Information Sciences 547 (2021) 553–567

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Enhancing gene expression programming based on space partition and jump for symbolic regression

Qiang Lu [a,*], Shuo Zhou [a], Fan Tao [a], Jake Luo [b], Zhiguang Wang [a]

[a] Beijing Key Laboratory of Petroleum Data Mining, China University of Petroleum-Beijing, Beijing, China
[b] Department of Health Sciences and Administration, University of Wisconsin Milwaukee, Milwaukee, WI, United States

ARTICLE INFO

Article history:
Received 23 April 2019
Received in revised form 15 August 2020
Accepted 18 August 2020
Available online 28 August 2020

Keywords:
Symbolic regression
Gene expression programming
Genetic programming
Multi-armed bandit
Evolutionary computation

ABSTRACT

When solving a symbolic regression problem, the gene expression programming (GEP) algorithm could fall into a premature convergence which terminates the optimization process too early, and may only reach a poor local optimum. To address the premature convergence problem of GEP, we propose a novel algorithm named SPJ-GEP, which can maintain the GEP population diversity and improve the accuracy of the GEP search by allowing the population to jump efficiently between segmented subspaces. SPJ-GEP first divides the space of mathematical expressions into k subspaces that are mutually exclusive. It then creates a subspace selection method that combines the multi-armed bandit and the ϵ-greedy strategy to choose a jump subspace. In this way, the analysis is made on the population diversity and the range of the number of subspaces. The analysis results show that SPJ-GEP does not significantly increase the computational complexity of time and space than classical GEP methods. Besides, an evaluation is conducted on a set of standard SR benchmarks. The evaluation results show that the proposed SPJ-GEP keeps a higher population diversity and has an enhanced accuracy compared with three baseline GEP methods.

1. Introduction

Symbolic regression (**SR**) is a regression analysis that discovers a model that best fits a given dataset in the space of mathematical expressions. Unlike machine learning or neural network regression analysis that focuses on optimizing parameters in a predefined model, SR aims to find appropriate models and their parameters at the same time. Genetic programming (**GP**) [1] is a commonly used approach in SR to search for the optimal model. GP evolves to change individual structures of the population to generate fitted models or computer programs by the three key genetic algorithm (**GA**) operations: selection, crossover, and mutation. To represent a mathematical expression, classical GPs usually describe individual encodings in trees [1–5]. Graph-based GPs, such as graph encoding GP [6,7] and Cartesian genetic programming [8,9], encode individuals into graphs. Linear GPs, such as gene expression programming (GEP) [10–12] and linear GP [13], convert individuals into linear strings.

Since these GPs all utilize GA operations, like genetic algorithms(GA), these GPs are prone to premature convergence [14]. From the perspective of exploration and exploitation [15], the reason for premature convergence is that individuals of a population are similar, hence, they tend to exploit their neighborhood instead of new regions. Therefore, maintaining the

* Corresponding author.
E-mail address: luqiang@cup.edu.cn (Q. Lu).

https://doi.org/10.1016/j.ins.2020.08.061
0020-0255/

population diversity is a crucial task in evolutionary algorithms. A diverse population can encourage global exploration and reduce premature convergence [16,17].

In order to preserve the population diversity, the evolutionary computing (EC) community often uses two strategies: 1) parameter control and 2) space partition. The parameter control strategy [18] adjusts parameters of evolutionary algorithms based on population diversity, such as varying population size [19,20], and dynamically adjusting the probability of crossover [21,22] and mutation [23,24]. The strategy is easy to implement population diversity and does not require additional storage spaces. However, it does not know or remember where individuals are in a search space so that it could produce invalid individuals, such as individuals similar to those of the previous generations.

The space partition strategy [25–30] splits a search space into many subspaces and generates individuals in different subspaces. As individuals in different subspaces have different phenotypes or genotypes, the strategy is easy to control population diversity quantitatively by generating individuals from different subspaces. Meanwhile, the strategy remembers an individual's approximate position in the search space according to the individual's subspace. Although the space partition strategy has been successfully applied in GA, it is not suitable for the SR problem, because the whole search space of SR is so large that maintaining fine-grained subspaces is intractable computationally.

In this paper, we propose a new gene expression programming based on **s**pace **p**artition and **j**ump (named **SPJ-GEP**) to maintain the population diversity. SPJ-GEP has the advantages of the above two strategies: it requires small additional storage space, remembers the position of an individual in the search space, and maintains quantitative population diversity. The SPJ-GEP partitions the space of mathematical expressions into k subspaces based on the chromosome coding. Moreover, it initializes individuals in one of the k subspaces, as shown in Step 1 in Fig. 1.

Next, SPJ-GEP selects a suitable subspace to search for individuals with better fitnesses based on a subspace selection method that combines the multi-armed bandit (**MAB**) [31] and the ϵ-greedy strategy [32], as shown in Step 2 and 3 in Fig. 1. This method utilizes MAB to choose one of the subspaces because MAB can balance the exploration by searching other subspaces while maintaining the exploitation of the selected subspace. However, MAB will be invalid when the number of visiting subspaces is higher than a specific value. To preserve population diversity, the method then switches to the ϵ-greedy strategy to choose a subspace according to a proposed time formula. The formula decides when to use the ϵ-greedy strategy.

At last, SPJ-GEP uses a new crossover method to make individuals jump from the original subspace to another selected subspace, as shown in Step 4 in Fig. 1. The method makes these newly selected individuals intersect with the best individual in the selected subspace so that they can start searching at the latest local optimal position.

The characteristics of SPJ-GEP indicates that classical GEPs [10–12] are a special case of SPJ-GEP when the number of subspaces k equals 1. On the other hand, if k is large enough that each subspace has only one individual, SPJ-GEP will degenerate into a random selection subspace algorithm. Therefore, k is a critical parameter in SPJ-GEP. In this paper, the range of k is decided by the population diversity and the probability of jump between subspaces. We analyze the complexity of time and space of SPJ-GEP and prove that SPJ-GEP does not significantly increase the time and space complexity compared with classical GEPs.

The main contributions in the paper are summarized as follows:

- We propose the SPJ-GEP algorithm, which allows individuals in a population to jump between subspaces according to the MAB and the ϵ-greedy strategy. This approach maintains the population diversity.

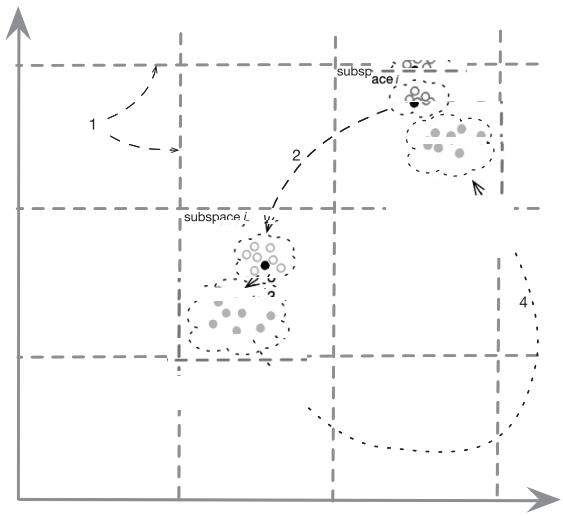


Fig. 1. The SPJ-GEP framework. Circles represent individuals, and dark circles are the best individuals in subspaces. Step 1. space segmentation; Step 2. subspace selection; Step 3. subspace exploitation; Step 4. escape from local optimum by subspace jump.

- We provide a solid analysis approach to evaluate the range of the number of subspaces k, and we analyze the algorithm's time and space complexity.
- Our evaluation results show that SPJ-GEP surpasses the three baseline GEP methods: GEP[10], GEP-ADF[11], and self-learning GEP [12].

The rest of this paper is organized as follows. Section 2 describes related background techniques. Then, Section 3 and 4 provide details for the proposed SPJ-GEP algorithm and its analysis, respectively. Moreover, Section 5 shows the experimental results and analysis. Section 6 discusses related works. Section 7 concludes the paper and points out possible future work.

2. Background

2.1. Gene expression programming for symbolic regression

For a given dataset $\{X, Y\}$, the goal of symbolic regression is to discover a function $f(X) = Y\prime$, which can minimize the error between $Y\prime$ and Y, from the space of mathematical expression that consists of **function symbols** (e.g., $+, -, \times, /, sin, cos, \ldots$) and **terminal symbols** (e.g., variables and coefficients). In order to find the best fit function, GP encodes an individual ($f(X)$) into a tree (as shown in Fig. 2), and applies crossover and mutation to change the individuals for the optimizing function f.

Unlike GP, the gene expression programming (GEP) algorithm [10] encodes an individual to a linear structure of fixed length. The linear structure includes a head and a tail. The head consists of function symbols and terminal symbols, while the tail consists of terminal symbols. As shown in Fig. 2, for the two individuals: '$++/-\times xxyyxy$' and '$\times xx\times +xyyxxy$', the two underlined sections: '$++/-\times$' and '$\times\ xx\ \times\ +$' are the heads. The other two sections: '*xxyyxy*' and '*xyyxxy*' are the tails. The head is part of the prefix expression and contains all function symbols; the tail provides variables or coefficients for the head. Besides, operations in GEP are similar to those in GA, such as crossover in Fig. 2. By exchanging the substring '++' in '$++/-\times xxyyxy$' and the substring '$\times x$' in '$\times xx\times +xyyxxy$', the crossover obtains two new individuals: '$++x\times +xyyxxy$' and '$\times x/-\times xxyyxy$'.

Moreover, some extended GEPs [11,33] that incorporate high-order knowledge have been proposed to improve the search's performance on large-scale applications. For example, each chromosome in GEP-ADF [11] consists of multiple conventional genes (called ADFs) and a homeotic gene. Each conventional gene is a prefix subexpression to describe a sub-solution, while the homeotic gene combines these conventional genes to provide a complete solution to a particular application.

2.2. Multi-armed bandit problem

A multi-armed bandit (MAB) is the problem to find the best way to maximize a cumulative reward earned by pulling one of K one-armed bandits that have unknown reward distributions [34]. In order to maximize the cumulative reward, MAB must balance between exploring more arms to improve the estimates of rewards, and exploiting the current best arm. Many

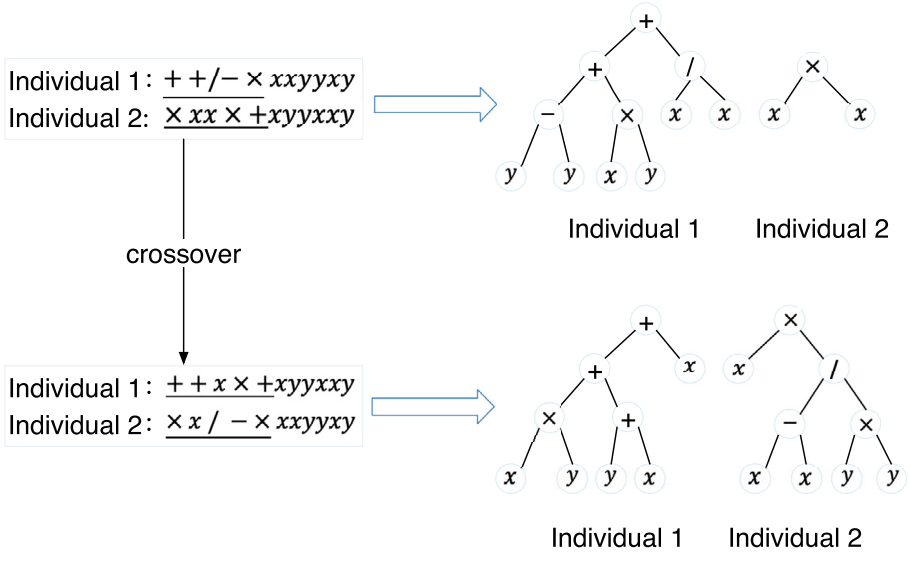


Fig. 2. Individual coding and crossover in GEP.

arm-selection strategies, such as ϵ-greedy[35], Boltzmann exploration [36], and upper-confidence bound (UCB) [31], are proposed to keep the exploration and exploitation balance. In the paper, we use the UCB1 as the arm-selection strategy according to Eq. 1.

$$UCB_1 = \mu_k + \lambda\sqrt{\frac{2lnt}{n_k}} \tag{1}$$

where $0 < \lambda \leqslant 1, t$ is the number of times of pulling all arms, μ_k is the mean reward of the arm k after it has been pulled n_k times. The reward term μ_k encourages the exploitation of higher-reward arms. The $\sqrt{\frac{2lnt}{n_k}}$ is the size of the confidence interval for the reward of the arm k, which encourages the exploration of less-visited arms.

3. Gene expression programming based on space partition and jump

The gene expression programming based on space partition and jump (SPJ-GEP) has four components: space partition, subspace selection, crossover, and mutation, as shown in Algorithm 1. Its frequently used notations are listed in Table 1.

SPJ-GEP first executes the space partition components (line 1) to split the mathematical expression space Ω into k subspaces $\{\omega_1, \ldots, \omega_k\}$. Then, according to UCB or ϵ-greed method (line 6–9), it runs the subspace selection strategy to choose a subspace ω_i for exploring Ω. At last, it calls crossover and mutation to exploit the subspace ω_i (line 10–11). Since the mutation is the same as the mutation in other GEPs [10–12], we will omit the detailed description of the mutation in the following subsections.

Algorithm 1 SPJ-GEP

Input: k, $n = |population|$, G, α, β, ϵ

```
1  partition_space(k);
2  g, t ← 0;
3  T ← G × n;
4  while (g < G) or (not find best results) do
5      //select a subspace ω_i according to THEOREM 1
6      if t < −2klnT / (α²lnβ) then
7          select a subspace ω_i* that has the maximal UCB by equation 2;
8      else
9          select a subspace ω_i* by the ϵ-greedy method;
10     crossover(ω_i) //exploit the subspace ω_i;
11     mutation(ω_i) ;
12     g++;
13     t ← g × n ;
```

3.1. Space partition

3.1.1. Encoding

Let a mathematical expression space be Ω. If the space can be partitioned into many subspaces, denoted as $\{\omega_1, \omega_2, \ldots, \omega_n\}$, then these subspaces that meet the two following conditions constitute a **space-partition set**.

- mutually exclusive: $\omega_i \bigcap \omega_j = \phi, i \neq j$
- completeness: $\Omega = \bigcup_{i=1}^{n} \omega_i$

Table 1
Notation.

Notation	Definition
Ω	The mathematical expression space
k	The number of subspaces in Ω
ω_i	The ith subspace
n	The number of individuals in the population
G	The iteration times in SPJ-GEP
α	The threshold value of the confidence interval $\sqrt{\frac{2lnt}{n_k}}$
β	The threshold value of the probability that UCB_{ω_i} loses its effect
ϵ	The parameter in the ϵ-greedy method
l	The length of an individual code
h	The head length of an individual code

According to the coding of individuals in GEP, i.e., encoding an individual to a linear structure with the fixed-length l and the head length h, the space of mathematical expression is denoted as $\Omega_{l,h} =$ '$*\ldots**\ldots*$'. l is the length of individuals (the total number of '$*$'); h is the head length (the number of '$\underline{*}$'); '$*$' can be anyone symbol from a symbol set $\boldsymbol{S}$ that consists of a function set $\boldsymbol{F}$ and a terminal set $\boldsymbol{T}$. In the head, if front '$*$'s are replaced by special symbols s, it can generate a **subspace** ω_s, such as $\omega_+ =$ '$+****$' in Fig. 3. Therefore, the more special symbols appear in the front of the head, the smaller the subspace's size becomes. For example, $\omega_{++} =$ '$\underline{++}***$' is a subspace of ω_+, and $\omega_{++} \subset \omega_+$.

Based on the above subspace encoding, these subspaces and their relationships can be represented as **a space-partition tree**, where the root node is $\Omega_{l,h}$, each of the other nodes is a subspace of $\Omega_{l,h}$, and a branch represents a containment relationship between two subspaces, e.g., $\omega_{++} \subset \omega_+$ as shown in Fig. 3. From the tree, a lot of space-partition sets can be found based on the above two conditions. For example, $\{\omega_+, \omega_-, \omega_x\}$ and $\{\omega_{++}, \omega_{+-}, \omega_{+x}, \omega_+, \omega_x\}$ both are space-partition sets.

3.1.2. Initialization

SPJ-GEP selects a space-partition set by the following strategy. It can easily find the first level ll where the number of nodes is equal or greater than the number of subspaces (k) according to $ll \geqslant log^k_{|S|}$, where $|S|$ is the number of symbols. For example, in Fig. 3, if $k = 5$ and $|S| = 3$, then $ll = 2$. Moreover, the algorithm discovers a space-partition set $\{\omega_{++}, \omega_{+-}, \omega_{+x}, \omega_{-+}, \omega_{--}, \omega_{-x}, \omega_x\}$. Then, for each subspace ω_i in the set, SPJ-GEP randomly generates an individual and computes UCB_{ω_i} according to Eq. 2.

3.2. Subspace selection

SPJ-GEP uses modified UCB1 to select a subspace from a space-partition set. When the modified UCB1 becomes invalid in most of the subspaces, SPJ-GEP then uses the ϵ-greedy method to choose a subspace.

3.2.1. Subspace selection based on UCB1

For a space-partition set $\{\omega_1, \omega_2, \ldots, \omega_k\rangle$, it represents the space of mathematical expressions. Moreover, any individual that SPJ-GEP generates must be in one of these subspaces. If each subspace is seen as an arm in the multi-armed bandit (MAB), and generated individuals in the subspace are considered as pulling the arm. SPJ-GEP faces the same problem as MAB does: how to balance between exploration and exploitation of these subspaces.

The difference between SPJ-GEP and MAB is that SPJ-GEP wants to obtain an individual with maximal fitness. Whereas, MAB wants to obtain a sequence of individuals so that their cumulative fitness is maximal. So, we modify Eq. 1 to Eq. 2 as the subspace selection method.

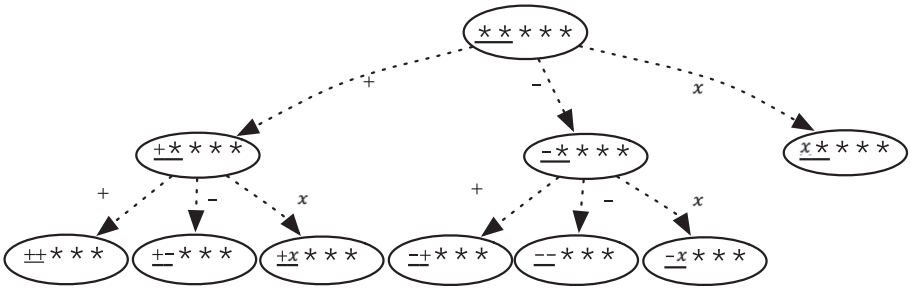


Fig. 3. A space-partition tree. The function set $F = \{+, -\}$ and the terminal set T={x}.

$$UCB_{\omega_i} = \frac{1}{f^*_{\omega_i} + 1} + \lambda\sqrt{\frac{2lnt}{n_{\omega_i}}} \tag{2}$$

where ω_i is a subspace, $f^*_{\omega_i}$ is the fitness of the best individual in ω_i, t is the number of visiting Ω until a particular time, and n_{ω_i} is the number of times that the subspace ω_i is accessed to. Then, SPJ-GEP selects the best subspace ω_{i^*} that has the maximal *UCB* as its exploration space.

3.2.2. Subspace selection based on ϵ-Greedy method

As visit times increase in a subspace, the size of the confidence interval ($\sqrt{\frac{2lnt}{n_i}}$ in Eq. 2) decrease to zero. That means UCB_{ω_i} falls back to a greedy method with the subspace value ($f^*_{\omega_i}$) and becomes invalid in the balance between exploration and exploitation.

To overcome the above invalidation, SPJ-GEP uses the ϵ-greedy method [35] to select a subspace when confidence intervals in most of the subspaces tend to be zero. Using Eq. 2 chooses a subspace with the probability $1-\epsilon$; random chooses a subspace from the above with the probability ϵ.

3.2.3. The time of using the ϵ-Greedy Method

To find out when UCB_{ω_i} (Eq. 2) loses its effect on most subspaces, SPJ-GEP uses Eq. 3 in the following Theorem 1. For example, given $k = 300, n = 100, G = 100000, \alpha = 0.1$ and $\beta = 0.8$, to choose a subspace ω_i, it uses Eq. 2 if $t < -\frac{2\times 300\times ln^T}{0.01^2\times ln^{0.8}} \approx 4333918$, where $T = n \times G = 100 \times 10000 = 10^7$; otherwise, it uses ϵ-greedy method.

Theorem 1. *Let k be the number of subspaces, and T be the total number of times of visiting the mathematical expressions space Ω after running SPJ-GEP. If the confidence interval $\sqrt{\frac{2ln^t}{n_{\omega_i}}} \leqslant \alpha$, where $\alpha \to 0^+$, UCB_{ω_i} will lose its effect on subspace ω_i. Assuming that different n_{ω_i} is independent identically distributed (i.i.d.), and each n_{ω_i} is an exponential distribution with the parameter λ, the probability that each UCB_{ω_i} loses its effect is greater than β when the number of times that Ω is accessed to.*

$$t \geqslant -\frac{2kln^T}{\alpha^2 ln^\beta}. \tag{3}$$

Proof. According to the constant T and $\sqrt{\frac{2ln^t}{n_{\omega_i}}} \leqslant \alpha$, we have $n_{\omega_i} \geqslant \frac{2ln^T}{\alpha^2}$ so that UCB_{ω_i} are invalid in ω_i. Therefore, if we want to assure that the probability, which each UCB_{ω_i} loses its effect, is greater than β, i.e., $P(n_{\omega_i} \geqslant \frac{2ln^T}{\alpha^2}) \geqslant \beta$, the following equation

$$\lambda \leqslant -\frac{\alpha^2 ln^\beta}{2ln^T} \tag{4}$$

must be satisfied, because

$$P(n_{\omega_i} \geqslant \tfrac{2ln^T}{\alpha^2}) = \int_{\frac{2ln^T}{\alpha^2}}^{\infty} \lambda e^{-\lambda n_{\omega_i}} dn_{\omega_i} = e^{-\lambda\frac{2ln^T}{\alpha^2}}$$
$$\Rightarrow e^{-\lambda\frac{2ln^T}{\alpha^2}} \geqslant \beta.$$

As different n_{ω_i} is i.i.d. and each n_{ω_i} is an exponential distribution with the parameter λ, we get

$$t = E[\sum_{i=1}^{k} n_{\omega_i}] = E[E[\sum_{i=1}^{k} n_{\omega_i}|k]] = E[k]E[n_{\omega_i}] = \frac{k}{\lambda}. \tag{5}$$

According to formulas 4 and 5, we finally obtain formula 3.

3.3. Exploitation with crossover

Suppose that the current population is at the subspace ω_j, after SPJ-GEP selects a subspace ω_i, it makes all individuals in the population **jump** from the subspace ω_j to the subspace ω_i. As the aforementioned subspace encoding, codes of these individuals in ω_j start with the code of ω_j. So, for making them jump, it is necessary to replace their head codes with the code of ω_i. For example, given two individuals '$\underline{++}/-\times xxxxxx$' and '$\underline{++}+\times\times xxxxxx$' in the subspace ω_{++}, the two individuals will jump into the subspace ω_{+-} after they change to '$\underline{+-}-\times xxxxxx$' and '$\underline{+-}+\times\times xxxxxx$' by replacing '++' in their heads with the code '+−' of 'ω_{+-}'.

Then, it exploits ω_i by recombining each of the transferred individuals with the best individual in the subspace ω_i. For example, if the best individual is '$\underline{+-}\times++xxxxxx$', the above two individuals '$\underline{+-}-\times xxxxxx$' and '$\underline{+-}+\times\times xxxxxx$' recom-

bine with the best individual, respectively. The recombination makes the jumped population start to search from the local optimal space, and speeds up the convergence.

Note that if the selected subspace ω_j is equal to ω_i, the recombination is the same as the recombination of classical GEPs [10–12] in that any two individuals in the population recombine randomly. So, if the above subspace selection continuously chooses the same subspace, SPJ-GEP will exploit the subspace persistently.

4. Analysis of SPJ-GEP

4.1. Time and space complexity

Compared with classical GEPs [33,37,38], SPJ-GEP requires additional structures to record the visiting times (n_{ω_i}) and the best fitness ($f^*_{\omega_i}$), as well as extra computation to obtain UCB_{ω_i} in each subspace. The additional time and space complexity are related to the number (k) of all subspaces. Suppose the time and space complexities for classical GEPs within g iterations are $O(gep)$ and $\Theta(gep)$, respectively. For SPJ-GEP, they are $O(gep + g \times c \times k) = O(gep + n \times k)$ and $\Theta(gep + m \times k)$, where $n = c \times k, k > 1$, and g, c, and m are constants. Therefore, if k is within a reasonable range, its time and space complexities are acceptable. In our experimental evaluation, its running time is almost as fast as GEP's because the value of k is generally not very large (detail in Section 4.3).

4.2. Population diversity

To preserve the population diversity, SPJ-GEP always lets the population jump from one subspace to another subspace. Even if an individual in a population immediately jumps back to its original subspace after two jumps, its structure has been significantly changed according to the following Lemma 1.

Lemma 1. *Suppose SPJ-GEP executes single-point crossovers with uniform distribution, and l is the length of an individual encoding without considering its subspace encoding. After the individual has jumped k subspaces, the similarity between the jumped individual, and the original individual is*

$$sim(k) = \left(\frac{1+l}{2l}\right)^k \tag{6}$$

Proof. Since the crossover point is randomly selected with uniform distribution, after a crossover, the expected length of the original fragment in the new individual is $(1 + l)/2$. Then, the similarity between the original individual and the jumped individual is $(1 + l)/2l$ after one crossover. Therefore, after k crossovers in k subspaces, the similarity is $sim(k) = (\frac{1+l}{2l})^k$.

For example, if the length of an individual is 20, after jumping only two subspaces, the similarity between original and jumped individuals changes to 0.276. As k increases, $sim(k)$ tends to be zero. Moreover, the two individuals become more different. Therefore, the subspace selection method, which lets the population jump from a subspace to another subspace, diversifies the population so that it helps prevent a local optimum in SPJ-GEP.

4.3. The number of subspaces

The number of subspaces k is a critical parameter in SPJ-GEP. If $k = 1$, it means that there is only one subspace in Ω. So, when the subspace is Ω, SPJ-GEP degenerates into a standard GEP. If k is large enough that there is only one individual in a subspace, each subspace is an individual. In this case, SPJ-GEP degenerates into the random initialization algorithm, randomly generating an individual (i.e., a subspace) in Ω. Therefore, if k is too large or too small, the algorithm performance will be degraded.

According to Lemma 1, the larger k, the smaller the similarity. Then, SPJ-GEP needs a smaller similarity to escape from the local optimum. Based on the following Theorem 2, we have the lower bound of k. Besides, the larger k, the smaller the probability that an individual jumps in its original subspace. The probability must be larger enough that SPJ-GEP can exploit a subspace continuously for a while. Otherwise, SPJ-GEP will always explore a different subspace that breaks the balance between exploration and exploitation. So, based on the following Theorem 3, we have the implicit expression of the upper bound of k.

Theorem 2. *Suppose an individual returns to the original subspace after it jumps k subspaces, in order to guarantee the similarity between the original individual and the new back individual is less than or equal to η, the lower bound of k is $\frac{ln^\eta}{ln^{1+l} - ln^{2l}}$.*

Proof. According to Eq. 6 in Lemma 1, $(\frac{1+l}{2l})^k \leqslant \eta$. Then, we have

$$k \geqslant \frac{ln^{\eta}}{ln^{1+l} - ln^{2l}} \tag{7}$$

When the running time of SPJ-SEP exceeds a specific time, SPJ-SEP uses the ϵ-greedy strategy to select a subspace according to Theorem 1. Most of the subspaces are selected randomly. So, the jump probability $P(i,j)$ between subspace i and j satisfies a long tail distribution. Moreover, the probability that the subspace with the best UCB_{ω_i} will be selected again is $1-\epsilon$, and the probability is higher than the probability of selecting other subspace, i.e., $P(i,i) > P(i,j)$. If SPJ-SEP uses the best UCB_{ω_i} (MAB) to select a subspace, the same conclusions $P(i,j)$ satisfies a long tail distribution and $P(i,i) > P(i,j)$ can be obtained through an analysis similar to the above.

Theorem 3. *Suppose the jump probability $P(i,j)$ between subspace i and j satisfies the Zipf distribution [39], in order to guarantee that $P(i,i) \geqslant \delta$, the number of subspace k satisfies the following inequality.*

$$\frac{1}{\sum_{i=1}^{k} \left(\frac{1}{i}\right)^{\gamma}} \geqslant \delta \tag{8}$$

Proof. Since $P(x_{ij})$ satisfies Zipf distribution, whose probability mass function of Zipf is $f(x) = \frac{1}{x^{\gamma} \sum_{i=1}^{k} (1/i)^{\gamma}}$, where $x = 1, 2, \ldots, k, k$ is the number of subspace, and γ is a parameter.

$$p(i,i) = f(1) = \frac{1}{\sum_{i=1}^{k} (1/i)^{\gamma}} \geqslant \delta \tag{9}$$

owing to $f(1) > f(x)$ when $x \neq 1$.

For example, if $\gamma = 0$, the Zipf distribution will degenerate into a uniform distribution. According to Eq. 8, $\frac{1}{k} \geqslant \gamma$. So, $k \in \left[\frac{ln^{\eta}}{ln^{1+l} - ln^{2l}}, \frac{1}{\gamma}\right]$.

Table 2
GP Problems.

Name	Formula	Dataset
F1	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	U[−1, 1, 20]
F2	$x^4 + x^3 + x^2 + x$	U[−1, 1, 20]
F3	$x^5 - 2x^3 + x$	U[−1, 1, 20]
F4	$sin(x^2)cos(x) - 1$	U[−1, 1, 20]
F5	$sin(x) + sin(x + x^2)$	U[−1, 1, 20]
F6	$ln(x + 1) + ln(x^2 + 1)$	U[0, 2, 20]
F7	$2sin(x)cos(y)$	U[−1, 1, 100]
F8	$1.57 + (24.3v)$	U[−50, 50, 10000]
F9	$6.87 + 11cos(7.23x^3)$	U[−50, 50, 10000]
F10	$2 - 2.1cos(9.8x)sin(1.3w)$	U[−50, 50, 10000]
F11	$0.3xsin(2 - x)$	E[−1, 1, 0.1]
F12	lnx	E[1,1]
F13	x^y	U[0, 1, 100]
F14	$x^4 - x^3 + \frac{y^2}{2} - y$	U[−3, 3, 20]
F15	$\frac{x^3}{5} + \frac{y^3}{2} - y - x$	U[−3, 3, 20]
F16	$e^{-x}x^3(cosxsinx)(cosxsin^2x - 1)$	E[0.05, 10, 0.1]
F17	$\frac{e^{-(x-1)^2}}{1.2+(y-2.5)^2}$	U[0.3, 4, 100]
F18	$e^{-x}x^3(cosxsinx)(cosxsin^2x - 1)$ $*(y - 5)$	x:E[0.05, 10, 0.1] y:E[0.05, 10.05, 2]
F19	$(x - 3)(y - 3) + 2sin((x - 4)(y - 4))$	U[0.05, 6.05, 300]
F20	$\frac{(x-3)^4+(y-3)^3-(y-3)}{(y-2)^4+10}$	U[0.05, 6.05, 50]

The function sets of $F_1 - F_7, F_8 - F_{10}, F_{11} - F_{15}$ and $F_{16} - F_{20}$ are from Koza [1], Korns [41], Keijzer [42] and Vladislavleva [43], respectively.

5. Experiments

5.1. Dataset and experimental parameters

In this paper, the dataset consists of 20 SR test problems that are derived from the GP benchmarks [40], as shown in Table 2. The functions and constants of the data set are shown in Table 3. To evaluate the proposed algorithm **SPJ-GEP**, we have created three algorithms SPJ-GEP, SPJ-GEP-ADF, and SPJ-SL-GEP based on the three baseline **GEPs**: GEP [10], GEP-ADF [11], and SL-GEP [12], respectively. The three new algorithms have the same parameters as these GEPs have except for the additional parameters k, α, and β. The detailed parameters of the above six algorithms are described in Table 4.

We set the number of subspaces k a particular value according to the number of nodes on a specific layer in the space-partition tree. Since the evaluation consists of six basic benchmarks, and each basic benchmark has different function symbols and terminal symbols, SPJ-GEP, SPJ-GEP-ADF, and SPJ-SL-GEP have different k values for different basic benchmarks. In Table 4, the last row shows the range of k is [144–1000]. For example, in the Koza basic benchmark, there is a function symbol set $\{+, -, \times, /, sin, cos, ln(|x|), e^x\}$, whose length is 8, and a terminal set $\{x_1, x_2\}$, whose length is 2. When a space-partition set is obtained by nodes on the 3rd layer in the space-partition tree, the number of the subspaces k is $(8+2)^3 = 1000$.

5.2. Verification of subspaces selection method

Looking back at the subspace selection in SPJ-GEP, Inequality 3 is a key to decide when the UCB (Eq. 2) loses its effect on selecting subspaces. To verify the accuracy and the correctness of Inequality 3, we run SPJ-GEP with different values of parameter k (100,200 and 300) on five test problems. When $t \geqslant -\frac{2kln^T}{\alpha^2 ln^\beta}$, where $T = 100000 \times 100$, such as $t \geqslant \frac{2\times 100\times ln^{10^7}}{(0.1)^2\times ln^{0.8}} \approx 1444639$, the algorithm stops. Then, we sum up the number of subspaces where UCB has lost its effect according to the following Inequality

$$\sqrt{\frac{2ln^T}{n_{\omega_i}}} \leqslant \alpha \tag{10}$$

where n_{ω_i} is the number of times that the subspace ω_i is accessed to. Finally obtain the probability of subspace convergence (PSC) by Eq. 11

$$PSC = \frac{s}{k} \tag{11}$$

Table 3
The Functions and Constants of Data Set.

Name	Functions	Constants(ERC)
Koza	$+, -, \times, /, sin, cos, e^n, ln(\|n\|)$	None
Korns	$+, -, \times, /, sin, cos, e^n, ln(\|n\|)$	Random finite 64bit
	$n^2, n^3, \sqrt{n}, tan, tanh$	IEEE double
Keijzer	$+, \times, \frac{1}{n}, -n, \sqrt{n}$	Random value from
		$N(u=0, \sigma=5)$
Vladislavleva-*A*	$+, -, \times, /, n^2$	$n^\epsilon \quad n+\epsilon \quad n\epsilon$
Vladislavleva-B	$+, -, \times, /, n^2, e^n, e^{-n}$	$n^\epsilon \quad n+\epsilon \quad n\epsilon$
Vladislavleva-C	$+, -, \times, /, n^2, e^n, e^{-n}, sin, cos$	$n^\epsilon \quad n+\epsilon \quad n\epsilon$

Table 4
Parameters of Algorithms.

	GEP	SPJ-GEP	GEP-ADF	SPJ-GEP-ADF	SL-GEP	SPJ-SL-GEP
Population size	100	100	100	100	100	100
Head length	20	20	5/10	5/10	5/10	5/10
Chromosome length	41	41	43	43	43	43
Max generations	10^5	10^5	10^5	10^5	10^5	10^5
Mutation rate	0.03	0.03	0.03	0.03		
Inversion rate	0.1	0.1	0.1	0.1		
1-point recombination	0.7	0.7	0.7	0.7		0.7
confidence interval α		0.1		0.1		0.1
prob of convergence β		0.8		0.8		0.8
num of subspaces k		144–1000		144–1000		144–1000

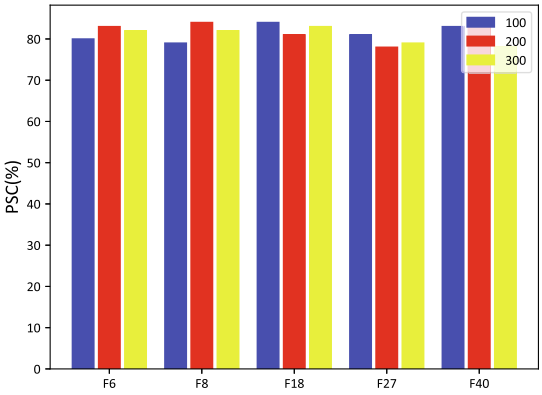


Fig. 4. The probability of subspace convergence with different k.

Table 5
Performance Metrics.

F	GEP		SPJ-GEP		GEP-ADF		SPJ-GEP-ADF		SL-GEP		SPJ-SL-GEP	
	Suc[1]	RMSE[2]	Suc	RMSE	Suc	RMSE	Suc	RMSE	Suc	RMSE	Suc	RMSE
F1	10%	**0.0207**	**20%**	0.0250	10%	0.0314	**60%**	**0.0153**	**30%**	**0.026**	10%	0.0346
F2	50%	0.0116	**70%**	**0.0068**	50%	0.0156	**60%**	**0.0091**	80%	0.0086	**100%**	**0**
F3	90%	0.0084	**100%**	**0.0037**	100%	0.0083	100%	**0.0075**	**100%**	**0.0037**	90%	0.0069
F4	80%	0.0071	**100%**	**0.0038**	80%	0.0100	**100%**	**0.0073**	100%	0.0074	100%	**0.0063**
F5	100%	0.0058	100%	**0**	30%	0.0149	**100%**	**0.0031**	**100%**	**0.0014**	30%	0.0209
F6	80%	0.0094	**90%**	**0.0076**	40%	0.0137	**60%**	**0.0114**	40%	0.0137	**50%**	**0.0088**
F7	20%	0.0146	**100%**	**0.0024**	50%	0.0141	**90%**	**0.0021**	60%	0.0096	**70%**	**0.0091**
F8	**10%**	**0.0296**	0%	0.0544	**20%**	0.0225	0%	**0.0087**	0%	0.5061	0%	**0.4075**
F9	**80%**	**0.2323**	20%	0.2557	80%	0.1582	80%	**0.1540**	0%	1.2302	0%	**1.0982**
F10	0%	0.9459	0%	**0.0451**	0%	0.9440	0%	**0.0398**	0%	0.1527	0%	**0.1497**
F11	0%	**0.0285**	0%	0.0430	10%	0.1668	**60%**	**0.0264**	0%	0.0381	0%	**0.0377**
F12	0%	**0.1073**	0%	0.1211	0%	0.0746	0%	**0.0226**	0%	0.0488	0%	**0.0431**
F13	0%	**0.0697**	0%	0.1376	0%	**0.0051**	0%	0.0268	0%	**0.1692**	0%	0.1713
F14	**40%**	**0.0082**	0%	0.0200	80%	0.0081	**90%**	**0.0079**	0%	**1.3877**	0%	1.5230
F15	0%	**0.0518**	0%	0.9583	**30%**	**0.0384**	20%	0.9440	0%	2.7841	0%	**2.6733**
F16	0%	0.1003	0%	**0.0652**	0%	**0.0659**	0%	0.0672	0%	0.1396	0%	**0.0558**
F17	0%	0.0428	0%	**0.0395**	0%	0.0406	0%	**0.0323**	0%	0.0697	0%	**0.0689**
F18	0%	0.4622	0%	**0.4243**	0%	0.4687	0%	**0.4086**	0%	0.7133	0%	**0.6393**
F19	0%	1.3266	0%	**1.2119**	0%	1.2740	0%	**1.0940**	0%	**1.3299**	0%	1.6041
F20	0%	0.3206	0%	**0.2801**	0%	0.2839	0%	**0.2627**	0%	0.7109	0%	**0.6504**
=[3]			11	0			10	0			14	0
+			6	12			8	17			3	14
−			3	8			2	3			3	6

[1] "Suc" represents the proportion of finding the correct result whose fitness computed by RMSE is less than 0.01.
[2] "RMSE" shows the average fitness obtained by running one of these Algorithms 10 times.
[3] "=,+,−" represents comparison results: same, good and poor. And the numeric in the row represents the comparison result between GEP and GPJ-GEP (GEP-ADF and SPJ-GEP-ADF, or SL-GEP and SPJ-SL-GEP).

where s is the number of subspaces that satisfy the above Inequality.

The PSCs on the five test problems with different values of k are illustrated in Fig. 4. All PSCs are close to 0.8, which is the value of parameter β. So, Inequality 3 is correct in determining when UCB_{ω_i} (Eq. 2) loses its effect on most subspaces.

5.3. Performance metrics for comparison

To obtain the performance metrics of the algorithms: GEP, SPJ-GEP, GEP-ADF, SPJ-GEP-ADF, SL-GEP, and SPJ-SL-GEP, each of the algorithms runs 10 times on the 20 test problems. Moreover, their results are shown in Table 5. For example, 50% in the column "suc" means that the number of finding correct results is 5 in 10 runs of one of these algorithms.

Observing the comparative data between GEP and SPJ-GEP from Table 5, we conclude that SPJ-GEP finds more correct results than GEP ("6" in row "+" and "3" in row "−") and obtains a better average fitnesses than GEP ("12" in row "+"). Comparing data between GEP-ADF and SPJ-GEP-ADF, SPJ-GEP-ADF can find more correct results and a better average fitnesses

Table 6
Wilcoxon's Sign Rank Test [44].

F	GEP/SPJ-GEP			GEP-ADF/SPJ-GEP-ADF			SL-GEP/SPJ-SL-GEP		
	T+	T−	Result[1]	T+	T−	Result	T+	T−	Result
F1	13	−42	≈	47	−8	≈	18	−37	≈
F2	40	−15	≈	40	−15	≈	47	−8	≈
F3	51	−4	+	38	−17	≈	10	−45	≈
F4	51	−4	+	44	−11	≈	45	−10	≈
F5	52	−3	+	52	−3	+	3	−52	−
F6	40	−15	≈	38	−17	≈	36	−19	≈
F7	51	−5	+	49	−6	+	34	−21	≈
F8	0	−55	−	54	−1	+	32	−23	≈
F9	2	−53	−	33	−22	≈	49	−6	+
F10	40	−15	≈	23	−32	≈	32	−23	≈
F11	3	−52	−	55	0	+	34	−21	≈
F12	19	−36	≈	46	−9	+	34	−21	≈
F13	1	−54	−	0	−55	−	24	−31	≈
F14	12	−43	≈	30	−25	≈	12	−43	≈
F15	1	−54	−	55	0	+	39	−16	≈
F16	49	−6	+	22	−33	≈	53	−2	+
F17	30	−25	≈	41	−14	≈	31	−24	≈
F18	49	−6	+	46	−9	+	21	−34	≈
F19	46	−9	+	37	−18	≈	14	−41	≈
F20	54	−1	+	41	−14	≈	7	−48	≈
≈			7			12			17
+			8			7			2
−			5			1			1

[1] "+, −, ≈" represents that the competitor is significantly better, worse than and similar to SPJ-GEP or SPJ-GEP-ADF, respectively, according to the Wilcoxon signed-rank test $\alpha = 0.05$. And the numeric in the row represents the comparison result between GEP and GP-GEP (GEP-ADF and GP-GEP-ADF, or SL-GEP/SPJ-SL-GEP).

than GEP-ADF. Although SL-GEP and SPJ-SL-GEP find the same number of correct results, SPJ-SL-GEP can still obtain a smaller average fitness value than SL-GEP.

In Table 6, the results of the Wilcoxon Signed-Rank Test[44] are listed. It considers both the average fitnesses and RMSE. According to the comparative data, SPJ-GEP, SPJ-GEP-ADF, and SPJ-SL-GEP still obtain a better performance than the original GEP, GEP-ADF, and SL-GEP, respectively.

All best fitnesses found by the above six algorithms are shown as points in Fig. 5. The results indicate that, in most of the problems, the range of fitness obtained by SPJ-GEP, SPJ-GEP-ADF, or SPJ-SL-GEP is smaller than the range of fitness obtained by GEP, GEP-ADF, or SL-GEP. These results show that SPJ-GEP can obtain more accurate performance than the baseline GEP algorithms.

However, SPJ-GEP is not superior to GEP, GEP-ADF or SL-GEP in all the tested problems. For a few special problems, the performance of GEP, GEP-ADF or SL-GEP is better than SPJ-GEP. For example, as shown in Fig. 6(a), the average and the range of GEP fitness value both are smaller than those of SPJ-GEP fitness value; similarly, in Fig. 6(h), the range of GEP-ADF (SL-GEP) fitness value is smaller than that of SPJ-GEP-ADF (SPJ-SL-GEP). That is because, from the subspace's view, although the mathematical expression space is split into many subspaces, the size of each subspace is still huge so that GEP, GEP-

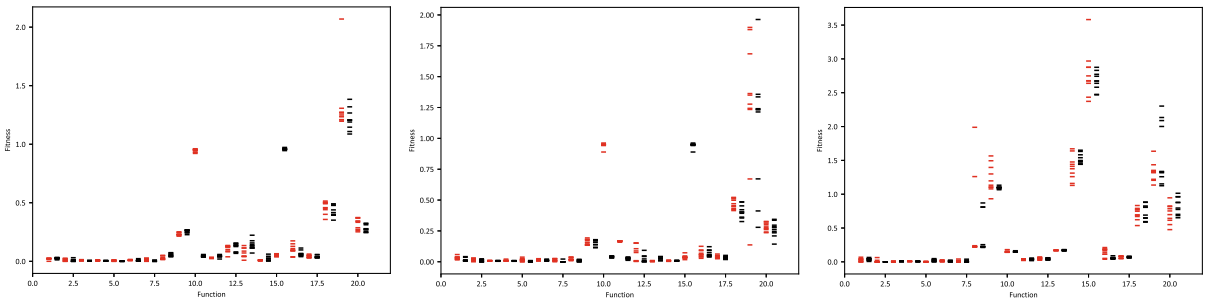


Fig. 5. Fitness comparison on all test problems. The x-coordinate shows the test problems. Red points represent the output of GEP, GEP-ADF, or SL-GEP, and black points represent the output of SPJ-GEP, SPJ-GEP-ADF, or SPJ-SL-GEP.

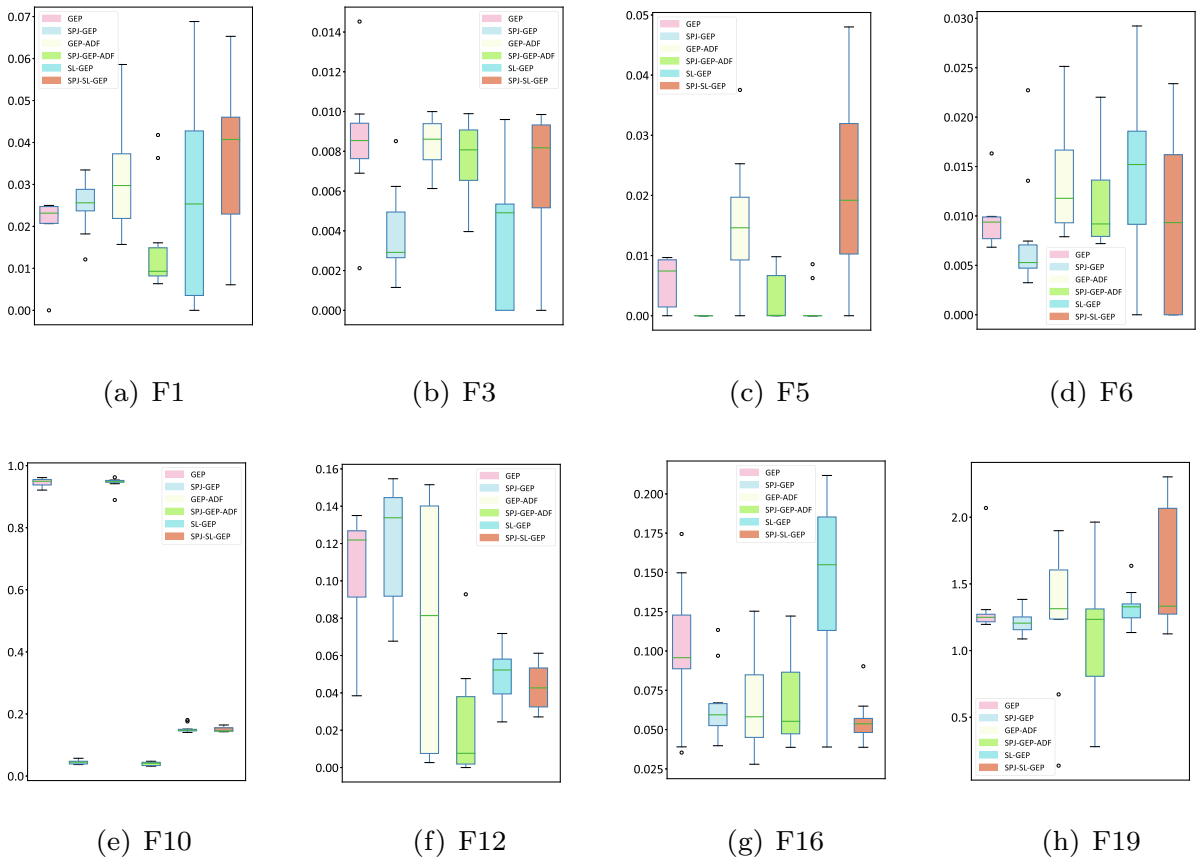


Fig. 6. Fitness comparison on special test problems. In each subgraph, boxes from left to right represent GEP, SPJ-GEP, GEP-ADF, SPJ-GEP-ADF, SL-GEP, and SPJ-SL-GEP, respectively.

ADF, and SL-GEP may only search in one or several subspaces. Moreover, the crossover and the mutation of SPJ-GEP (SPJ-GEP-ADF or SPJ-SL-GEP) are the same as those of GEP (GEP-ADF or SL-GEP) in a subspace. Those make it possible for the baseline GEPs to find more accurate results than these SPJ-GEP algorithms if the subspace that they search for is the target subspace in which the correct results are located.

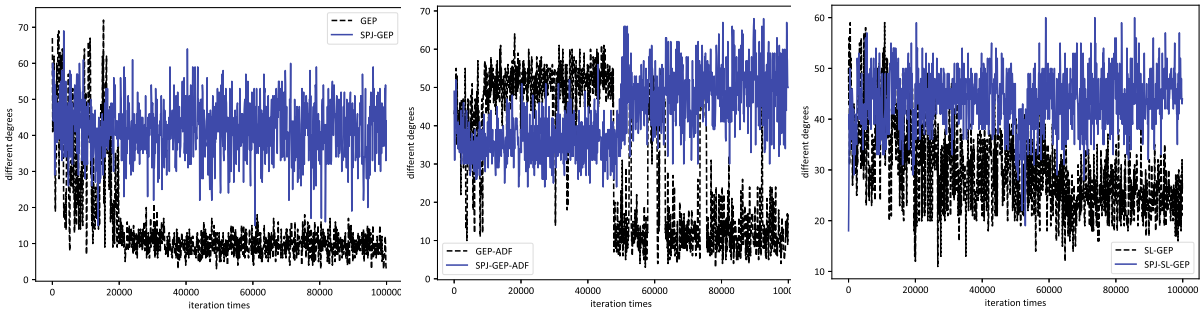
5.4. Comparison of population diversity

Since the crossover and the mutation of SPJ-GEP are the same as those of the baseline GEPs in a subspace, what makes SPJ-GEP obtain more accurate results than these GEPs in most of the problems? It is attributed to the SPJ-GEP's ability to jump between subspaces, which maintains population diversity. To observe the population diversity, the different degree (dg) of individuals is defined by the following equation.

$$dg = \frac{D}{N} \tag{12}$$

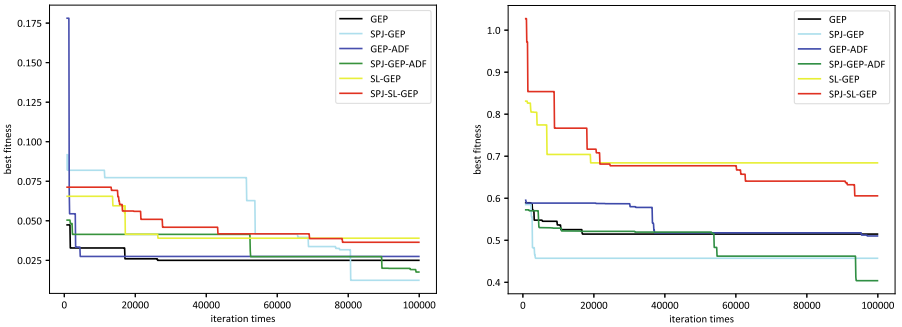
where N is the number of individuals in a population, and D is the number of different fitness values of individuals. For example, if 100 individuals generate 70 different fitness values, the different degree is 0.7. Therefore, dg can represent the population diversity, and dg is computed at every 100 generations in the above algorithms. By collecting dg values of the above six algorithms that run in the test problem - F11, the population diversity changes of these algorithms are shown in Fig. 7.

As shown in Fig. 7, GEP, GEP-ADF, and SL-GEP have a lower level of the population diversity, while SPJ-GEP. SPJ-GEP-ADF and SPJ-SL-GEP have a higher level of the population diversity. Besides, the average amplitude of blue curves that represent these SPJ-GEPs is higher than that of black curves that represent the three baseline GEPs. Although crossover and mutation contribute to maintaining population diversity in the early stages of these GEPs running, they tend to make little differences between individuals in the later stages. However, the jump always makes significant differences between individuals in SPJ-GEPs, according to Lemma 1.



(a) GEP vs SPJ-GEP (b) GEP-ADF vs SPJ-GEP-ADF (c) SL-GEP vs SPJ-SL-GEP

Fig. 7. The different degree comparison on F11.



(a) F1 (b) F18

Fig. 8. Convergence comparison.

5.5. Convergence comparison

Fig. 8(a) and (b) illustrate that SPJ-GEPs (SPJ-GEP, SPJ-GEP-ADF, and SPJ-SL-GEP) can obtain better results than the baseline GEPs(GEP, GEP-ADF, and SL-GEP). As the number of iterations increases, the baseline GEPs gradually fall into local optimum space. Meanwhile, owing to the low population diversity, they cannot escape from the local optimum space with a high probability. However, SPJ-GEPs can easily escape from a local optimum space because the jump between subspaces transfers individuals to different subspaces and maintains a high population diversity. Therefore, when the baseline GEPs present premature convergence, SPJ-GEPs still find new better results even in the later stage of 100,000 generations. That demonstrates that SPJ-GEPs can always explore new subspaces with a high probability.

However, owing to the jump between subspaces and the above subspace selection method, it is difficult for SPJ-GEPs to exploit a local space continuously in a period. Therefore, the convergence speed of SPJ-GEPs is slower than that of these baseline GEPs in the early stage of 100,000 generations. For example, in Fig. 8(a), at approximately 40,500 generations, GEP finds the best result with the fitness 0.025, while SPJ-GEP finds that with 0.078.

6. Related work

Similar to our idea of space partition that maintains the population diversity in this study, a method named NrGA was proposed to use a binary space partitioning (BSP) tree. NrGA recursively subdivides space into two and stores individual visiting information, and the method is integrated with GA so that individual revisits are completely eliminated [25–27]. Although NrGA can maintain the population diversity by visiting the BSP tree, it needs a lot of additional time and space to maintain the tree. Especially for the huge space of mathematical expressions in SR, maintaining the BSP tree will become an impossible mission. However, in the paper, SPJ-GEP does not construct a space-partition tree but utilizes its abstract structure to obtain a space-partition set. Moreover, SPJ-GEP maintains population diversity by letting individuals jump between subspaces instead of eliminating revisiting individuals.

Many distributed evolutionary algorithms (dEA) use a similar idea of space partition to distribute individuals of the population to different subspaces (multiple processors or computing nodes) [28]. These dEAs can be divided into island [29] [45],

cellular [46] [30], hierarchical [47], and pool models [48]. These dEAs can increase the population diversity because their subpopulations run independently in different subspaces, and transfer their best individuals by a migration strategy. As they are distributed algorithms, they pay more attention to the communication cost and scalability and pay no attention to the subspace selection method, so that they could waste a lot of computation time on invalid subspaces.

Different from the above methods of partitioning global space, Tsutsui [49] and Huang [50] proposed two methods of local spaces that are created by the convergence status of the present population respectively. Tsutsui [49] proposed the forking GA (fGA) which divides the search space for each population into subspaces depending on the convergence status of the population and the solutions obtained so far. Then two types of fGAs (genotypic fGA and phenotypic fGA) are created to maintain population diversity by defining two searching subspaces of each sub-populations, respectively. One is the salient schema which defines subspaces by phenotype parameters of the present population. The other is the neighborhood hypercube which defines the local subspaces around the current best individual in the phenotypic feature space. The forking GA can avoid the premature convergence of populations because the searching method enables the population to exploit different local subspaces. Although Huang [50] proposed a differential evolution (DE) method based on the three spaces: local space, opposition space, and global space, the three spaces are all local spaces because the global space refers to the space near the best individual in a population, and its opposition space. So, the method can accelerate convergence but cannot avoid falling into a local optimum.

7. Conclusion and future work

In the paper, we propose a novel algorithm, SPJ-GEP, to deal with the SR problem. Using the new approach that partitions the space of mathematical expressions into subspaces, SPJ-GEP guides the population effectively jump among these subspaces with a subspace selection method. SPJ-GEP maintains the population diversity while keeping the balance between subspace exploration and exploitation. Therefore, the proposed SPJ-GEP has the following advantages. SPJ-GEP can be easily embedded in other GEPs because its three key components – space partition, subspace selection, and crossover, are compatible with other GEPs. As shown in the evaluation analysis, SPJ-GEP does not significantly increase the time and space complexity compared with classical GEPs. SPJ-GEP can overcome the problem of premature convergence and avoid falling into a local optimum.

Although SPJ-GEP surpasses the tested baseline GEPs on most benchmarks, it has two weaknesses that prevent it from quickly finding better results than the baseline GEPs on a few benchmarks. One is that SPJ-GEP does not define how to choose the best space-partition set from the space-partition tree. After all, the quality of the set directly affects the search results of SPJ-GEP. The other weakness is that the selected subspace may not be the subspace where the optimal result is located. In this case, the jump to these selected subspaces could result in an unreasonable search. In the future, we will address these weaknesses by quantifying these subspaces.

CRediT authorship contribution statement

Qiang Lu: Conceptualization, Methodology, Validation, Writing - original draft, Writing - review & editing. **Shuo Zhou:** Formal analysis, Data curation, Software, Visualization. **Fan Tao:** Visualization, Validation. **Jake Luo:** Writing - review & editing. **Zhiguang Wang:** Supervision.

Declaration of Competing 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.

Acknowledgements

This work was supported by China National Key Research Project (No. 2019YFC0312003), National Natural Science Foundation of China (No. 61402532) and the Science Foundation of China University of Petroleum-Beijing (No. 01JB0415).

References

[1] John R. Koza, Genetic programming as a means for programming computers by natural selection, Stat. Comput. 4 (2) (1994) 87–112, https://doi.org/10.1007/BF00175355.
[2] R.I. McKay, N.X. Hoai, P.A. Whigham, Y. Shan, M. O'Neill, Grammar-based genetic programming: a survey, Genet. Program Evolvable Mach. 11 (3) (2010) 365–396.
[3] Q. Lu, J. Ren, Z. Wang, Using genetic programming with prior formula knowledge to solve symbolic regression problem, Comput. Intell. Neurosci. 1 (2016) 1–19.
[4] A. Moraglio, K. Krawiec, C.G. Johnson, Geometric Semantic Genetic Programming, in: Parallel Problem Solving from Nature - PPSN XII, Vol. 7491, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 21–31..
[5] Q. Chen, B. Xue, M. Zhang, Improving generalization of genetic programming for symbolic regression with angle-driven geometric semantic operators, IEEE Trans. Evol. Comput. 23 (3) (2019) 488–502.

[6] M. Schmidt, H. Lipson, Comparison of tree and graph encodings as function of problem complexity, in: Proceedings of the 9th annual conference on Genetic and evolutionary computation, GECCO '07, Association for Computing Machinery, London, England, 2007, pp. 1674–1679.
[7] M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data, Science 324 (5923) (2009) 81–85.
[8] J.F. Miller, S.L. Harding, Cartesian Genetic Programming, in: Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '08, ACM, New York, NY, USA, 2008, pp. 2701–2726.
[9] J.F. Miller, Cartesian genetic programming: its status and future, Genet. Program Evolvable Mach. (2019) 1–40.
[10] C. Ferreira, Gene expression programming: a new adaptive algorithm for solving problems, Complex Syst. 13 (2) (2001) 87–129.
[11] C. Ferreira, Automatically defined functions in gene expression programming, Genetic Systems Programming, Springer, 2006, pp. 21–56.
[12] J. Zhong, Y.S. Ong, W. Cai, Self-learning gene expression programming, IEEE Trans. Evol. Comput. 20 (1) (2016) 65–80, https://doi.org/10.1109/TEVC.2015.2424410.
[13] M.F. Brameier, W. Banzhaf, Linear genetic programming, Springer Science & Business Media, 2007.
[14] Yee Leung, Yong Gao, Xu. Zong-Ben, Degree of population diversity: a perspective on premature convergence in genetic algorithms and its markov chain analysis, IEEE Trans. Neural Networks 8 (5) (1997) 1165–1176, https://doi.org/10.1109/72.623217.
[15] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and Exploitation in Evolutionary Algorithms: A Survey, ACM Comput. Surv. 45 (3) (2013) 35:1–35:33. doi:10.1145/2480741.2480752..
[16] E. Burke, S. Gustafson, G. Kendall, Diversity in genetic programming: an analysis of measures and correlation with fitness, IEEE Trans. Evol. Comput. 8 (1) (2004) 47–62, https://doi.org/10.1109/TEVC.2003.819263.
[17] D. Sudholt, The Benefits of Population Diversity in Evolutionary Algorithms: A Survey of Rigorous Runtime Analyses. http://arxiv.org/abs/1801.10087..
[18] G. Karafotias, M. Hoogendoorn, A.E. Eiben, Parameter control in evolutionary algorithms: trends and challenges, IEEE Trans. Evol. Comput. 19 (2) (2015) 167–187, https://doi.org/10.1109/TEVC.2014.2308294.
[19] R.E. Smith, E. Smuda, Adaptively resizing populations: algorithm, analysis, and first results, Complex Systems 9 (1) (1995) 47–72.
[20] G.R. Harik, F.G. Lobo, A Parameter-less Genetic Algorithm, in: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1, GECCO'99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 258–265..
[21] L.J. Eshelman, D.J. Schaffer, in: R.K. Belew, L.B. Booker (Eds.), Preventing premature convergence in genetic algorithms by preventing incest Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, CA, 1991, pp. 115–122.
[22] Y. Wang, Z. Cai, Q. Zhang, Enhancing the search ability of differential evolution through orthogonal crossover, Inf. Sci. 185 (1) (2012) 153–177.
[23] Sibylle D. Mfiller, Nicol N. Schraudolph, Petros D. Koumoutsakos, Step size adaptation in evolution strategies using reinforcement learning, in: Congress on Evolutionary Computation, Vol. 1, IEEE, 2002, pp. 151–156. doi:10.1109/CEC.2002.1006225..
[24] K.M.S. Badran, P.I. Rockett, The Roles of Diversity Preservation and Mutation in Preventing Population Collapse in Multiobjective Genetic Programming, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, ACM, New York, NY, USA, 2007, pp. 1551–1558, https://doi.org/10.1145/1276958.1277272.
[25] S.Y. Yuen, C.K. Chow, A non-revisiting genetic algorithm, in: IEEE Congress on Evolutionary Computation, 2007, pp. 4583–4590. doi:10.1109/CEC.2007.4425072..
[26] S.Y. Yuen, C.K. Chow, A genetic algorithm that adaptively mutates and never revisits, IEEE Trans. Evol. Comput. 13 (2) (2009) 454–472, https://doi.org/10.1109/TEVC.2008.2003008.
[27] C.K. Chow, S.Y. Yuen, An evolutionary algorithm that makes decision based on the entire previous search history, IEEE Trans. Evol. Comput. 15 (6) (2011) 741–769, https://doi.org/10.1109/TEVC.2010.2040180.
[28] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, J.-J. Li, Distributed evolutionary algorithms and their models: a survey of the state-of-the-art, Appl. Soft Comput. 34 (2015) 286–300, https://doi.org/10.1016/j.asoc.2015.04.061.
[29] D. Whitley, S. Rana, R.B. Heckendorn, Island model genetic algorithms and linearly separable problems, in: D. Corne, J.L. Shapiro (Eds.), Evolutionary Computing, Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 1997, pp. 109–125.
[30] E. Alba, B. Dorronsoro, Cellular Genetic Algorithms, Operations Research/Computer Science Interfaces Series, Springer, US, 2008.
[31] Peter Auer, Nicolò Cesa-Bianchi, Paul Fischer, Finite-time analysis of the multiarmed bandit problem, Mach. Learn. 47 (2) (2002) 235–256, https://doi.org/10.1023/A:1013689704352.
[32] R. Sutton, A. Barto, Reinforcement Learning: An Introduction 2nd, MIT Press, 2018.
[33] J. Zhong, Y.S. Ong, W. Cai, Self-learning gene expression programming, IEEE Trans. Evol. Comput. 20 (1) (2016) 65–80, https://doi.org/10.1109/TEVC.2015.2424410.
[34] Michael N. Katehakis, Arthur F. Veinott, The multi-armed bandit problem: decomposition and computation, Math. Operations Res. 12 (2) (1987) 262–268.
[35] Zhi-xiong Xu, Xi-liang Chen, Lei Cao, Chen-xi Li, A study of count-based exploration and bonus for reinforcement learning, International Conference on Cloud Computing and Big Data Analysis (2017) 425–429doi:10.1109/ICCCBDA.2017.7951951..
[36] N. Cesa-Bianchi, C. Gentile, G. Lugosi, G. Neu, Boltzmann exploration done right, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates Inc, 2017, pp. 6284–6293.
[37] X. Li, W. Zhou, Chiand Xiao, P.C. Nelson, Prefix gene expression programming, Genetic and Evolutionary Computation Conf. (2005) 55–31..
[38] Jonathan Mwaura, Ed Keedwell, Adaptive gene expression programming using a simple feedback heuristic (2009) 6..
[39] M.W. David. Powers, Applications and explanations of zipf's law, in: Proceedings of the joint conferences on new methods in language processing and computational natural language learning, 1998, pp. 151–160..
[40] James McDermott, David R. White, et al, Genetic programming needs better benchmarks, ACM Press (2012) 791, https://doi.org/10.1145/2330163.2330273.
[41] Michael F. Korns, Accuracy in symbolic regression, in: Genetic Programming Theory and Practice IX, Springer, New York, 2011, pp. 129–151, https://doi.org/10.1007/978-1-4614-1770-5-8.
[42] Maarten Keijzer, Improving symbolic regression with interval arithmetic and linear scaling, in: Genetic Programming, Vol. 2610, Springer, Berlin Heidelberg, 2003, pp. 70–82. doi:10.1007/3-540-36599-0-7..
[43] E. Vladislavleva, G. Smits, D. den Hertog, Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming, IEEE Trans. Evol. Comput. 13 (2) (2009) 333–349, https://doi.org/10.1109/TEVC.2008.926486.
[44] Frank Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (2) (1994) 80–83, https://doi.org/10.2307/3001968, http://www.jstor.org/stable/3001968.
[45] F. Herrera, M. Lozano, Gradual distributed real-coded genetic algorithms, IEEE Trans. Evol. Comput. 4 (1) (2000) 43–63, https://doi.org/10.1109/4235.843494.
[46] D. Bernabé, T. Marco, G. Mario, A. Enrique, Decentralized cellular evolutionary algorithms, in: Handbook of Bioinspired Algorithms and Applications, Chapman and Hall/CRC, 2005, pp. 121–138..
[47] G. Folino, C. Pizzuti, G. Spezzano, Training distributed GP ensemble with a selective algorithm based on clustering and pruning for pattern classification, IEEE Trans. Evol. Comput. 12 (4) (2008) 458–468, https://doi.org/10.1109/TEVC.2007.906658.
[48] G. Roy, H. Lee, J.L. Welch, Y. Zhao, V. Pandey, D. Thurston, A distributed pool architecture for genetic algorithms, in: 2009 IEEE Congress on Evolutionary Computation, 2009, pp. 1177–1184, https://doi.org/10.1109/CEC.2009.4983079.
[49] S. Tsutsui, Y. Fujimoto, A. Ghosh, Forking genetic algorithms: Gas with search space division schemes, Evol. Comput. 5 (1) (1997) 61–80.
[50] W. Huang, S.-K. Oh, Z. Guo, W. Pedrycz, A space search optimization algorithm with accelerated convergence strategies, Appl. Soft Comput. 13 (12) (2013) 4659–4675, https://doi.org/10.1016/j.asoc.2013.06.005.