Advances in Data Analysis and Classification
https://doi.org/10.1007/s11634-024-00621-6

REGULAR ARTICLE

Enhancing symbolic regression with side information for data analysis

Xiangwu Zuo[1] · Anxiao Jiang[1]

Received: 3 January 2024 / Revised: 4 June 2024 / Accepted: 23 November 2024

Abstract
This paper introduces the Side Information Boosted Symbolic Regression (SIBSR) model, an enhanced approach in symbolic regression aimed at improving data analysis. SIBSR integrates side information to increase the accuracy and efficiency of modeling complex data relationships. In addition, we introduce the Side Information Generator, a complementary tool designed to assist in generating a range of potential side information options. This enables users to select the most effective side information for specific tasks, thereby enhancing practical utility. Our experimental findings demonstrate the efficacy of SIBSR in standard symbolic regression tasks and its practical application in economic contexts, notably in formulating Nash Equilibrium expressions in Game Theory. These results underscore SIBSR's potential in advancing the field of data analysis. The source codes are available at: https://github.com/dkflame/SIBSR.

Keywords Symbolic regression · Data analysis · Genetic programming · Side information · Game theory · Deep learning

Mathematics Subject Classification 68T07

1 Introduction

Symbolic regression, a field distinguished by its interpretability compared to black-box neural network-based regression methods, focuses on recovering the mathematical symbolic expression to describe the relationship between the input variables

✉ Xiangwu Zuo
dkflame@tamu.edu

Anxiao Jiang
ajiang@cse.tamu.edu

1 Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA

$X = (x_1, x_2, \cdots, x_n)$ and the target variable y. Consider a dataset $(\mathbf{X}, \mathbf{y})$ composed of m samples, where $\mathbf{X} = \{X_1, X_2, \cdots, X_m\}$ represents m samples of X, and $\mathbf{y} = \{y_1, y_2, \cdots, y_m\}$ represents the m corresponding values of $y = f(X)$, where f is an unknown function. Given such a dataset, symbolic regression autonomously discovers the symbolic form of the function that best models the data (such as $y = 1.2x_1 + 5.3\log(x_2) - 2.4\cos(x_3)$ or $y = 2.8\cos(x_1)\sin(3.7x_2/x_3^2) + 2.9x_4$). Its ability to generate human-interpretable symbolic forms marks a distinct advantage in applications requiring clear insights and explanations.

The space of mathematical expressions in symbolic regression is characterized by a combination of discreteness in model structure and continuity in model parameters. This aspect leads to an exponential growth in the complexity of the solution space as the expression length increases, making symbolic regression a challenging task in machine learning, and it is widely considered to be NP-hard (Lu et al. 2016). Genetic Programming (GP), inspired by Darwinian evolutionary principles, addresses this challenge by evolving populations of mathematical expressions represented as binary trees (Ahn 2020; Al-Helali 2019; Schmidt and Lipson 2009; Fortin et al. 2012; Whitley 1994), in which internal nodes serve as mathematical operators, while terminal nodes act as either variables or constants. This structure allows for the concise representation of complex mathematical expressions in a form amenable to evolutionary optimization. Recent advancements include applying deep reinforcement learning methods (Landajuela et al. 2021; Zhang et al. 2021) to symbolic regression, exemplified by approaches of Deep Symbolic Regression (DSR) (Petersen et al. 2021) that utilize recurrent neural networks to sample expressions.

Despite notable advancements, current symbolic regression methods encounter difficulties when dealing with complex expressions and high-dimensional inputs. While domain experts can intuitively identify patterns in data distributions, such as wave-like forms suggesting the presence of sin or cos operators, there is potential that machines can autonomously generate functional elements of the desired symbolic representations. These functions, whether derived from expert insights or machine-generated, can serve as valuable side information, potentially enhancing the effectiveness of symbolic regression solutions. This leads us to explore this topic: how to incorporate such side information to improve the efficiency and accuracy of symbolic regression methods?

To address this, we introduce Side Information Boosted Symbolic Regression (SIBSR), a new approach that integrates auxiliary or "side" information into symbolic regression. This method refines the search for accurate and interpretable mathematical expressions within datasets. Traditional symbolic regression methods (Petersen et al. 2021; Mundhenk et al. 2021) present several limitations: (1) handling multiple input variables: these methods often struggle to accurately recover ground truth expressions when dealing with datasets that have multiple input variables. For instance, for ground truth expressions such as $x^4 - x^3 + \frac{1}{2}y^2 - y$, the traditional symbolic regression methods often fail to recover them. More details can be seen in the Nguyen-12 (Table 2) and Livermore-5 problems (Tables 3 and 2) trigonometric operators: traditional methods also face challenges in accurately identifying symbolic expressions that contain trigonometric operators. This is evident in benchmark problems like Livermore-10 (Table 3), where the presence of trigonometric functions complicates the symbolic regression process, leading to lower recovery rates; (3) scalability and efficiency: the scalability of these methods is often limited, resulting in inefficiencies when applied to larger datasets or more complex problems. The computational resources required to handle such scenarios can be prohibitively high, further limiting their practical applicability. By leveraging side information, SIBSR improves the recovery rate on various benchmark datasets (Refer to Fig. 1 for an illustrative example).

While existing symbolic regression methods have achieved notable progress, there still exist many datasets that pose serious challenges, such as modeling the relationship between Nash Equilibrium points and payoffs in Game Theory. To bridge this gap, we introduce the Side Information Generator, a new tool tailored to automatically generate relevant side information. This tool enhances the efficacy of our SIBSR framework, enabling it to effectively tackle symbolic regression challenges in economic contexts, particularly for Game Theory.

The contributions of this paper include: (1) We introduce SIBSR, a new symbolic regression method that effectively integrates side information, which improves performance compared to existing methods. (2) SIBSR's design allows for the customization of expression similarity metrics and optimization objectives, giving users control over the influence of side information. (3) We also present the Side Information Generator, an auxiliary tool independent of the SIBSR framework, designed to automate the creation of side information expressions for use by SIBSR. (4) Besides applying SIBSR model to standard benchmark data analysis, it is also applied to specific Game Theory problems. This extension showcases the model's capability in analyzing game-related data and identifying Nash Equilibrium expressions within certain limitations. This approach underlines the adaptability of SIBSR in economic scenarios, particularly in the context of strategic game analysis.

2 Related work

Symbolic regression has evolved significantly since the introduction of Genetic Programming (GP) Al-Helali 2019. In symbolic regression, handling high-dimensional data and incomplete datasets poses challenges. High-dimensional data, characterized by a large number of features, complicates learning and hinders model generalization, while incomplete datasets contain missing values that can bias results. Chen et al. (2016) introduced Genetic Programming with Feature Selection (GPWFS), a two-stage method to enhance GP's generalization in high-dimensional tasks by selecting important features from the fittest individuals for regression. This method produces compact, interpretable models with lower computational costs. Building on this, Chen et al. (2017) proposed a permutation-based feature selection method, further improving GP's performance and interpretability by focusing on truly relevant features. For incomplete datasets, Al-Helali et al. (2019) developed a GP-based wrapper imputation method that considers the target variable when constructing imputation models, enhancing symbolic regression performance. They further combined feature selection and imputation (Al-Helali et al. 2020), introducing a sensitivity-based feature importance measure evaluated by noise injection, resulting in improved imputation

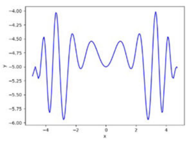
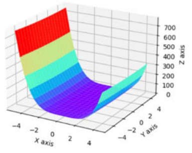
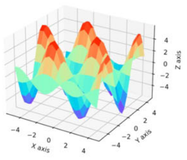
Given a set of data, can we use any expression to describe it?	Side information expression	With the side information, our model approaches the real expression to model the data
	$y = sin(x)$	$y = \sin(x^2)\cos(x) - 5$
	$z = x^2 - y$	$z = x^4 - x^3 + \frac{1}{2}y^2 - y$
	$z = \sin(x) + \sin(y)$	$z = 6\sin(x)\cos(y)$

Fig. 1 Illustration of how side information can help symbolic regression in capturing data relationships and recovering the mathematical expression

accuracy, regression performance, and reduced feature set sizes. However, GP's scalability issues in higher dimensions and the complexity of tuning its hyperparameters remain challenging.

The integration of Neural Networks (NN) in symbolic regression signifies an important shift, primarily due to NN's capabilities in various machine learning tasks (Attali and Pagès 1997; Lawrence et al. 1997), Lample (2020). Efforts to make NN applicable to symbolic regression include Grammar Variational Autoencoders for Bayesian optimization (Kusner et al. 2017) and the use of NN in dimensional analysis for sub-problem fitting (Udrescu and Tegmark 2020). These approaches, though innovative, often result in expressions that lack validity or interpretability. The Equation Learner network (EQL) (Sahoo et al. 2018) attempts to address this by using symbolic operators as activation functions, but this method faces challenges in performing backpropagation and limitations in its practical implementation.

The emergence of Deep Symbolic Regression (DSR) (Petersen et al. 2021) marks a departure from traditional methods, employing a recurrent neural network (RNN) for expression generation and reinforcement learning for expression selection. The RL-GEP system (Zhang et al. 2021) further explores this direction, combining reinforcement learning with GP. The DREGS method (Mundhenk et al. 2021) represents a culmination of these efforts, combining DSR and GP to refine the expression generation process.

There is also a deep connection between symbolic regression and a field named black box optimization. Black box optimization refers to optimization problems where the internal workings of the objective function are unknown, and is widely used in machine learning, industrial design and science (Hansen et al. 2016; Meunier et al. 2021). Symbolic Regression can be seen as a black box optimization problem because the symbolic form of the function is unknown; and interestingly, that symbolic form is also the item that it searches for. For black box optimization, the design of good benchmarks is of critical importance because they help evaluate the general capabilities of the optimization algorithms (Hansen et al. 2009; Liang et al. 2013). For symbolic regression, good benchmarks should capture various aspects of the problem, including the number of variables in the function, the structural complexity of the function, and more. Existing benchmarks for symbolic regression (Petersen et al. 2021; Mundhenk et al. 2021) and the new benchmark of this work gradually increase the difficulty level of those aspects as the algorithms improve over time.

Game Theory presents a practical application for symbolic regression, particularly in generating functions that relate game payoffs to Nash Equilibrium points. Traditional algorithms such as support enumeration (Avis et al. 2010) calculate Nash Equilibrium given specific payoffs, but the challenge lies in the generation of symbolic functions (instead of numerical solutions) for these equilibria.

To address these challenges, we propose the Side Information Boosted Symbolic Regression (SIBSR) method. SIBSR integrates side information into the symbolic regression process, enhancing the accuracy and interpretability of generated expressions. To assess the similarity between side information and generated expressions, we employ sentence similarity metrics from Natural Language Processing (NLP) (Levenshtein 1966). Furthermore, our Side Information Generator automates the generation of side information expressions, streamlining the symbolic regression process. SIBSR aims to bridge the gap in symbolic regression's applicability to complex, real-world problems, exemplified in the domain of Game Theory.

3 Preliminaries

In this section, we mainly discuss the foundational concepts employed in our models. Additionally, we delve into the Deep Symbolic Regression (DSR) and DREGS methods, as well as introducing a baseline method we propose, termed SIBSR-base, which represents an enhanced version of DREGS.

3.1 Notations

In this paper, we employ the following key notations:

- $X = (x_1, x_2, \cdots, x_n)$ represents the set of input variables, with each x_i signifying an individual variable.
- y represents the output target variable.

- $(\mathbf{X}, \mathbf{y})$ denotes a dataset comprising m samples, where $\mathbf{X} = \{X_1, X_2, \cdots, X_m\}$ and $\mathbf{y} = \{y_1, y_2, \cdots, y_m\}$. Each pair (X_j, y_j) constitutes a sample of the unknown ground truth function $y = f(X)$.
- Unless specified otherwise, T, H, and E in subsequent sections refer to the pre-order traversal (Polish notation) of symbolic functions. For example, the pre-order traversal of the symbolic function $\sin(x+1)$ is represented as $[\sin, +, x, 1]$.
- Unless specified otherwise, N denotes the number of expressions sampled from the Expression Generator, M denotes the number of expressions extracted from the GP process, P denotes the number of expressions kept in the Priority Queue, and S denotes the number of generations performed in GP.

3.2 Mathematical token tree and expressions

Mathematical expressions, fundamental in symbolic regression, are efficiently represented as binary token trees. These trees consist of terminal nodes (variables or constants like x, y, 1, 2, 3) and internal nodes (mathematical operators such as $+, -, \times, \div$, sin, cos, log, exp), forming a versatile token library (Petersen et al. 2021; Lample and Charton 2020). This structure facilitates a systematic organization where mathematical relationships are hierarchically arranged.

The token tree is constructed by placing these tokens in a binary tree layout, with the encapsulated expression extracted through a pre-order traversal. This traversal translates the token sequence into a coherent mathematical expression, as depicted in Fig. 2B. It demonstrates the conversion of a structured token arrangement into an interpretable mathematical formula.

In the process of forming mathematical expressions through symbolic regression, it is essential to apply certain constraints to ensure both mathematical validity and computational efficiency. These constraints are designed to eliminate redundant or non-contributory structures, thereby streamlining the search space for more meaningful and efficient expressions. The constraints applied include: (1) Limiting expression length within a pre-determined range, dictated by the token tree's pre-order traversal sequence. (2) Prohibiting trigonometric operators (e.g., sin, cos) from having similar operators as descendants to avoid redundancy (e.g., $\sin(\cos(x))$ is prohibited). (3) Disallowing unary operators (e.g., sin, cos, log, exp) from containing their inverse operators as children to maintain mathematical integrity (e.g., $\exp(\log(x))$ is not allowed).

3.3 Genetic programming

Genetic Programming (GP), a cornerstone in solving symbolic regression challenges, models mathematical expressions as evolvable tree structures. Each node in these trees is either an operator or operand, representing an element of the mathematical expression. GP's evolution operations-selection, mutation, and crossover-facilitate the iterative refinement of expressions (Whitley 1994).

Selection Operation involves choosing superior expressions from the current generation to parent the next. Mutation Operation randomly alters an expression by

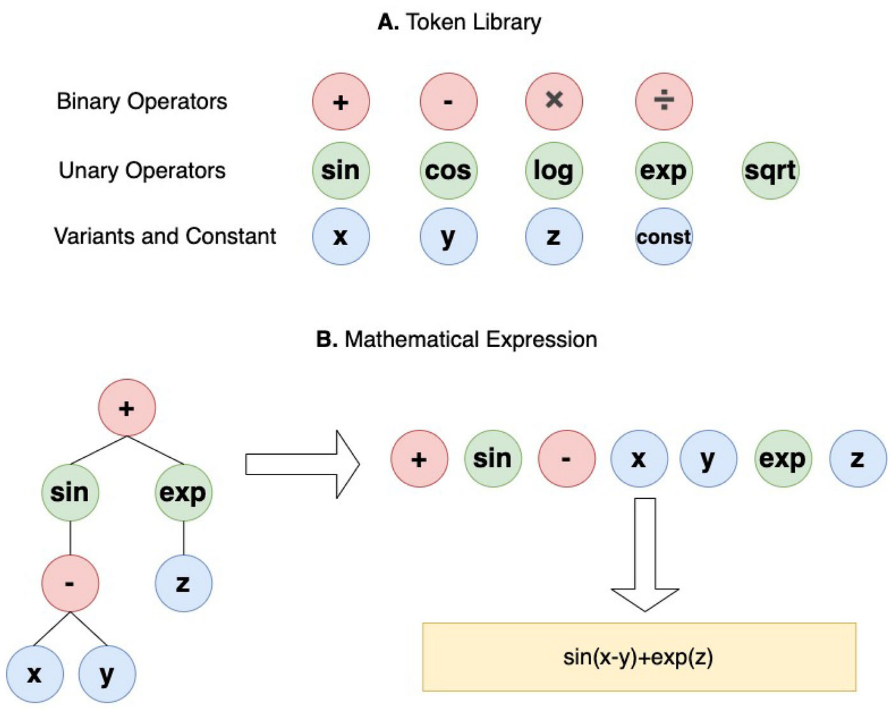


Fig. 2 **A** Illustration of the token library. **B** Example of converting a binary token tree into a mathematical expression through pre-order traversal

modifying its tokens. Crossover Operation exchanges tokens between pairs of expressions, thereby combining their features. This process, illustrated in Fig. 3, showcases GP's dynamic approach to evolving expressions, where different operations can simultaneously occur within the same generation.

3.4 Deep symbolic regression (DSR)

DSR (Petersen et al. 2021) is an important baseline model being compared to in our experiments. A token library like Fig. 2A is prepared. Using an RNN controller with parameters θ, DSR autoregressively samples tokens from the library at each time step. The sampled tokens are combined to form a token tree, which can be written as a pre-order traversal sequence and then converted into a mathematical expression. The likelihood of the generated expression T is $p(T|\theta) = p(t_1|\theta)\prod_{i=2}^{n} p(t_i|t_1, \ldots, t_{i-1}, \theta)$, where t_i denotes the token generated at time step i. In each training epoch, N expressions are sampled. The RNN controller is treated as an agent that is trained using reinforcement learning. The reinforce policy reward is a squashing function of the normalized root-mean-square error (NRMSE). Given an expression T and a dataset $(\mathbf{X}, \mathbf{y})$ with m data samples, NRMSE can be defined as

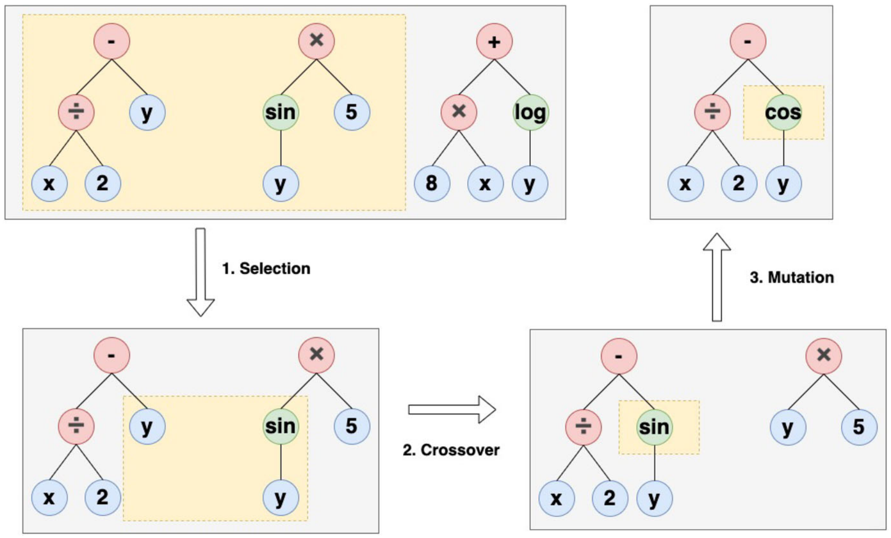


Fig. 3 A Genetic Programming process example. The operations are performed based on fitness

$\mathrm{NRMSE}(T) = \frac{1}{\sigma_y}\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}$, where $\hat{y}_i$ is the predicted value of the expression T with input X_i, and σ_y is the standard deviation of the target value y. Finally, the policy reward function is written as $R(T) = 1/[1 + \mathrm{NRMSE}(T)]$. The structure of DSR is illustrated in Fig. 4A.

3.5 DREGS: A model that combines deep symbolic regression and genetic programming

This model is proposed in Mundhenk et al. (2021), where it is called Symbolic Regression via Neural-Guided Genetic Programming Population Seeding. It combines the ability of recurrent neural networks (RNNs) to generate expressions with the need of genetic programming (GP) for an initial population of expressions. In DREGS, expressions are sampled from the RNN controller and used as the initial population for GP. The set of N expressions generated by the RNN is denoted by $\mathcal{T}_{rnn}$, while the set of M expressions extracted by GP is denoted by $\mathcal{T}_{gp}$. DREGS combines $\mathcal{T}_{rnn}$ with $\mathcal{T}_{gp}$ and then keeps the best P of them in a priority queue, based on their fitness values. (Here P is a preset parameter. For example, P can be 100.) The expressions are evaluated for their fitness, and those with the highest fitness values are selected to be retained in the priority queue. This ensures that the most promising expressions are prioritized for further optimization. The RNN controller is trained on the combined expressions using policy-based reinforcement learning (e.g. VPG, RSPG, PQT). The reward function $R(T)$ for DREGS is the same as the $R(T)$ for DSR and is given by $R(T) = 1/[1+\mathrm{NRMSE}(T)]$. In DREGS, the GP process is restarted at each epoch, because the initial population changes as the RNN controller is updated during training. The structure of DREGS is shown in Fig. 4B.

3.6 DREGS with side information (SIBSR-base)

For our target, side information is required to be implemented in the symbolic regression task. Our baseline method (SIBSR-base) is an improvement over the DREGS method, which uses a recurrent neural network (RNN) to sample expressions before the genetic programming process. In addition to the expressions generated by the RNN, SIBSR-base also considers side information expressions as part of the initial population for genetic programming. Instead of using only the expressions $\mathcal{T}_{rnn}$ generated by the RNN, SIBSR-base's initial population is given by $\mathcal{T}_{init\_gp} = \mathcal{T}_{rnn} \cup \{H\}$, where H denotes the side information expression. To ensure that the RNN generates expressions that are close to the side information expression, we design new metrics to compare the generated expressions with the side information expression and adjust the optimization objective accordingly. The adjusted optimization objective is the same for both SIBSR-base and our proposed SIBSR model, which is described in more detail in the following section. The structure of the SIBSR-base method is illustrated in Fig. 4C.

4 Side information boosted symbolic regression

The Side Information Boosted Symbolic Regression (SIBSR) method synergistically integrates three core components to enhance symbolic regression: (1) An Expression Generator, featuring an encoder-decoder mechanism adept at processing pre-order traversal sequences from mathematical token trees, specifically those derived from side information expressions. (2) A Genetic Programming (GP) module, utilizing both the side information expression sequence and the expressions sampled by the Expression Generator. (3) A Reinforcement Learning Reward Function, tailored to incorporate the side information expression, enabling adjustable weighting based on user preference. This section delves into the comprehensive architecture of SIBSR.

4.1 Side information

"Side Information" refers to any knowledge that is correlated with the ground-truth function $y = f(X)$. In fields of science and engineering, people often leverage functions derived from prior knowledge or experience. For instance, exponential or power-law functions might be employed to model the decaying tails of bell-shaped distributions, and trigonometric functions to represent oscillating data.

Furthermore, an emerging dimension of side information arises from machine-generated functions. These functions, produced through algorithms or AI-based processes, can provide valuable insights or partial components of the target function f. For example, a machine-learning algorithm might autonomously identify and generate polynomial functions that partially match the behavior of f in specific data

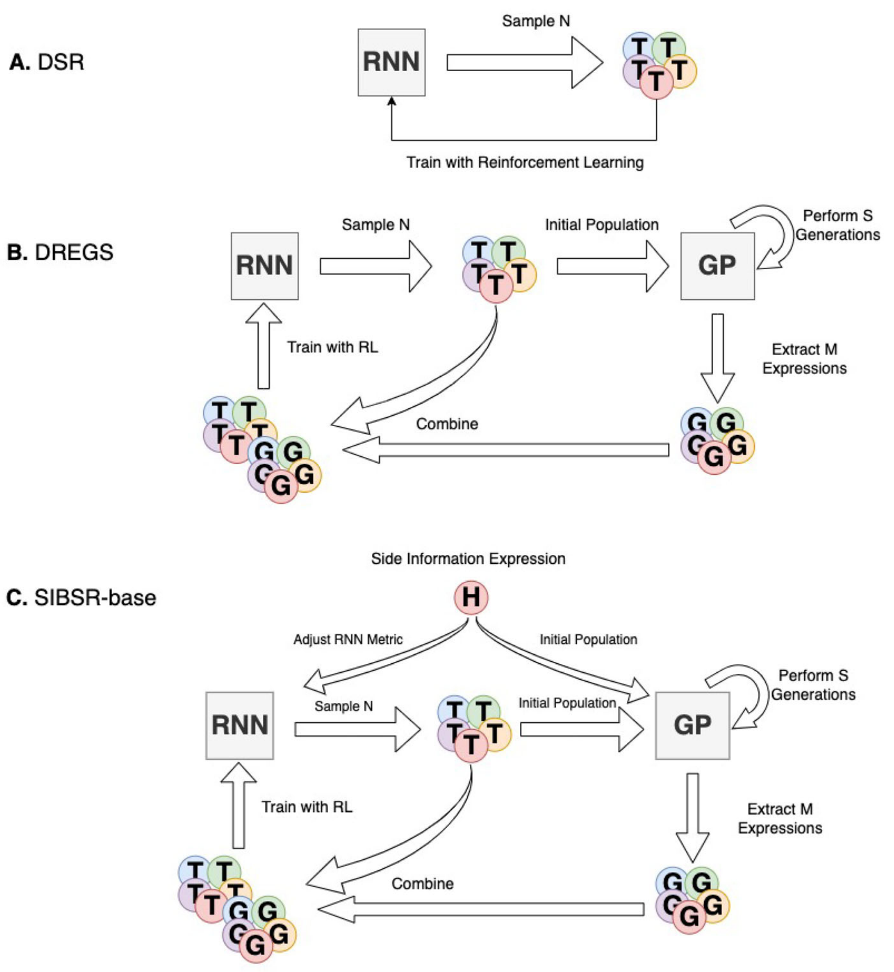


Fig. 4 **A** DSR. **B** DREGS. **C** Our baseline method structure (SIBSR-base)

ranges, or it could propose logarithmic or exponential functions that capture certain trends or patterns within the dataset.

Generally, both human-generated and machine-generated functions, when aligned with the data's inherent characteristics, serve as valuable side information. In this work, we particularly focus on side information with specific formats that enhance the efficiency of searching within the extensive function space of symbolic regression.

4.2 Model pipeline

Symbolic regression, essentially an inverse problem-solving approach, starts with datapoints and seeks a mathematical expression that aptly models the data. Conventional methods like DSR, GEGL, and DREGS focus on exploring token combinations in a predefined space (Petersen et al. 2021; Kim et al. 2021; Mundhenk et al. 2021; Zhang et al. 2021; Ahn et al. 2020). However, increasing sequence length leads to exponential growth in search space, posing challenges in timely identification of accurate expressions.

Our approach acknowledges that simplified versions of mathematical expressions can serve as side information in the symbolic regression process. These simplified expressions often encapsulate key aspects of datasets with straightforward distributions. Incorporating such side information into symbolic regression can narrow down the search space and enhance model efficiency. SIBSR embeds this concept, effectively integrating machine-generated side information into the regression process. This not only narrows down the search space but also adds an element of automated insight into the symbolic representation of complex datasets. The methodology for automatically generating this side information is detailed in Sect. 5. The pipeline of SIBSR is illustrated in Fig. 5.

(1) Side Information encapsulates a probable mathematical function characterizing the data. This function is translated into a pre-order traversal expression H, which is then input into the Expression Generator's encoder. The encoder distills H into a context vector C. This step is illustrated by "1) Use as input expression" in Fig. 5.

(2) We prepare a token library, as shown in Fig. 2A, comprising mathematical operators (e.g., $+$, $-$, $\times$, $\div$, sin, log) and variables/constants. The Expression Generator's decoder samples tokens from this library, blending them with C to establish a categorical distribution over the tokens at each RNN timestep. The decoder's output forms a new pre-order traversal sequence T for the generated token tree. This step is illustrated by "2) Decode" in Fig. 5.

(3) Each training epoch involves evaluating the sampled N expressions for their policy rewards $R(T)$, preparing for subsequent steps. This step is illustrated by "3) Sample N Expressions T" in Fig. 5.

(4) The Side Information Expression H is replicated K times, where K is the dataset's variable count. These replications, combined with the N sampled expressions, constitute the initial population for S GP generations. The top M expressions from these generations E are selected based on their reward values. This step is illustrated by "4) Extract M Expressions E" in Fig. 5.

(5) The M GP-derived expressions are merged with the N expressions from the Expression Generator, culminating in a pool of $M + N$ expressions. The best P of these are chosen for the priority queue, which further trains the Expression Generator via policy-based reinforcement learning, such as the PQT algorithm. This step is illustrated by "5) Keep P Expressions" in Fig. 5.

(6) Post training, the optimal expression is selected based on the highest reinforcement policy reward $R(T)$, or upon early stopping of the training process. (In our experiment, the training meets the early stopping criterion when NRMSE $< 10^{-12}$.) This step is illustrated by "6) Calculate $R(T)$ for each expression. Train Expression Generator with RL" in Fig. 5.

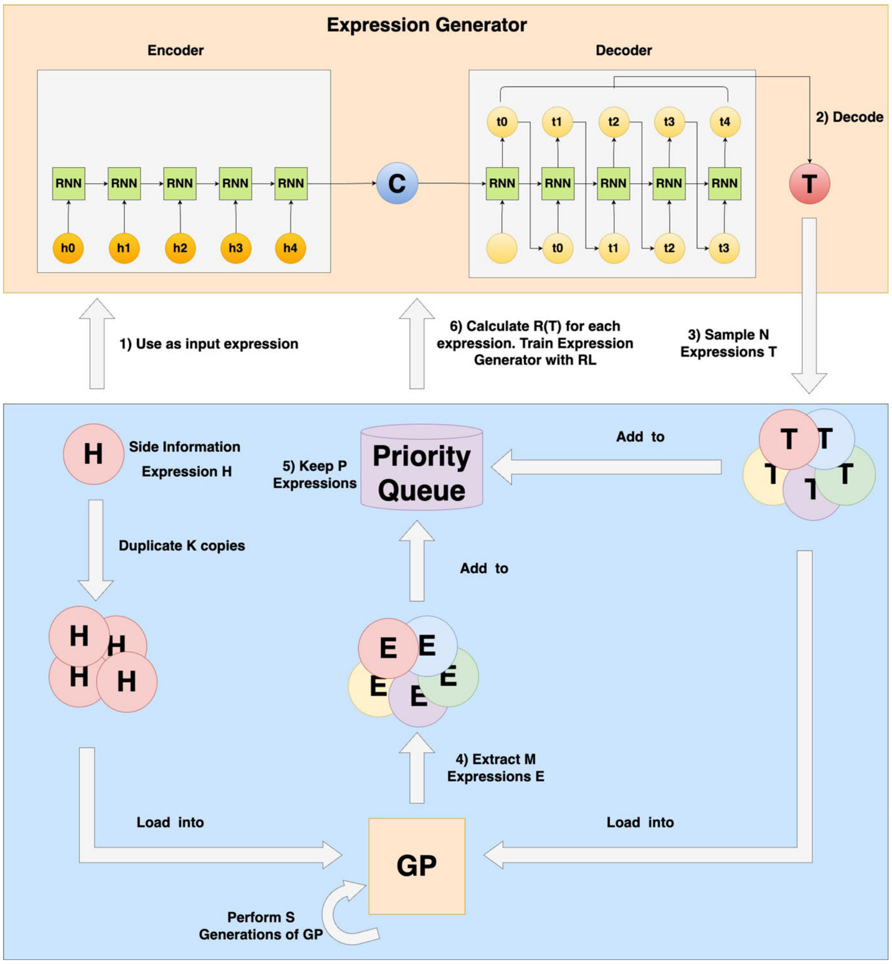

Expression Generator
Encoder
Decoder
RNN
h0
h1
h2
h3
h4
C
t0
t1
t2
t3
t4
2) Decode
T
1) Use as input expression
6) Calculate R(T) for each expression. Train Expression Generator with RL
3) Sample N Expressions T
H
Side Information Expression H
5) Keep P Expressions
Priority Queue
Add to
Duplicate K copies
Add to
E
4) Extract M Expressions E
Load into
Load into
GP
Perform S Generations of GP

Fig. 5 Side Information Boosted Symbolic Regression pipeline structure

4.3 Expression generator

The Expression Generator, pivotal in SIBSR, employs RNN-based cells (like RNNs, LSTMs, GRUs) for its encoder and decoder. The side information is denoted by $H = [h_1, \cdots, h_s]$, where each h_i represents a token in the expression sequence. The encoder compresses H into a context vector C, initiating the decoder's RNN state. The decoder, functioning as an autoregressive network, iteratively emits a categorical distribution at each step, leading to the generation of an expression T. Mathematical operators and input variables/constants are treated as tokens, each embedded using one-hot encoding. The Expression Generator's parameters are symbolized as θ. The probability of generating $T = [t_1, \cdots, t_v]$ given H and θ is defined as follows

$$\begin{aligned} p(T|H;\theta) &= p(t_1, \cdots, t_v|h_1, \cdots, h_s;\theta) \\ &= \prod_{i=1}^{n} p(t_i|t_1, \cdots, t_{i-1}, h_1, \cdots, h_s;\theta) \\ &= \prod_{i=1}^{n} p(t_i|t_1, \cdots, t_{i-1}, C;\theta) \end{aligned} \tag{1}$$

4.4 Initializing GP with the side information

Each training batch commences by sampling the N highest-ranked expressions (T) generated by the Expression Generator. These expressions, along with the side information (H), form the initial GP population. GP's evolutionary mechanics—selection, crossover, and mutation—reshape this population across generations. The GP output, $\mathcal{E} = \{E_1, \cdots, E_M\}$, emerges as the set of top-ranked M expressions E_i after performing S generations. These are then combined with the initial N expressions from the Expression Generator, with the top P retained for training via policy-based reinforcement techniques like PQT.

4.5 Optimization objective

Given a dataset $(\mathbf{X}, \mathbf{y})$, for the expression T, we generate a set of datapoints $(\mathbf{X}, \mathbf{y}_T)$, where each y_T value is calculated based on the expression T applied to the dataset's X sample. Similarly, we create another set of datapoints $(\mathbf{X}, \mathbf{y}_H)$ using the expression H. Here, each y_H value is derived by applying the expression H to the same X sample. We evaluate the similarity between the expression T generated by our Expression Generator and side information H with two metrics. The first metric is the normalized Euclidean distance of the sampled datapoints between T and H, which is defined as $d(T, H) = \sum_{i=1}^{m} \sqrt{(y_H - y_T)^2/y_H^2}$. The second metric is the sentence similarity of the two expressions. Since each expression can be treated as a sequence in our model, we calculate the Levenshtein distance (Navarro 2001) between the two expressions. The Levenshtein distance is a measure of the similarity between two strings, and is defined as the minimum number of deletions, insertions, or substitutions required to transform one string into the other. We define sentence similarity as the Levenshtein distance divided by the sum of the lengths of the two expressions. This can be written as $ss(T, H) = \frac{Levenshtein\_distance(T,H)}{length(H)+length(T)}$

The standard fitness metric for a symbolic regression task is typically the normalized root-mean-square error (NRMSE). Both DSR and DREGS use NRMSE as their metric, and the reward function is given by $R(T) = 1/[1 + NRMSE(T)]$. In our case, we combine NRMSE with our defined two metrics, $d(T, H)$ and $ss(T, H)$, to create our optimization objective.

4.6 Reward function

As the DSR method uses a reward function of $1/[1+NRMSE(T)]$ and employs reinforcement learning to train the model, we apply a squashing function to our combined metric. The reward function become

$$R(T) = w_1/[1+NRMSE(T)] + w_2/[1+d(T,H)] + w_3/[1+ss(T,H)], \quad (2)$$

where w_1, w_2, w_3 are the weights defined by user.

The reinforce policy gradient objective $J(\theta)$ is defined as the expectation of the reward function $R(T)$: $J(\theta) = E_{T\sim p(T|H;\theta)}[R(T)]$. As we sample N expressions from Expression Generator at each training epoch, the objective function can be maximized via gradient ascent with the learning rate α

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta E_{T\sim p(T|H;\theta)}[R(T)] = E_{T\sim p(T|H;\theta)}[R(T)\nabla_\theta \log p(T|H;\theta)] \\ &\approx \frac{1}{N}\sum_{i=1}^{N} R(T^{(i)})\nabla_\theta \log p(T^{(i)}|H;\theta) \\ \theta &\leftarrow \theta + \alpha\nabla_\theta J(\theta) \end{aligned} \quad (3)$$

4.7 SIBSR algorithm

The pseudocode of our SIBSR algorithm is provided in Algorithm 1. The definitions for the mathematical symbols used in the algorithm are explained in previous sections.

Algorithm 1 Side Information Boosted Symbolic Regression

```
Input: Side Information expression $H$; token library $TL$; Expression Generator (EG) sample
    size $N$; GP sample size $M$; Priority Queue length $P$; number of data features $K$; minimal
    and maximal expression length $L_{min}$ and $L_{max}$; number of GP generations per epoch
    $S$; GP crossover probability $P_c$; GP mutation probability $P_m$; policy reward function R;
    weights $w1, w2, w3$ for $NRMSE, d, ss$ metrics; Threshold $\eta_{thr}$; Learning rate $\alpha$
Output: Best generated expression $T^*$
 1: Initialize the Expression Generator Encoder and Decoder with $\theta$
 2: Initialize GP with $\omega$, $GP = GP_{P_c,P_m}(\omega)$
 3: Initialize policy reward function $R = R_{NRMSE,d,ss}$ with $w1, w2, w3$
 4: repeat
 5:     Encode $H$ to $C$, $C \leftarrow Encoder(H)$
 6:     Sample $N$ expressions from Expression Generator, $\mathcal{T}_{EG} \leftarrow \{T^{(i)} \sim p(\cdot|C;\theta)\}_{i=1}^{N}$
 7:     $\mathcal{H} \leftarrow$ Duplicate $K$ copies of $H$
 8:     Combine $\mathcal{H}$ and $\mathcal{T}_{EG}$ to be the initial GP population, $\mathcal{T}_{GP}^{(0)} \leftarrow \mathcal{T}_{EG} \cup \mathcal{H}$
 9:     Initialize $\mathcal{T}_{GP} \leftarrow GP(\mathcal{T}_{GP}^{(0)})$
10:     for s = 1 to S do
11:         $\mathcal{T}_{GP}^{(s)} \leftarrow GP(\mathcal{T}_{GP}^{(s-1)})$
12:         $\mathcal{T}_{GP} \leftarrow \mathcal{T}_{GP} \cup GP(\mathcal{T}_{GP}^{(s)})$
13:     end for
14:     Extract best $M$ expressions from all the $S$ generations of GP, $\mathcal{E} \leftarrow BestM(GP(\mathcal{T}_{GP}))$
15:     Combine $T_{EG}$ with $\mathcal{E}$, $\mathcal{T}_{comb} \leftarrow \mathcal{T}_{EG} \cup \mathcal{E}$
16:     Filter the best $P$ expressions and load to the priority queue, $\mathcal{T}_{train} \leftarrow BestP(T_{comb})$
17:     Calculate rewards of the best $P$ expressions, $\mathcal{R} \leftarrow \{R(T), \forall T \in \mathcal{T}_{train}\}$
18:     Train the Expression Generator with policy-based Reinforcement Learning, $\theta \leftarrow \theta +$
        $\alpha \nabla_\theta J(\theta)$
19:     Select the expression with maximal reward, $R(T^*) \leftarrow \max \mathcal{R}$, $T^* \leftarrow T^{\arg\max \mathcal{R}}$
20: until Training Finished or $NRMSE(T^*) < \eta_{thr}$
21: return $T^*$
```

5 Side information generator

This section introduces the Side Information Generator, a key component of our SIBSR method that autonomously provides side information for the symbolic regression process.

To reduce the symbolic regression search space, we consider two strategies: (1) Constraining the length of the target expression based on side information: We align the length of our search expressions with that of the side information expressions. This strategy is particularly practical when the side information closely approximates the length of the true function, as often observed in applications like Game Theory, where expressions generally utilize every payoff value. (2) Modifying the token library based on side information: If the side information includes specific basis functions (e.g., $\frac{1}{x}$, $\sin(x)$), these can be used to replace certain tokens in the library. This approach aids in focusing the search and enhancing the noise resilience of the symbolic regression.

Our method generates a suite of side information expressions adaptable for symbolic regression, drawing inspiration from McConaghy (2011). It encompasses three main stages: (1) Basis function generation, (2) Side information generation using ElasticNet, and (3) Conversion of basis function combinations into tokens.

5.1 Basis function generator

Basis functions, containing simple operators, are the building blocks for target functions. For instance, in a target function like $\sin(x) + \log(y) + xy$, the terms $\sin(x)$, $\log(y)$, xy are the basis functions. We create a series of these functions using the original variables combined with various operators. Algorithm 2 outlines this generation process.

Algorithm 2 Basis Function Generator

```
Input: Unary Operators $UOP = \{\sin, \cos, \log, \exp \cdots\}$; Binary Operators $BOP = \{+, -, \times, \div \cdots\}$; Exponents $EP = \{1, 2 \cdots\}$; Input Variables $X = (x_1, x_2, \cdots, x_n)$; Output target $y$;
Output: A set of basis functions $B$.
 1: $B_1 \leftarrow []$
 2: for each $x_i$ in $X$ do
 3:     for each exponent $ep$ in $EP$ do
 4:         Generate basis function of $b_{exp} = x_i^{ep}$
 5:         $B_1 \leftarrow B_1 \cup \{b_{exp}\}$.
 6:         for each unary operator op1 in $UOP$ do
 7:             Generate basis function of $b_{op1} = op1(b_{exp})$
 8:             $B_1 \leftarrow B_1 \cup \{b_{op1}\}$.
 9:         end for
10:     end for
11: end for
12: $B_2 \leftarrow []$
13: for $i = 1$ to $length(B_1) - 1$ do
14:     $b_i \leftarrow B_1[i]$
15:     for $j = i + 1$ to $length(B_1)$ do
16:         $b_j \leftarrow B_1[j]$
17:         for each binary operator op2 in $BOP$ do
18:             Generate basis function of $b_{op2} = op2(b_i, b_j)$
19:             $B_2 \leftarrow B_2 \cup \{b_{op2}\}$.
20:         end for
21:     end for
22: end for
23: $B \leftarrow B_1 \cup B_2$.
24: return $B$
```

5.2 Generating side information with basis functions

In our analysis of Game Theory problems, we observe a predominant use of specific expression formats. These recurring formats, primarily linear and fractional combinations, effectively serve as implicit side information, guiding our focus. Consequently, we prioritize linear and fractional basis function combinations in our approach, aligning our method with the inherent patterns observed in Game Theory scenarios.

5.2.1 Linear combination

In a linear combination of N basis functions $B = \{B_1, B_2, \cdots, B_N\}$, the expression is formulated as $y = c_0 + \sum_{i=1}^{N} c_i B_i(X)$. ElasticNet is applied to determine the coefficients C in this linear context, as shown in the following equation:

$$C = \text{minimize}||y - CB(X)|| + \lambda_1||C|| + \lambda_2||C||^2 \tag{4}$$

5.2.2 Fractional combination

For a fractional combination of N basis functions $B = \{B_1, B_2, \cdots, B_N\}$, the denominator has N_{BD} number of basis functions sampled from B and the numerator has N_{BN} number of basis functions from B. It is described as:

$$y = \frac{c_0 + \sum_{j=1}^{N_{BN}} c_j B_j(X)}{1 + \sum_{i=1}^{N_{BD}} c_i B_i(X)} \tag{5}$$

Performing the simple algebra to Eq. 5, we get:

$$y + \sum_{i=1}^{N_{BD}} c_i B_i(X) y = c_0 + \sum_{j=1}^{N_{BN}} c_j B_j(X) \tag{6}$$

Then we get:

$$y = c_0 + \sum_{j=1}^{N_{BN}} c_j B_j(X) - \sum_{i=1}^{N_{BD}} c_i B_i(X) y \tag{7}$$

Define $N_{all} = N_{BN} + N_{BD}$, then equation is identical to:

$$y = c_0 + \sum_{i=N_{BD}+1}^{N_{all}} c_i B_i(X) - \sum_{i=1}^{N_{BD}} c_i B_i(X) y \tag{8}$$

Equation 8 can be split into two different equations to calculate y_{pred}:

$$y = \begin{cases} c_0 - \sum_{i=1}^{N_{BD}} c_i B_i(X) y & for\ i \leq N_{BD} \\ c_0 + \sum_{i=N_{BD}+1}^{N_{all}} c_i B_i(X) & otherwise \end{cases} \tag{9}$$

By treating $B_i(X)y$ as the basis functions for $i \leq N_{BD}$, we implement an elasticNet to calculate all the coefficients c_i. With these coefficients, we can fully derive the fractional function of Eq. 5. In contrast to the definition in McConaghy (2011), our method guarantees that the denominator of the fractional function contains basis functions.

Algorithm 3 describes how the side information expression is generated with basis functions.

Algorithm 3 Generating Side Information using Basis Functions

```
Input: Input Variables $X = (x_1, x_2, \cdots, x_n)$; Output target $y$; Basis functions $B$; Number
    of basis function $N = length(B)$, number of denominator basis functions $N_{BD} \leq N$,
    number of numerator basis functions $N_{BN} \leq N$, and $N_{ALL} = N_{BN} + N_{BD}$;
Output: A set of Side Information Expressions $\mathcal{H}$.
1: $\mathcal{H} \leftarrow \{\}$
2: for $i = 1$ to $N$ do
3:     Compute $B_i(X)$
4: end for
5: if Use Linear Model then
6:     $N_{base} = 1$
7:     while $N_{base} \leq N$ do
8:         $M_{linear} \leftarrow y = c_0 + \sum_{i=1}^{N_{base}} c_i B_i(X)$
9:         Fit ElasticNet($M_{linear}$) and generate $H_{linear}$.
10:        $\mathcal{H} \leftarrow \mathcal{H} \cup \{H_{linear}\}$
11:        $N_{base} = N_{base} + 1$
12:    end while
13: end if
14: if Use Fractional Model then
15:    $N_{base} = 1$
16:    while $N_{base} \leq N_{BD}$ do
17:        $M_{frac} \leftarrow y = c_0 - \sum_{i=1}^{N_{base}} c_i B_i(X) y$
18:        Fit ElasticNet($M_{frac}$) and generate $H_{frac}$.
19:        $\mathcal{H} \leftarrow \mathcal{H} \cup \{H_{frac}\}$
20:        $N_{base} = N_{base} + 1$
21:    end while
22:    while $N_{BD} < N_{base} \leq N_{ALL}$ do
23:        $M_{frac} \leftarrow y = c_0 + \sum_{i=N_{base}+1}^{N_{ALL}} c_i B_i(X)$
24:        Fit ElasticNet($M_{frac}$) and generate $H_{frac}$.
25:        $\mathcal{H} \leftarrow \mathcal{H} \cup \{H_{linear}\}$
26:        $N_{base} = N_{base} + 1$
27:    end while
28: end if
29: return $\mathcal{H}$
```

5.3 Token adaptation

Given the generated side information expressions closely align with our target symbolic forms, we adapt our token library accordingly. The method for converting basis functions into tokens, enhancing the Expression Generator's efficiency, is outlined in Algorithm 4.

Algorithm 4 Basis Functions to Tokens

```
Input: Side Information Expression $H$; Basis functions $B_H$; Coefficients $C$ for $B_H$ in $H$;
    Input variables $X = (x_1, x_2, \cdots, x_n)$;
Output: A set of Tokens $TK$ for Symbolic Regression.
 1: $TK \leftarrow X$
 2: if $H$ is linear expression then
 3:     Sort $B_H[i] \in B_H$ by the order of $C[i] \in C$
 4:     Assign $C[i] \cdot B_H[i]$ into three groups based on the large, middle, and small values of
        $C[i]$
 5:     $TK_{large} = \sum C(large)[i] \cdot B_H(large)[i]$
 6:     $TK_{middle} = \sum C(middle)[i] \cdot B_H(middle)[i]$
 7:     $TK_{small} = \sum C(small)[i] \cdot B_H(small)[i]$
 8:     $TK \leftarrow TK \cup TK_{large} \cup TK_{middle} \cup TK_{small}$
 9: end if
10: if $H$ is fractional expression then
11:     Split the $B_H$ to $B_H^N$ as numerator and $B_H^D$ as denominator
12:     $TK^N \leftarrow \sum C[i] \cdot B_H^N[i]$
13:     $TK \leftarrow TK \cup TK^N$
14:     Sort $B_H^D[i] \in B_H^D$ by the order of $C[i] \in C$
15:     Group $C[i] \cdot B_H^D[i]$ into three of large, middle, small value of $C[i]$
16:     $TK_{large} = \sum C(large)[i] \cdot B_H^D(large)[i]$
17:     $TK_{middle} = \sum C(middle)[i] \cdot B_H^D(middle)[i]$
18:     $TK_{small} = \sum C(small)[i] \cdot B_H^D(small)[i]$
19:     $TK \leftarrow TK \cup TK_{large} \cup TK_{middle} \cup TK_{small}$
20: end if
21: return $TK$
```

6 Experiments on benchmark problems

This section demonstrates the efficacy of our SIBSR model on benchmark problems, highlighting its capabilities in data analysis contexts, particularly in handling multi-variable and complex expressions. We compare SIBSR's performance against state-of-the-art models in terms of accuracy and computational efficiency.

6.1 Settings of hyper-parameters

In this work, we build the SIBSR framework and the Side Information Generator using Python and Tensorflow. One important package that it utilizes is the DSO (Deep Symbolic Optimization) package, which realizes the Deep Symbolic Regression (DSR) Algorithm of Petersen et al. (2021); Mundhenk et al. (2021). Note that our framework SIBSR generalizes the DSR algorithm by incorporating side information. We set the weights of the optimization objective in Eq. 2 to be $w_1 = 0.7$, $w_2 = 0.1$, and $w_3 = 0.2$. In our experiments, the weights w_1, w_2, w_3 are set within the range [0, 1] and constrained to sum to 1. We conduct trials with 22 different combinations, allowing w_1, w_2, w_3 to vary from 0 to 1 in steps of 0.1, while ensuring that $w_1 \geq w_2$ and $w_1 \geq w_3$. This configuration is chosen to explore a substantial but manageable number of settings, as excessively numerous combinations could be impractical due to the time required for evaluation. The selection criteria for these weights are primarily based on achieving the most stable performance. Stability here is defined as the ratio of recovery rate to the time consumption of the algorithm, and we select the combination of w_1, w_2, w_3 that achieves the highest value. The total number of expression samples for the Expression Generator is set to 50, 000, with a batch size of 500. GP runs 10 generations in each epoch, and the number of GP-generated expressions is set to 30. Both the GP crossover rate and mutation rate are set to 50%. The length of the expressions is allowed to be between 5 and 50, and the length of the maintained priority queue is set to 10. The threshold for the early stop is set to 10^{-12}.

6.2 Benchmarks

We use two existing benchmark problem sets, Nguyen (Uy et al. 2011) and Livermore (Mundhenk et al. 2021), to compare the performance of our method (SIBSR) with DSR, DREGS, and SIBSR-base methods. A symbolic regression benchmark problem is defined by a ground truth expression, a set of data points, and a set of allowable tokens. In this work, we also introduce a new benchmark problem set, called SIBC, which is shown in Table 1. The SIBC benchmarks are specifically designed to test the model's proficiency in handling data analysis scenarios with multiple variables and complex expressions, addressing the limitations in existing benchmarks (Uy et al. 2011; Mundhenk et al. 2021). As described in the Introduction, traditional symbolic regression methods have limitations in handling symbolic formulas with multiple variables or trigonometric operators. Consequently, the existing benchmarks often focus on simpler scenarios involving single variables and without trigonometric operators. In contrast, the SIBC benchmark includes many symbolic expressions with multiple variables or complex trigonometric operators, thereby overcoming these noted limitations and providing a more robust test of the model's capabilities.

6.3 Comparison of recovery rates

Recovery is defined as the exact symbolic expression being discovered by the model. Therefore, the recovery rate is defined as the number of times the real expression is discovered divided by the total number of experiments. For each benchmark, we run experiments on the SIBSR and SIBSR-base models and compare their performance with that of the DREGS and DSR from Petersen et al. (2021); Mundhenk et al. (2021). The comparison results of the recovery rate on the Nguyen and Livermore benchmarks are shown in Tables 2 and 3. The given side information expressions are typically set as the highest power of a variable in the real expression. From the tables, we observe that the SIBSR method discovers the exact expressions for most benchmark problems in this paper. The SIBSR method achieves higher recovery rates compared to state-of-the-art models on those benchmarks.

Table 1 Specifications of SIBC benchmarks. Input variables are denoted by $x_1, x_2, \cdots, x_i$; $U(r, s, t)$ denotes t datapoints of each x are uniformly and randomly sampled from r to s

Benchmark	Expression	Dataset
SIBC-1	$\sin(x_1^2) + x_2 - \sin(x_3)$	$U(1, 20, 20)$
SIBC-2	$\sin(x_1) + 2\sin(x_2) - \cos(x_3)$	$U(-10, 10, 20)$
SIBC-3	$\sin(x_1) + 2\sin(x_2) + 3\sin(x_3)$	$U(-1, 1, 20)$
SIBC-4	$x_1x_2 + x_3$	$U(-1, 1, 20)$
SIBC-5	$x_1 + 2x_2 + 3x_3$	$U(-1, 1, 20)$
SIBC-6	$\log(x_1) + 2\log(x_2) + 3\log(x_3)$	$U(1, 10, 20)$
SIBC-7	$\log(x_1 + 3) + \log(x_2 + 2) + \log(x_3 + 1)$	$U(1, 10, 20)$
SIBC-8	$x_1x_2 + \sin(x_2) + \log(x_3)$	$U(1, 10, 20)$
SIBC-9	$\sin(x_1) + \log(x_2) + \exp(x_3) + 2x_4$	$x_1 \in U(-5, 5, 20), x_2 \in U(1, 10, 20), x_3 \in U(-5, 5, 20), x_4 \in U(-5, 5, 20)$
SIBC-10	$\sin(x_1)\cos(x_2) + \log(x_3)$	$x_1 \in U(-1, 1, 20), x_2 \in U(-1, 1, 20), x_3 \in U(1, 10, 20)$
SIBC-11	$\sin(x_1) + 2\sin(x_2) + \cos(x_3) + 2\cos(x_4)$	$U(-5, 5, 20)$
SIBC-12	$\log(x_1) + 2\log(x_2) + \log(x_3) - 2\log(x_4)$	$U(1, 20, 20)$
SIBC-13	$\log(x_1) + \log(x_2) + \exp(x_3) + \exp(x_4)$	$U(1, 2, 20)$
SIBC-14	$\sin(x_1) + \log(x_2) + \exp(x_3) + \cos(x_4)$	$U(1, 2, 20)$
SIBC-15	$\sin(x_1)\log(x_2) + \exp(x_3)\cos(x_4)$	$U(1, 2, 20)$
SIBC-16	$\sin(x_1^2) + \cos(x_1) + x_1 - x_1^3$	$U(1, 2, 20)$
SIBC-17	$\log(x_1^2) + x_2^3 - \frac{\exp(x_2)}{x_2}$	$U(1, 2, 20)$
SIBC-18	$\log(2x_1) + x_2\exp(x_2) - x_3\exp(x_3)$	$U(1, 2, 20)$
SIBC-19	$2x_1\sin(x_2) + \frac{\sin(x_3)}{x_3} - x_4\cos(x_4)$	$U(1, 2, 20)$

Most of the benchmarks in Nguyen and Livermore have no more than two input variables. We are interested in examining the performance of the models when the input dimension increases and the ground truth expression becomes more complex. To this end, we run further experiments on our new benchmark SIBC (shown in Table 1), which has at least three input variables.

The recovery rate performances of SIBSR, SIBSR-base, DREGS, and DSR on SIBC-1 to SIBC-10 are compared in Table 4. The results show that DSR can only recover expressions with simple linear combinations of the features (e.g. $x_1+2x_2+3_3$). For expressions with more mathematical operators like sin or log, DSR does not work. DREGS performs well on some short expressions with three variables, but its results are not good when the expression has more different types of mathematical operators. SIBSR-base performs slightly better than DREGS, but it cannot find the correct expressions for some benchmark problems (e.g. SIBC-7). SIBSR recovers most

Table 2 Recovery rate (%) for different models on Nguyen benchmark with given side information expression

Benchmark	Expression	Side Information	SIBSR	SIBSR-base	DREGS	DSR
Nguyen-1	$x^3 + x^2 + x$	x^3	100	100	100	100
Nguyen-2	$x^4 + x^3 + x^2 + x$	x^4	100	100	100	100
Nguyen-3	$x^5 + x^4 + x^3 + x^2 + x$	x^5	100	100	100	100
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	x^6	100	100	100	100
Nguyen-5	$\sin(x^2)\cos(x) - 1$	$\sin(x^2)$	100	100	100	72
Nguyen-6	$\sin(x) + \sin(x + x^2)$	$\sin(x^2)$	100	100	100	100
Nguyen-7	$\log(x+1) + \log(x^2+1)$	$\log(x^2)$	100	100	97	35
Nguyen-8	$\sqrt{x}$	x	100	100	100	96
Nguyen-9	$\sin(x) + \sin(y^2)$	$\sin(x)$	100	100	100	100
Nguyen-10	$2\sin(x)\cos(y)$	$\sin(x)$	100	100	100	91
Nguyen-11	x^y	x	100	100	100	100
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	$x^4 + \frac{1}{2}y^2 - y$	100	100	0	0
Average	-	-	100	100	91.42	82.83

Table 3 Recovery rate (%) for different models on Livermore benchmark with given side information expression

Benchmark	Expression	Side Information	SIBSR	SIBSR-base	DREGS	DSR
Livermore-1	$\frac{1}{3} + x + \sin(x^2)$	$\sin(x^2)$	100	100	100	3
Livermore-2	$\sin(x^2)\cos(x) - 2$	$\sin(x^2)$	100	100	100	87
Livermore-3	$\sin(x^3)\cos(x^2) - 1$	$\sin(x^3)$	100	100	100	66
Livermore-4	$\log(x+1) + \log(x^2+1) + \log(x)$	$log(x)$	100	100	100	76
Livermore-5	$x^4 - x^3 + x^2 - y$	$x^4 - y$	59	12	4	0
Livermore-6	$4x^4 + 3x^3 + 2x^2 + x$	x^4	100	95	88	97
Livermore-7	$\sinh(x)$	$\frac{\exp(x)}{2}$	19	0	0	0
Livermore-8	$\cosh(x)$	$\frac{\exp(x)}{2}$	15	0	0	0
Livermore-9	$x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	x^9	37	25	24	0
Livermore-10	$6\sin(x)\cos(y)$	$\sin(x)$	100	24	24	0
Livermore-11	$\frac{x^2y^2}{x+y}$	x^2y^2	100	100	100	17
Livermore-12	$\frac{x^5}{y^3}$	$\frac{x}{y}$	100	100	100	61
Livermore-13	$x^{\frac{1}{3}}$	x	100	100	100	55
Livermore-14	$x^3 + x^2 + x + \sin(x) + \sin(x^2)$	$sin(x^2)$	100	100	100	0
Livermore-15	$x^{\frac{1}{5}}$	x	100	100	100	0

Table 3 (continued)

Benchmark	Expression	Side Information	SIBSR	SIBSR-base	DREGS	DSR
Livermore-16	$x^{\frac{2}{3}}$	x	100	100	92	4
Livermore-17	$4\sin(x)\cos(y)$	$\sin(x)$	100	69	68	0
Livermore-18	$\sin(x^2)\cos(x) - 5$	$\sin(x^2)$	100	89	56	0
Livermore-19	$x^5 + x^4 + x^2 + x$	x^5	100	100	100	100
Livermore-20	$\exp(-x^2)$	$\exp(x)$	100	100	100	98
Livermore-21	$x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	x^8	39	24	24	2
Livermore-22	$\exp(-0.5x^2)$	$\exp(x)$	100	93	84	3
Average	-	-	84.95	74.13	71.09	30.41

Table 4 Recovery rate (%) for different models on SIBC-1 to SIBC-10 benchmarks with given side information expression

Benchmark	Expression	Side Information	SIBSR	SIBSR-base	DREGS	DSR
SIBC-1	$\sin(x_1^2) + x_2 - \sin(x_3)$	$\sin(x_1^2)$	100	87	67	0
SIBC-2	$\sin(x_1) + 2\sin(x_2) - \cos(x_3)$	$\cos(x_3)$	83	80	78	0
SIBC-3	$\sin(x_1) + 2\sin(x_2) + 3\sin(x_3)$	$\sin(x_1)$	100	96	91	0
SIBC-4	$x_1x_2 + x_3$	x_1	100	100	100	100
SIBC-5	$x_1 + 2x_2 + 3x_3$	x_1	100	100	100	100
SIBC-6	$\log(x_1) + 2\log(x_2) + 3\log(x_3)$	$\log(x_1)$	100	100	100	0
SIBC-7	$\log(x_1 + 3) + \log(x_2 + 2) + \log(x_3 + 1)$	$\log(x_1 + 3)$	28	0	0	0
SIBC-8	$x_1x_2+\sin(x_2)+\log(x_3)$	$\log(x_3)$	100	100	95	0
SIBC-9	$\sin(x_1) + \log(x_2) + \exp(x_3) + 2x_4$	$\sin(x_1)$	38	29	27	0
SIBC-10	$\sin(x_1)\cos(x_2) + \log(x_3)$	$\log(x_3)$	77	51	43	0
Average	-	-	82.6	74.3	70.1	20

three-variable expressions even with a basic side information given (e.g. $\sin(x_1)$), and it achieves a recovery rate of 28% on SIBC-7, which none of the other models can recover. Overall, the recovery rates are SIBSR > SIBSR-base > DREGS > DSR.

6.4 Runtime comparison on fully recovered expressions

SIBSR, SIBSR-base, and DREGS models all implement GP in their search for expressions, so their runtime is comparable on benchmark problems where all three models have a 100% recovery rate. We record the time these models take when running experiments on the Nguyen benchmarks with the given side information in Table 2. The comparison results are shown in Table 5. The DREGS model runtime is less than the model in Mundhenk et al. (2021) because the hyper-parameter settings and machines are different. Our experiments are conducted on machine with two GeForce GTX 1080 Ti GPUs. From these results, we observe that on average, SIBSR and SIBSR-base find the correct expression faster than DREGS. For the benchmark problems that DREGS solves quickly (less than 60 s), all three models tend to take similar amounts of time. For the benchmark problems that DREGS takes more than 100 s to solve, SIBSR and SIBSR-base save more time. In terms of average runtime performance, SIBSR performs the best.

Table 5 Single-core runtimes comparison on the Nguyen benchmark

Benchmark	SIBSR (sec)	SIBSR-base (sec)	DREGS (sec)
Nguyen-1	19.72	15.35	13.05
Nguyen-2	22.32	21.06	18.07
Nguyen-3	68.97	60.13	61.41
Nguyen-4	65.00	86.29	128.27
Nguyen-5	67.09	70.66	297.31
Nguyen-6	18.08	26.89	12.77
Nguyen-8	84.15	104.18	113.13
Nguyen-9	14.16	14.34	15.88
Nguyen-10	60.90	85.81	50.92
Nguyen-11	13.01	12.83	12.85
Average	39.4	45.24	65.79

6.5 Impact of side information

We observe that for the SIBC-7 and SIBC-9 benchmarks in Table 4, SIBSR has a recovery rate of less than 50% in finding the ground truth expression with the given side information. We notice that the SIBC-7 and SIBC-9 benchmarks have either more input variables or longer expressions than the others. However, the side information expressions given to these benchmarks are much different to the ground truth expression (The sentence similarity values are high for the side information and ground truth expression). We are interested in whether the given side information affects the performance of SIBSR in terms of recovery rate. Therefore, we conduct another series of experiments on SIBC-11 to SIBC-15, which have more variables and more complicated ground truth expressions. For each of these benchmark problems, different side information expressions are given with a sentence similarity value $ss(T_{true}, H)$ ranging from 0 to 1, where T_{true} denotes the ground truth expression of the benchmark. If the given side information H is exactly the same as T_{true}, based on the definition of sentence similarity, the $ss(T_{true}, H)$ value is 0. If the given side information H is totally different from T_{true}, then the $ss(T_{true}, H)$ value is 1 according to the definition. Note that our SIBSR-base method uses the same optimization objective function as SIBSR, which is $w_1/[1 + NRMSE(T)] + w_2/[1 + d(T, H)] + w_3/[1 + ss(T, H)]$. Therefore, we run these experiments on both the SIBSR-base method and SIBSR to compare the results.

The recovery rate curves for SIBSR and SIBSR-base are shown in Fig. 6. In this figure, we observe that the blue curve (SIBSR) and red curve (SIBSR-base) only overlap when $ss(T_{true}, H)$ is 0 or 1. This is because 0 means the ground truth is provided as side information, and 1 means that both models are downgraded to DREGS. Besides these two overlapped points, the blue curve always stays above the red curve, indicating that for any given side information expression, SIBSR performs better than the SIBSR-base method in terms of recovery rate.

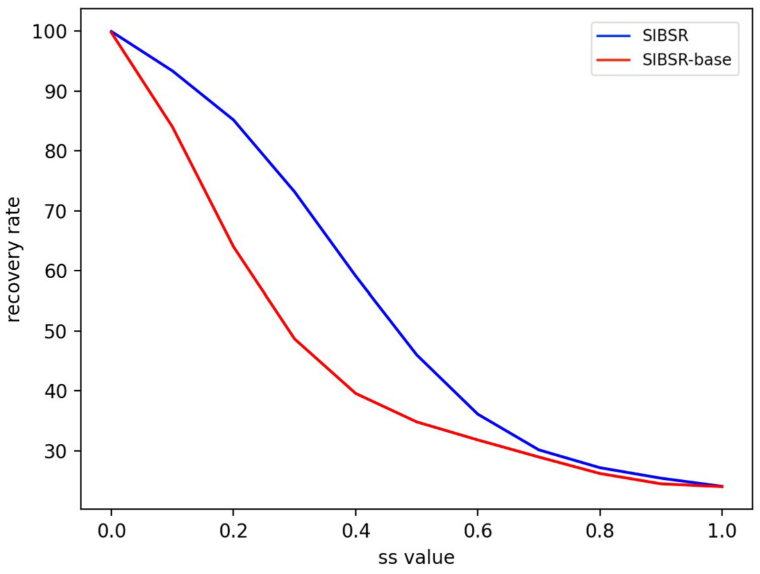


Fig. 6 SIBSR (blue) vs SIBSR-base method (red) in recovery rate for different side information expressions being provided to SIBC-11 to SIBC-15 benchmarks. Sentence similarity (ss) value from 0 to 1 denotes the similarity between the side information expression and the ground truth expression from high to low (color figure online)

The runtime curves of SIBSR and SIBSR-base are illustrated in Fig. 7. In this figure, the blue curve (SIBSR) initially takes the same amount of time as the red curve (SIBSR-base) when $ss(T_{true}, H)$ is equal to 0. As the ss value increases, the blue curve starts to increase more quickly than the red curve, indicating that SIBSR takes a longer runtime. Then, these two curves cross when the ss value is approximately 0.15, after which the red curve increases faster than the blue curve. Once the ss value is close to 1, these two curves become close to each other again, indicating that SIBSR and SIBSR-base take similar amounts of time if the Side Information Expression is significantly different from the ground truth expression.

6.6 SIBSR's advantages in data analysis for benchmarks

Through these experiments, we demonstrate SIBSR's adaptability and efficiency in complex data analysis scenarios, particularly in cases involving multiple variables and intricate mathematical expressions. This aligns with the growing need for advanced data analysis tools in various fields, emphasizing SIBSR's potential as a valuable asset in modern data science.

7 Experiments on game theory

In this section, we construct various Game Theory problems and apply our SIBSR and Side Information Generator to find the corresponding symbolic expressions.

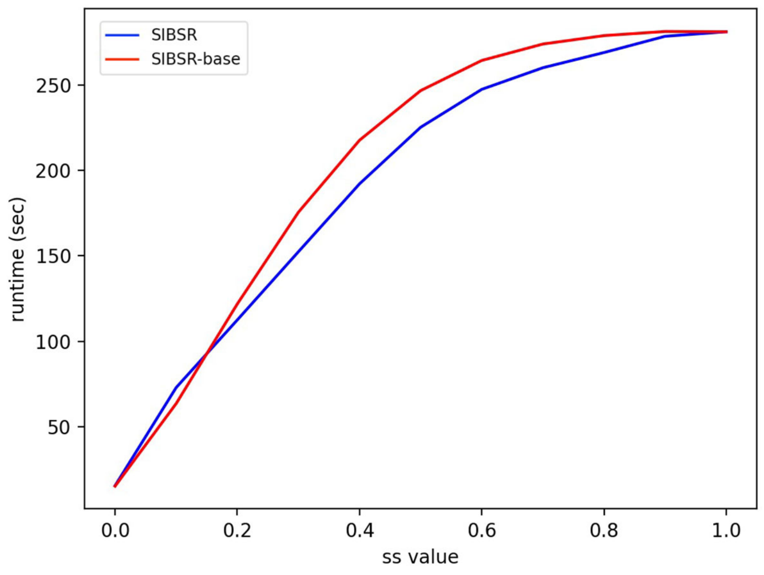


Fig. 7 SIBSR (blue) vs SIBSR-base method (red) in runtime for different side information expressions being provided to SIBC-11 to SIBC-15 benchmarks. The maximal time is set to 300 s regardless if it successfully found the expression or not. Sentence similarity (ss) value from 0 to 1 denotes the similarity between the side information expression and the ground truth expression from high to low (color figure online)

In Game Theory, Nash Equilibrium describes a state of a game involving two or more players in which no player can improve their payoff by deviating from the equilibrium strategy (Nash 1951; Avis et al. 2010). In other words, each player's strategy is the best response to the strategies of the other players at a Nash Equilibrium point.

Consider a game with N players, each possessing a strategy set S_i for $i = 1, 2, \cdots, N$. A player i's strategy is denoted by s_i, and s_{-i} represents the strategies of the other $N-1$ players. The expected payoff for player i is $u_i(s_i, s_{-i})$, considering all players adopt strategies $s = \{s_i, s_{-i}\}$. Nash Equilibrium s^* is achieved when no player can unilaterally improve their payoff, i.e., $u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$ for all $s_i \in S_i$.

7.1 Bi-matrix game definition

Our study focuses on the two-player non-cooperative game (or bi-matrix game) (Avis et al. 2010; Abbott and Kane 2004). We define a bi-matrix game $G = (A, B)$, where

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \quad a_{ij} \in \mathbb{R}$$

		Player B			
		Strategy 1	Strategy 2	$\cdots$	Strategy n
Player A	Strategy 1	(a_{11}, b_{11})	(a_{12}, b_{12})	$\cdots$	(a_{1n}, b_{1n})
	Strategy 2	(a_{21}, b_{21})	(a_{22}, b_{22})	$\cdots$	(a_{2n}, b_{2n})
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	Strategy n	(a_{n1}, b_{n1})	(a_{n2}, b_{n2})	$\cdots$	(a_{nn}, b_{nn})

Fig. 8 Bi-matrix game payoff matrix

and

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{pmatrix}, \quad b_{ij} \in \mathbb{R}$$

are the payoff matrices for players A and B, respectively. Each matrix is an $n \times n$ matrix, indicating n strategies for each player. The Nash Equilibrium strategy vectors $x = (x_1, x_2, \cdots, x_n)$ and $y = (y_1, y_2, \cdots, y_n)$ define the players' strategy probabilities, with each component $x_i, y_i \in [0, 1]$. The expected payoffs for players A and B are $x^T Ay$ and $x^T By$, respectively.

For a bi-matrix game as in Fig. 8 that merges matrices A and B together, the Nash Equilibrium, denoted by $\{x^*, y^*\}$, satisfies:

$$\begin{aligned} x^{*T} Ay^* &\geq x^T Ay^*, \quad \forall x \in \mathcal{X}, \\ x^{*T} By^* &\geq x^{*T} By, \quad \forall y \in \mathcal{Y}, \end{aligned} \tag{10}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the sets of all possible strategies for players A and B, respectively.

The best response conditions for x^* and y^* are given by:

$$x_i > 0 \Rightarrow (Ay)_i = \max_{k \in N}\{(Ay)_k\}, \forall i \in \{1, \ldots, n\}, \tag{11}$$

$$y_j > 0 \Rightarrow (B^T x)_j = \max_{k \in N}\{(B^T x)_k\}, \forall j \in \{1, \ldots, n\}. \tag{12}$$

7.2 Settings on input data and side information

In our experiments, the datasets for bi-matrix games of sizes 2×2 (denoted by $D2$) and 3×3 (denoted by $D3$) are generated using the Nashpy Whitley (1994) Python package, each comprising $10,000$ samples. In this context, a "sample" refers to a unique instance of a game scenario, including specific payoff configurations for each player in the bi-matrix game. Each sample represents a different combination of strategies and their corresponding payoffs, providing a comprehensive dataset that captures a wide range of possible game outcomes. The complexity of these datasets is captured by the number of payoff variables: 8 in $D2$ and 18 in $D3$, represented in their respective matrices A and B. Consequently, the Nash Equilibrium strategy vectors x and y in $D2$ encompass 4 values, while in $D3$, they include 6 values. The task for SIBSR is to map these payoff variables to strategy values through 4 symbolic expressions for $D2$ and 6 for $D3$. The elements of matrices A and B are sampled from the interval (0, 10] .

To accommodate the symmetric properties of the strategy targets in these games, the Side Information expression H and the token library TL are specifically designed to prioritize fractional-style symbolic expressions, as suggested by initial experiments.

7.3 2 × 2 Bi-Matrix Game.

Consider a 2×2 bi-matrix game as defined in Fig. 9. Our objective is to elucidate the functional relationship between the Nash Equilibrium strategies $\{x, y\}$ and the payoff matrix elements $\{a_{ij}, b_{ij}\}$ for this game configuration.

Applying the SIBSR methodology to this 2×2 game scenario yielded expressions correlating Nash Equilibrium strategies to the payoff variables. These expressions, representing the strategy probabilities for the players, are formulated as:

$$x_1 = \frac{b_{22} - b_{21}}{b_{11} - b_{12} - b_{21} + b_{22}} \tag{13}$$

$$x_2 = \frac{b_{11} - b_{12}}{b_{11} - b_{12} - b_{21} + b_{22}} \tag{14}$$

$$y_1 = \frac{a_{22} - a_{12}}{a_{11} - a_{12} - a_{21} + a_{22}} \tag{15}$$

$$y_2 = \frac{a_{11} - a_{21}}{a_{11} - a_{12} - a_{21} + a_{22}} \tag{16}$$

The symbolic expressions derived via the SIBSR approach above are identical to the ground truth results from conventional analytical methods (Avis et al. 2010). Since 2×2 bi-matrix games are among the most basic forms in game theory, the success of symbolic regression here lays a foundation for its application to more complex games.

		Player B	
		Strategy 1	Strategy 2
Player A	Strategy 1	(a_{11}, b_{11})	(a_{12}, b_{12})
	Strategy 2	(a_{21}, b_{21})	(a_{22}, b_{22})

Fig. 9 2×2 game payoff matrix

7.4 3 × 3 Bi-Matrix Game

The complexity of a 3 × 3 bi-matrix game, as detailed in Fig. 10, escalates with the increase in the number of variables, from 8 in 2 × 2 games to 18. Despite this complexity, our SIBSR methodology effectively derived symbolic expressions for the strategy probabilities.

Specifically, for Player A's first strategy x_1, the SIBSR process produced various expressions, each reflecting different facets of the integrated side information. These expressions include:

$$x_1 = \frac{0.118}{(0.0014b_{31} + 0.0014b_{12} + 0.0014b_{32})(0.0012b_{33} + 0.001b_{11} + 0.0008b_{21}) + 0.303} \tag{17}$$

$$x_1 = \frac{0.6153}{2.5307 - 0.0031b_{23} - 0.0031b_{32} - 0.0022b_{31} - 0.0021b_{22} - 0.0020b_{21} - 0.0018b_{13}} \tag{18}$$

$$x_1 = \frac{0.6146}{2.3772 - 0.0034b_{32} - 0.0031b_{23} - 0.0023b_{22} - 0.0018b_{13} - 0.0011b_{33} - 0.0003b_{11}} \tag{19}$$

$$x_1 = \frac{0.7095}{2.6313 - 0.0033b_{32} - 0.0032b_{23} - 0.0023b_{22} - 0.0017b_{13} - 0.0013b_{33} - 0.0004b_{11}} \tag{20}$$

Our analysis indicates that the strategy probability x_i for Player A is primarily influenced by Player B's payoffs b_{ij}, and similarly, Player B's strategy probability y_j is influenced by Player A's payoffs a_{ij}. This relationship underscores a symmetric interdependence in the game's structure. By generating expressions for x_1, we can infer the probabilities for the remaining strategies. This symmetry in the strategic interactions is a key feature harnessed by our SIBSR approach, demonstrating its capability to unravel and model the complexities inherent in 3 × 3 bi-matrix games.

7.5 Advantages of SIBSR in game theory

The application of SIBSR in 2 × 2 and 3 × 3 bi-matrix games has demonstrated its notable effectiveness in deriving accurate symbolic expressions that correlate with Nash Equilibrium points and payoff variables. In 2 × 2 games, the expressions obtained through SIBSR align closely with those derived from traditional game theory methods, validating its accuracy. In the more complex 3 × 3 game scenarios, despite the heightened complexity, SIBSR has proven its robustness and adaptability, effectively modeling strategy probabilities. These applications underscore the potential of SIBSR

		Player B		
		Strategy 1	Strategy 2	Strategy 3
Player A	Strategy 1	(a_{11}, b_{11})	(a_{12}, b_{12})	(a_{13}, b_{13})
	Strategy 2	(a_{21}, b_{21})	(a_{22}, b_{22})	(a_{23}, b_{23})
	Strategy 3	(a_{31}, b_{31})	(a_{32}, b_{32})	(a_{33}, b_{33})

Fig. 10 3 × 3 game payoff matrix

to transform the analysis of strategic decision-making in Game Theory, offering a faster and more focused approach to understanding and predicting game outcomes.

8 Conclusion

In this study, the Side Information Boosted Symbolic Regression (SIBSR) model has demonstrated enhanced recovery rates and search speeds across various symbolic regression tasks, indicating its effectiveness in uncovering complex relationships within benchmark datasets, and in analyzing Nash Equilibrium strategies and payoffs in bi-matrix game scenarios. Furthermore, we present the Side Information Generator, a tool developed to autonomously generate relevant side information expressions tailored to specific datasets. This addition enhances the precision and utility of the SIBSR model in economic data analysis challenges.

However, the method here also exhibits certain limitations. While it reduces the time complexity compared to previous methods, the SIBSR model may still struggle with lengthy and complex ground truth expressions, such as $\sin(x_1) + \log(x_2) + \exp(x_3) + 2x_4$ (SIBC-9 in Table 4). For future work, we plan to enhance the method to handle longer and more complex expressions. One promising approach under consideration is leveraging the Kolmogorov-Arnold Network (KAN) (Liu et al. 2024) to focus on the most promising subspaces of symbolic expressions, followed by applying our method to accurately identify the exact expressions.

Overall, the SIBSR model contributes to the field of data-driven symbolic regression. It offers an approach for analyzing and modeling complex data relationships across various domains. The application of SIBSR in game theory, as explored in this study, highlights its potential to advance economic data analysis methodologies.

Acknowledgements This work was supported in part by NSF Project CCF-2416361. We would like to thank the reviewers and the editor-in-chief for their valuable comments and suggestions that have improved the quality of this paper.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abbott TG, Kane D (2004) On algorithms for nash equilibria

Ahn S, Kim J, Lee H, Shin J (2020) Guiding deep molecular optimization with genetic exploration. In: Proceedings of the 34th International Conference on Neural Information Processing Systems

Al-Helali BM, Chen Q, Xue B, Zhang M (2019) A genetic programming-based wrapper imputation method for symbolic regression with incomplete data. 2019 IEEE Symposium Series on Computational Intelligence (SSCI)

Al-Helali B, Chen Q, Xue B, Zhang M (2020) Genetic programming with noise sensitivity for imputation predictor selection in symbolic regression with incomplete data. In: 2020 IEEE Congress on Evolutionary Computation (CEC)

Attali J-G, Pagès G (1997) Approximation of functions by a multilayer perceptron: a new approach. Neural Netw 10:1069

Avis D, Rosenberg GD, Savani R, Stengel BV (2010) Enumeration of nash equilibria for two-player games. Econ Theor 42:9

Chen Q, Zhang M, Xue B (2017) Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. IEEE Trans Evolut Comput 21:792

Chen Q, Xue B, Niu B, Zhang M (2016) Improving generalisation of genetic programming for high-dimensional symbolic regression with feature selection. In: 2016 IEEE Congress on Evolutionary Computation (CEC)

Evolutionary Computation 1: (2000) Basic Algorithms and Operators. Institute of Physics Publishing, Bristol

Fortin F-A, De Rainville F-M, Gardner M-AG, Parizeau M, Gagné C (2012) Deap: Evolutionary algorithms made easy. J Mach Learn Res 13:2171

Genetic Programming:(1992) On the Programming of Computers by Means of Natural Selection. MIT Press

Hansen N, Auger A, Mersmann O, Tusar T, Brockhoff D (2016) COCO: A platform for comparing continuous optimizers in a black-box setting. Optim Method Softw 36(1):114–144

Hansen N, Finck S, Ros R, Auger A (2009) Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions

Kim S, Lu PY, Mukherjee S, Gilbert M, Jing L, Ceperic V, Soljacic M (2021) Integration of neural network-based symbolic regression in deep learning for scientific discovery. IEEE Transactions on Neural Networks and Learning Systems

Knight V, katiemcgoldrick Panayides M, Wang Y, Gaba AS, Konovalov O, Rivière P, Baldevia R, Jr, IF, newaijj, volume-on-max: drvinceknight/Nashpy: V0.0.34

Kusner MJ, Paige B, Hernández-Lobato JM (2017) Grammar variational autoencoder. In: Proceedings of the 34th International Conference on Machine Learning

Lample G, Charton F (2020) Deep learning for symbolic mathematics. In: International Conference on Learning Representations

Landajuela M, Petersen BK, Kim S, Santiago CP, Glatt R, Mundhenk N, Pettit JF, Faissol D (2021) Discovering symbolic policies with deep reinforcement learning. In: Proceedings of the 38th International Conference on Machine Learning

Lawrence S, Giles CL, Tsoi AC, Back AD (1997) Face recognition: a convolutional neural-network approach. IEEE Trans Neural Netw 8:98

Levenshtein V (1966) Binary Codes Capable of Correcting Deletions. Insertions and Reversals, Soviet Physics Doklady

Liang J, Qu B, Suganthan P, Hernández-Díaz A (2013) Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization. Technical Report 201212, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China

Liu Z, Wang Y, Vaidya S, Ruehle F, Halverson J, Soljacic M, Hou TY, Tegmark M (2024) KAN: Kolmogorov-Arnold Networks

Lu Q, Ren J, Wang Z (2016) Using genetic programming with prior formula knowledge to solve symbolic regression problem. Intell Neuroscience 2016:1021378

McConaghy T (2011) FFX: Fast. Scalable, Deterministic Symbolic Regression Technology

Meunier L, Rakotoarison H, Wong PK, Roziere B, Rapin J, Teytaud O, Moreau A, Doerr C (2021) Black-Box Optimization Revisited: Improving Algorithm Selection Wizards through Massive Benchmarking

Mundhenk TN, Landajuela M, Glatt R, Santiago CP, Faissol DM, Petersen BK (2021) Symbolic regression via neural-guided genetic programming population seeding. CoRR **abs/2111.00053**

Nash JF (1951) Non-cooperative games. Annals of Mathematics (2)

Navarro G (2001) A guided tour to approximate string matching. ACM Comput, Surv

Petersen BK, Larma ML, Mundhenk TN, Santiago CP, Kim SK, Kim JT (2021) Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In: International Conference on Learning Representations

Recurrent Neural Networks: Design and Applications. CRC Press (1999)

Sahoo SS, Lampert CH, Martius G (2018) Learning equations for extrapolation and control. CoRR **abs/1806.07259**

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

Udrescu S-M, Tegmark M (2020) AI Feynman: a physics-inspired method for symbolic regression. Sci Adv 6:eaay2631

Uy NQ, Hoai NX, O'Neill M, Mckay RI, Galván-López E (2011) Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genet Program Evol Mach 12:91

Whitley D (1994) A genetic algorithm tutorial. Stat Comput 4:65

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)