eds) Proceedings of the 38th International Conference on Machine Learning, PMLR, pp 1999–2009

Cremers A, Ginsburg S (1975) Context-free grammar forms. Journal of Computer and System Sciences 11(1):86–117

Crochepierre L, Boudjeloud-Assala L, Barbesant V (2021) Interpretable dimensionally-consistent feature extraction from electrical network sensors. In: Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track, pp 444–460

Ebner M (1999) On the search space of genetic programming and its relation to nature's search space. In: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99, pp 1357–1361 Vol. 2

Emani CK, Silva CFD, Fiés B, et al (2019) NALDO: from natural language definitions to OWL expressions. Data Knowl Eng 122:130–141

Fortin FA, De Rainville FM, Gardner MA, et al (2012) DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research 13:2171–2175

Hein D, Udluft S, Runkler TA (2018) Interpretable policies for reinforcement learning by genetic programming. Engineering Applications of Artificial Intelligence 76:158–169

Heng K, Morris BM, Kitzmann D (2021) Closed-form ab initio solutions of geometric albedos and reflected light phase curves of exoplanets. Nature Astronomy 5(10):1001–1008

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural computation 9(8):1735–1780

Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural networks 2(5):359–366

Huang S, Ontañón S (2020) A closer look at invalid action masking in policy gradient algorithms. arXiv preprint arXiv:200614171

Jin Y, Fu W, Kang J, et al (2019) Bayesian symbolic regression. arXiv preprint arXiv:191008892

Kaelbling LP, Littman ML, Cassandra AR (1998) Planning and acting in partially observable stochastic domains. Artificial Intelligence 101(1):99–134

Keijzer M (2003) Improving symbolic regression with interval arithmetic and linear scaling. In: European Conference on Genetic Programming, Springer, pp 70–82

Keijzer M, Babovic V (1999) Dimensionally aware genetic programming. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2, p 1069–1076

Khurana U, Samulowitz H, Turaga D (2018) Feature engineering for predictive modeling using reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence

Kim S, Lu PY, Mukherjee S, et al (2020) Integration of neural network-based symbolic regression in deep learning for scientific discovery. IEEE Transactions on Neural Networks and Learning Systems

Knuth DE (1964) backus normal form vs. backus naur form. Commun ACM 7(12):735–736

Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms. In: Advances in neural information processing systems, pp 1008–1014

Koza JR (1990) Concept formation and decision tree induction using the genetic programming paradigm. In: Parallel Problem Solving from Nature, 1st Workshop, PPSN I, Dortmund, Germany, Proceedings, pp 124–128

Koza JR (1992) Hierarchical automatic function definition in genetic programming. In: Proceedings of the Second Workshop on Foundations of Genetic Algorithms., pp 297–318

Koza JR, Keane MA, Streeter MJ, et al (2006) Genetic programming IV: Routine human-competitive machine intelligence, vol 5

Kusner MJ, Paige B, Hernández-Lobato JM (2017) Grammar variational autoencoder. In: International Conference on Machine Learning, PMLR, pp 1945–1954

Landajuela M, Petersen BK, Kim S, et al (2021) Discovering symbolic policies with deep reinforcement learning. In: International Conference on Machine Learning, PMLR, pp 5979–5989

Lau K, Lopez R, Navarra EOI (2009) A neural networks approach to aerofoil noise prediction

Lucena-Sánchez E, Sciavicco G, Stan IE (2021) Feature and language selection in temporal symbolic regression for interpretable air quality modelling. Algorithms 14(3)

Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. Annals of Mathematical Statistics 18:50–60

McConaghy T (2011) FFX: Fast, Scalable, Deterministic Symbolic Regression Technology, pp 235–260

McDermott J, White DR, Luke S, et al (2012) Genetic programming needs better benchmarks. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, p 791–798

McKay RI, Hoai NX, Whigham PA, et al (2010) Grammar-based genetic programming: a survey. Genetic Programming and Evolvable Machines 11(3):365–396

Mnih V, Kavukcuoglu K, Silver D, et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533

Montana DJ (1995) Strongly typed genetic programming. Evolutionary computation 3(2):199–230

Pagie L, Hogeweg P (1997) Evolutionary consequences of coevolving targets. Evolutionary computation 5:401–18

Petersen BK, Landajuela M, Mundhenk TN, et al (2021) Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In: Proc. of the International Conference on Learning Representations

Piñol M, Sappa AD, López A, et al (2012) Feature selection based on reinforcement learning for object recognition. In: adaptive learning agent workshop, pp 4–8

Prieschl S, Girardi D, Kronberger G (2019) Using ontologies to express prior knowledge for genetic programming. In: International Cross-Domain Conference for Machine Learning and Knowledge Extraction, Springer, pp 362–376

Ratle A, Sebag M (2000) Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In: Parallel Problem Solving from Nature, 6th International Conference, pp 211–220

Rosca JP (1996) Generality versus size in genetic programming. MIT Press, p 381–387

Russakovsky O, Deng J, Su H, et al (2015) ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115(3):211–252

Sahoo S, Lampert C, Martius G (2018) Learning equations for extrapolation and control. In: International Conference on Machine Learning, PMLR, pp 4442–4450

Sakakibara Y (2017) Probabilistic context-free grammars. In: Sammut C, Webb GI (eds) Encyclopedia of Machine Learning and Data Mining, Springer, pp 1013–1017

Schmidt M, Lipson H (2009) Distilling free-form natural laws from experimental data. science 324(5923):81–85

Silva SGOd (2008) Controlling bloat: individual and population based approaches in genetic programming. PhD thesis, Coimbra

Sotto LFDP, de Melo VV (2017) A probabilistic linear genetic programming with stochastic context-free grammar for solving symbolic regression problems. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17, p 1017–1024

Sutton RS, Barto AG (2018) Reinforcement learning: An introduction

Tamaki H, Kita H, Kobayashi S (1996) Multi-objective optimization by genetic algorithms: a review. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp 517–522

Udrescu SM, Tegmark M (2020) Ai feynman: A physics-inspired method for symbolic regression. Science Advances 6(16):eaay2631

Uy NQ, Hoai NX, O'Neill M, et al (2011) Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines 12(2):91–119

Verma A, Murali V, Singh R, et al (2018) Programmatically interpretable reinforcement learning. In: Proceedings of the 35th International Conference on Machine Learning, pp 5045–5054

Vladislavleva EJ, Smits GF, den Hertog D (2009) Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Transactions on Evolutionary Computation 13(2):333–349

Whigham PA, et al (1995) Grammatically-based genetic programming. In: Proceedings of the workshop on genetic programming: from theory to real-world applications, pp 33–41

Wierstra D, Foerster A, Peters J, et al (2007) Solving deep memory pomdps with recurrent policy gradients. In: de Sá JM, Alexandre LA, Duch W, et al (eds) Artificial Neural Networks – ICANN 2007, pp 697–706

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning 8(3-4):229–256

Zhang H, Zhou A (2021) Rl-gep: Symbolic regression via gene expression programming and reinforcement learning. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp 1–8