Research Article

A Search History-Driven Offspring Generation Method for the Real-Coded Genetic Algorithm

Takumi Nakane,1 Xuequan Lu,2 and Chao Zhang1

1University of Fukui, Fukui, Japan
2Deakin University, Melbourne, Victoria, Australia

Correspondence should be addressed to Chao Zhang; zhang@u-fukui.ac.jp

Received 13 May 2020; Revised 1 September 2020; Accepted 9 September 2020; Published 27 September 2020

Academic Editor: José Alfredo Hernández-Pérez

Copyright © 2020 Takumi Nakane et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In evolutionary algorithms, genetic operators iteratively generate new offspring which constitute a potentially valuable set of search history. To boost the performance of offspring generation in the real-coded genetic algorithm (RCGA), in this paper, we propose to exploit the search history cached so far in an online style during the iteration. Specifically, survivor individuals over the past few generations are collected and stored in the archive to form the search history. We introduce a simple yet effective crossover model driven by the search history (abbreviated as SHX). In particular, the search history is clustered, and each cluster is assigned a score for SHX. In essence, the proposed SHX is a data-driven method which exploits the search history to perform offspring selection after the offspring generation. Since no additional fitness evaluations are needed, SHX is favorable for the tasks with limited budget or expensive fitness evaluations. We experimentally verify the effectiveness of SHX over 15 benchmark functions. Quantitative results show that our SHX can significantly enhance the performance of RCGA, in terms of both accuracy and convergence speed. Also, the induced additional runtime is negligible compared to the total processing time.

1. Introduction

Evolutionary algorithms (EAs) have been shown to be generic and effective to search for global optima in the complex search space theoretically [1–3] and practically [4–6]. The exploration process of EAs imitates the natural selection process, which is realized by conducting the offspring generation and survivor individual selection alternately and iteratively. The population quality is gradually improved throughout the exploration process, which can be viewed as a stochastic population-based generation-and-test process. Because of the offspring generation, a large number of candidate solutions (i.e., individuals) are sampled, accompanied by corresponding fitness values, genetic information, and genealogy information. Such accumulated search data constitute search history which can be very informative and valuable for boosting the overall performance. For instance, exploiting search history can be useful for improving the search procedure under a limited budget of fitness evaluations (FEs). That is, no additional FEs are allowed for improving the search performance. Also, the computational cost of a single FE can be high when the fitness functions are complicated. To enable a better solution for the population without increasing the number of FEs, the way of exploiting the search history truly matters. Nevertheless, search history has been sparsely exploited and studied in existing methods.

Real-coded genetic algorithm (RCGA) has been widely studied in the past decades [7–11], and the main efforts for improving the performance of RCGA have been focused on the development of the crossover techniques [12]. Because the crossover operator is to generate new offspring from the current population, the quality of the new solutions straightforwardly affects the evolution direction and convergence speed. Given different mechanisms, crossover methods can differ from (1) parent selection, (2) offspring generation, and (3) offspring selection. Both parent and offspring can be more than two, depending on the design. The abovementioned three aspects associate the exploration...
ability with exploitation ability, and the degree and balance between both abilities affect the performance largely [13]. Although the self-adaptive feature of RCGA [14] can adjust the relationship to a certain extent, the "best" degrees and balance between exploration and exploitation for achieving a satisfactory solution can differ greatly with respect to different problem settings and can be hardly achieved with the adaptive feature.

With a large amount of search history data up to the current generation in hand, we attempt to introduce a crossover method that effectively exploits the history data in this paper. At first, an archive is defined to collect the survivor individuals over generations as the search history. Then, the stored individuals are clustered by $k$-means [15], and each cluster is assigned a score depending on the number of belonging individuals. At last, offspring is generated and selected according to the scores. We introduce two different schemes to update the archive. The proposed crossover operator, named search history-driven crossover (SHX), generates offspring by considering the cluster scores. Since SHX enables an offspring selection mechanism, any existing parent selection and offspring generation mechanisms can be easily integrated with it. To our knowledge, this is the first work to design the crossover model by effectively exploiting search history. We present a set of experiments to systemically evaluate the effectiveness of the proposed method using 15 benchmark functions. Three conventional crossover operators are employed, and the results with/without SHX are compared. Apart from the above, two archive update methods are also analyzed.

The main technical contributions of this paper are threefold. First, we propose a novel crossover model by effectively exploiting the search history. Second, we introduce the offspring selection based on the clusters calculated from the search history. Third, we introduce two schemes to update the survivor archive. A preliminary version of this paper appears in GECCO2020 [16].

2. Related Work

Crossover is one of the principal operators for generating offspring and deeply relates to the performance of the real-coded genetic algorithm (RCGA). Blend-$\alpha$ crossover (BLX-$\alpha$) [17] proposed by Eshelman and Schaffer is one of the most popular operators. Offspring genes are independently and uniformly sampled within an interval between a gene pair of parents. The parameter $\alpha$ corresponds to the extension of the sampling interval, which plays a key role in maintaining the diversity of offspring. Eshelman et al. proposed Blend-$\alpha$-$\beta$ crossover (BLX-$\alpha$-$\beta$) [18] which involves two extension parameters. Deb and Agrawal introduced simulated binary crossover (SBX) [19] which simulates the single-point crossover in binary-coded GA for continuous search space. The interval used in SBX is determined by a polynomial probability distribution $\beta$ depending on the distribution index $\eta$. $\eta$ indirectly adjusts the tendency of offspring generation. The above crossover operators have a common feature that the offspring genes are extracted according to a certain probability distribution from the predefined interval on the parent genes. This feature enables better results than using crossover operators for binary coding in the continuous search space. On the other hand, some crossover operators set more than two individuals as parents, which aim to generate offspring with well-preserved population statistics. In the case of unimodal normal distribution crossover (UNDX) [20], the generation of offspring follows a unimodal normal distribution defined on the line connecting two of the three parents. For simplex crossover (SPX) [21], $D + 1$ individuals are taken as parents in the $D$-dimensional search space. SPX uniformly generates offspring within $D$-dimensional simplex constructed by parent individuals and expanded by a parameter $\epsilon$.

Search history has also been exploited in some research, but to the best of our knowledge, none of them is for the purpose of improving the crossover model. Since online real systems often provide uncertain evaluation values which lead to unreliable convergence of GA, Sano and Kita proposed memory-based fitness estimation GA (MFEGA) [22]. MFEGA estimates the fitness from neighboring individuals stored in the search history. Leveraging search history allows estimation without requiring additional evaluation. Amor and Rettinger proposed GA using self-organizing maps (GASOM) [23]. SOM (self-organizing maps) can provide a visualized search history, which makes the regions explored intuitive for users. Moreover, individual novelty is introduced by the activation frequency in the search history table and utilized by the reseeding operator to preserve the exploration power. Yuen and Chow presented the continuous nonrevisiting GA (cNrGA) [24]. A binary partitioning tree called a density tree stores all evaluated individuals and divides the search space into nonoverlapped partitions by means of distributions. These subregions are used to check whether a new individual needs to be evaluated or not.

3. Overview

Principles of designing good crossover operators for RCGA are discussed in [25]. Two among them are especially important: (1) the crossover operator should preserve the statistics of the population; (2) the crossover operator should generate offspring with as much diversity as possible under the constraint of (1). By following these suggestions, the key idea of SHX is to cluster the search history and select population members from excessively generated candidate solutions by preserving the statistics represented by the clusters. Figure 1 illustrates the overview of our SHX. The proposed method is performed under the framework of RCGA which mainly involves survivor selection and crossover. Mutation is optional, but we exclude it to clearly investigate the effectiveness of SHX in this work.

The proposed method is described in Algorithm 1. Population is denoted by $P$ which comprises $n_p$ individuals, and the population at the $t$-th generation is denoted as $P^t$. Similarly, parents for SHX, excessively generated candidate solutions during SHX, offspring after SHX, and survivors for the next generation are represented by $P_{par}$, $P_{can}$, $P_{off}$, and $P_{sur}$, respectively. The size of each set is denoted using $n$ with a subscript of the set name (e.g., the size of parents is denoted as $n_p$).
Population $P$ is conducted after the survivor selection. Survivor individuals in the search space. Nfl_he archive update process is generated from generation. Different from conventional RCGA, individuals generated from $P_{par}$ are regarded as offspring candidates $P_{can}$. The main purpose of SHX is to narrow down $P_{can}$ to $n_{P_{par}}$ individuals denoted by $P_{off}$ according to the statistics provided by $S$. $S$ is calculated from the clustering result of the archive and immediately impacts the offspring selection.

SHX can adopt any existing crossover operators (e.g., BLX-$\alpha$ [17] and SPX [21]) for the offspringGeneration function (Algorithm 1, line 8) to generate $P_{can}$ from $P_{par}$. For the parentSelection function (Algorithm 1, line 6) and the survivorSelection function (Algorithm 1, line 11), the just generation gap (JGG) [26, 27] is employed in this work. That is, the parentSelection function randomly extracts $n_{P_{par}}$ individuals from $P$ as $P_{par}$, and the survivorSelection function selects top-$n_{P_{par}}$ individuals in $P_{off}$ as $P_{sur}$ according to the fitness value. To show the performance increase brought by SHX, we choose the widely applied BLX-$\alpha$, SPX, and UNDX for the offspring generation and compare the results in Section 6. We explain archiveUpdate (Algorithm 1, lines 3 and 13) and offspringSelection (Algorithm 1, line 9) in detail in Section 4 and Section 5, respectively.

4. Survivor Archive

Since the genetic operations are run alternately and iteratively, collecting and analyzing the history data may be beneficial for boosting performance. Given that SHX is to maintain the historical statistics $S$ while producing offspring for the next generation, the archive $A$ is designed to store $P_{sur}$ over few past generations and extracts statistics $S$. The calculation of $S$ is based on the $k$-means, which is an off-the-shelf nonsupervised clustering method. The pseudocode of $k$-means is shown in Algorithm 2. In particular, $k$-means is employed to cluster the individuals in $A$ based on their position in the search space, and $S$ is a normalized frequency histogram to show the proportion of each cluster size to $n_{A}$. A higher score indicates that the corresponding cluster is more likely to be a promising search region. The statistics can then be maintained by probabilistically assigning newly generated candidates to each cluster according to $S$.

To keep the computational cost brought by $k$-means within an acceptable and constant range, the archive size is fixed to $n_{A}$. That is, a part of individuals in $A$ must be replaced with new survivors $P_{sur}$ during the archive update to incorporate new information. Two types of update methods are considered in this work: (1) randomly selecting individuals in $A$ and replacing them with $P_{sur}$ (denoted by random); (2) replacing a part of $A$ with $P_{sur}$ in the order in which the individuals of $A$ arrived (denoted by sequential). The performance comparison between these two approaches is discussed in Section 6.

The update of $A$ and calculation of $S$ are executed in the function archiveUpdate (Algorithm 1, lines 3 and 13) which is summarized in Algorithm 3. At the replacement step (Algorithm 3, line 4), $n_{P_{sur}}$ individuals are discarded from $A$ based on random or sequential approaches, and new $P_{sur}$ are stored to $A$. Initialization is executed when $t$ equals 0. The $k$-meansFit function (Algorithm 3, line 7) updates the centroids of the clusters according to the updated $A$ and assigns updated cluster labels to each individual in $A$. After that, the normalized frequency histogram $S$ for each cluster is calculated by the hist function (Algorithm 3, line 9) for further usage in offspring selection (Algorithm 4). Note that the initial centroids of the clusters in the current generation are inherited from the previous generation, as most individuals in $A'$ are the same as $A^{t-1}$.
5. Search History-Driven Crossover (SHX)

SHX randomly selects parents by following the strategy of existing crossover operators (e.g., two parents in the case of BLX and $D + 1$ parents in the case of SPX) and excessively generates candidate offspring $P_{can}$ for further offspring selection. $n_{P_{can}} \gg n_{P_{off}}$ because $P_{can}$ must ensure a sufficient number of individuals that can be assigned to each cluster in $A$. Here, generating individuals excessively can also be considered as a mechanism of diversity preservation. It is worth pointing out that the offspring selection is a different procedure from the survivor selection. Offspring selection belongs to the crossover model and is conducted before fitness evaluation. Survivor selection is conducted after fitness evaluation. Offspring selection narrows down $P_{can}$ to $P_{off}$ based on roulette wheel selection [28]. Each proportion of the wheel relates to each possible selection (i.e., clusters), and $S$ is used to associate a probability of selection with each cluster in $A$. This can also be viewed as a procedure that SHX preferentially selects individuals in more “promising” regions. This bias selection can encourage the evolution of the population and accelerate the whole convergence. Besides,
of roulette selection produces a cluster ID, and one candidate is selected and assigned to the new generation is sampled based on the statistics of the population (e.g., cluster size) can be maintained between two consecutive generations because the new generation is sampled based on the statistics of the history. Also, the diversity of $P_{\text{off}}$ can be preserved because each newly generated individual from $P_{\text{can}}$ has a probability to be assigned to $A$.

The algorithm of offspring selection is shown in Algorithm 4. Input $P_{\text{can}}$ is excessively generated by existing crossover operators (Algorithm 1, line 8). Each candidate is labeled by the $k$-means Predict function (Algorithm 4, line 1) based on the current clusters estimated from $A$. Then, the roulette is constructed based on $S$. The roulette selection is called $n_{\text{p,off}}$ times, yielding $n_{\text{p,off}}$ selected offspring. Each time of roulette selection produces a cluster ID, and one candidate in $P_{\text{can}}$ that belongs to the corresponding cluster is randomly selected and assigned to $P_{\text{off}}$. To avoid duplicate selection, a selected candidate will be excluded from $P_{\text{can}}$. If no more candidates correspond to a certain cluster (this is rarely the case by assuming $n_{\text{p,can}} \gg n_{\text{p,off}}$), the roulette is reconstructed by eliminating the proportion of the corresponding cluster. Finally, $P_{\text{off}}$ is passed to the survivor selection process which determines $P_{\text{sur}}$ using JGG.

6. Experimental Results

The performance of SHX is investigated over 15 benchmark functions with different characteristics from the literature [31–33] for evaluation. Detailed information of each function is summarized in Table 1. Initialization of the population and the archive is conducted within the range provided by the 4th column in Table 1. It is worth mentioning that the searching space (i.e., range of parameters) during the generation alternation is not constrained. Each function is labeled according to different combination of characteristics (U + S, U + NS, M + S, and M + NS). By involving various characteristics of functions, we can analyze the proposed method more comprehensively and objectively. Furthermore, as all selected functions are adjustable in the setting of dimension, we adopt two different numbers of dimensions ($D = 5$ and $D = 10$) to control the difficulty degree of the search problem.

The setting of hyperparameters of the proposed method is listed in Table 2. The proposed method includes hyperparameters of not only RCGA (number of generations, $n_p$, and $n_{\text{p,off}}$) but also SHX ($n_{\text{p,can}}, n_A$, and $k$). Basically, the search problem defined by each function becomes more hard as the number of dimensions increases, which requires a lot of evaluations. For adaptive adjustment, the number of generations, $n_p$, and $n_{\text{p,off}}$ are set proportional to the number of dimensions. The constant values of each parameter are empirically determined because the purpose of the experiments is to validate the effectiveness of having SHX, rather than achieving the best solution for each function.

All experiments are executed 100 times with different random seeds. In each experiment, the generation alternation completely executed the number of generation times defined in Table 2. For a fair comparison, iterations under the same random seed start using the same population. The runtime and fitness are recorded with Python implementation (without either parallelization or optimization) on a i7-7700 CPU at 3.60 GHz, 12.0 GB RAM desktop computer.

**Algorithm 3: Archive update.**

| Function: $A^t, S^t = \text{archive Update} \left( A^{t-1}, P_{\text{sur}}, n_A \right)$. |
| --- |
| Input: archive $A^{t-1}$. |
| Survivors $P_{\text{sur}}$. |
| Size of the archive $n_A$. |
| Output: updated archive $A^t$, Score $S^t$. |
| (1) If $t = 0$ then. |
| // initialization. |
| (2) $n_A$ individuals in $A^t$ are randomly initialized. |
| (3) Else. |
| // archive update. |
| (4) randomly or sequentially (first in first out) select $n_{\text{p,off}}$ individuals. from $A^{t-1}$ to form $P_{\text{div}}$. |
| (5) $A^t = (A^{t-1}/P_{\text{div}}) \cup P_{\text{sur}}$. |
| (6) end. |
| // score update. |
| (7) labels = $k$-meansFit ($A^t$). |
| (8) $S^t = \text{hist(labels)}$; Calculating frequency histogram. |
| (9) Return $A^t, S^t$. |
6.2. Comparison in the Final-Generation-Elite. The results of the absolute error between the optimal value and the final-generation-elite fitness with respect to all combinations of functions, dimension, and methods are displayed in Table 3. Table 3 shows the minimum, maximum, median, mean, standard deviation (SD), and p value of the Mann–Whitney U test by each combination. The Mann–Whitney U test evaluates the significance of SHX results against results without SHX under the significance level \( p \leq 0.05 \). Before showing the superiority by involving SHX, we first exclude a few results that all the methods are trapped by local optima or cannot reach the global optima. (1) Easom Function \( f_8 \). This function has several local minima. It is unimodal, and the global minimum only has a small area corresponding to the search space, which can be hardly arrived at. (2) Schwefel 2.26 \( f_{10} \). Since the setup of this experiment does not restrict the range of parameters during search, an extremely small fitness value (even smaller than the global optimum) can be achieved with this function, which is not suitable for comparisons.

From Table 3, we can observe the clear improvement of performance brought by SHX. The results of the p value show that the methods with SHX have recognized the significance at least in 23 settings among all 30 settings. In the other five results (minimum, maximum, median, mean, and SD), the methods without SHX cannot achieve outperformed results for most settings. For instance, focusing on the minimum results, the methods without SHX outperform the methods with SHX only 5, 0, and 4 times by BLX, SPX, and UNDX, respectively. On the other hand, SHX with sequential archive update achieves the best performance. SH-BLX_sequential, SH-SPX_sequential, and SH-UNDX_sequential show the significance in 27, 26, and 27 settings, respectively. In addition, they achieve the best results in most settings with respect to the maximum, median, and mean results. One possible reason for sequential outperforming random in most cases is that sequential removes the oldest individual which arrived first, and therefore SHX can select offspring according to the up-to-date search history to reflect the trend of evolution more sensitively. In contrast, random uniformly removes individuals in the archive, which may impede the discovery of new solutions since old individuals may be retained for more generations in the archive.

6.2.1. Analysis on BLX vs. SH-BLX. It has been already known that the standard BLX [17] faces difficulties especially when the target function is nonseparable [34] due to the parameter-wise sampling. By observing the results of \( f_4 \) to \( f_7 \) and \( f_{12} \) to \( f_{15} \) from Table 3, we can find that involving SHX significantly improves the performance, which indicates that SHX can help BLX to greatly mitigate this drawback. It is easy to understand because offspring selection with clusters embeds distance measure which builds the relationship among parameters.
The last column (Label) represents the characteristics that the functions hold: unimodal (U), multimodal (M), separable (S), and nonseparable (NS).

### Table 1: Benchmark functions $f_1 \sim f_{15}$ used in the experiments.

| ID | Name               | Definition                                                                 | Range       | Label |
|----|--------------------|----------------------------------------------------------------------------|-------------|-------|
| f_1 | Schwefel 2.21      | $f(x) = \max_{1 \leq i \leq D} |x_i|$ | [−100, 100] | U, S  |
| f_2 | Sphere             | $f(x) = \sum_{i=1}^{D} x_i^2$                                             | [−10, 10]   | U, S  |
| f_3 | Sum squares        | $f(x) = \sum_{i=1}^{D} |x_i|^2$                                         | [−10, 10]   | U, S  |
| f_4 | Rosenbrock         | $f(x) = \sum_{i=1}^{D-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$            | [−30, 30]   | U, NS |
| f_5 | Schwefel 1.2       | $f(x) = \sum_{i=1}^{D} (x_i - \pi)^2$                                      | [−100, 100] | U, NS |
| f_6 | Schwefel 2.22      | $f(x) = \sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i|$                          | [−100, 100] | U, NS |
| f_7 | Zakharov           | $f(x) = \sum_{i=1}^{D} x_i^2 + ((1/2) \sum_{i=1}^{D} x_i^2)^2 + ((1/2) \sum_{i=1}^{D} x_i^2)^4$ | [−5, 10]   | U, NS |
| f_8 | Easom              | $f(x) = (-1)^D (\prod_{i=1}^{D} \cos^2(x_i)) \exp[-\sum_{i=1}^{D} (x_i - \pi)^2]$ | [−2π, 2π] | M, S  |
| f_9 | Rastrigin          | $f(x) = 10D + \sum_{i=1}^{D} (x_i^2 - 10 \cos(2\pi x_i))$                  | [−5.12, 5.12] | M, S  |
| f_10| Schwefel 2.26      | $f(x) = 418.9829D - \sum_{i=1}^{D} x_i \sin(\sqrt{|x_i|})$                | [−500, 500] | M, S  |
| f_11| Weierstrass1       | $f(x) = \frac{1}{\sqrt{D}} [\sum_{i=1}^{D} (x_i - \frac{1}{2})^2 + \sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i|]$ | [−0.5, 0.5] | M, S  |
| f_12| Ackley 1           | $f(x) = -20 \exp(-0.2 \sqrt{(1/D) \sum_{i=1}^{D} x_i^2}) - \exp((1/D) \sum_{i=1}^{D} \cos(2\pi x_i)) + 20 + \exp(1)$ | [−35, 35] | M, NS |
| f_13| Griewank           | $f(x) = (1/4000) \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos(x_i/\sqrt{D}) + 1$ | [−100, 100] | M, NS |
| f_14| Salomon            | $f(x) = 1 - \cos(2\pi \sqrt{\sum_{i=1}^{D} x_i^2}) + 0.1 \sqrt{\sum_{i=1}^{D} x_i^2}$ | [−100, 100] | M, NS |
| f_15| Xin-She Yang 2     | $f(x) = \frac{1}{4\pi} \min_{1 \leq i \leq D} \{ |x_i| \}$                | [−2π, 2π]   | M, NS |

The last column (Label) represents the characteristics that the functions hold: unimodal (U), multimodal (M), separable (S), and nonseparable (NS).

### Table 2: Hyperparameters of RCGA and SHX ($n_{P,t}$, $n_{P,o}$, $n_A$, and $k$).

| Parameter                        | Value                     |
|----------------------------------|---------------------------|
| Number of generations           | 10 $D$                    |
| Population size, $n_p$           | 10 $D$                    |
| Number of offspring, $n_{P,o}$   | 6 $D$                     |
| Number of candidates, $n_{P,t}$  | 3$n_{P,o}$                |
| Archive size, $n_A$              | 30$n_{P,o}$               |
| Number of clusters, $k$          | [n_{P,o}/2]               |

$D$ is the number of dimensions of test functions. All the parameters are fixed throughout the experiments.

### Table 3: The results of the absolute error between the optimal value and the final-generation-elite fitness over 100 runs.

| $f$  | Minimum | Maximum | $p$ value |
|------|---------|---------|-----------|
| D    | Mean    | SD      | Mean      | SD      | p value | Mean    | SD      | p value |
|      | BLX     | SH-BLX_random | SH-BLX_sequential |      |         |         |         |         |         |
| f_1  | 3.37E+00 | 1.58E+01 | 2.07E+00 | 1.51E+01 | 2.45E+00 | 1.36E+01 |      |         |         |
| 5    | 8.82E+00 | 8.92E+00 | 7.27E+00 | 7.26E+00 | 7.13E+00 | 7.35E+00 |      |         |         |
| ± 2.87E+00 | 2.40E+00 | 2.76E+05 | 2.29E+00 | 6.54E+05 |         |         |      |         |         |
| f_2  | 1.03E+01 | 2.58E+01 | 8.60E+00 | 2.28E+01 | 9.20E+00 | 2.68E+01 |      |         |         |
| 10   | 1.80E+01 | 1.84E+01 | 1.72E+01 | 1.68E+01 | 1.66E+01 | 1.69E+01 |      |         |         |
| ± 3.20E+00 | 2.90E+00 | 1.11E+03 | 3.36E+00 | 8.49E+04 |         |         |      |         |         |
| f_3  | 1.07E-01 | 3.40E+00 | 1.19E-02 | 1.69E+00 | 7.60E+00 | 2.14E+00 |      |         |         |
| 5    | 7.75E-01 | 9.28E-01 | 5.67E-01 | 5.57E-01 | 4.47E-01 | 5.63E-01 |      |         |         |
| ± 6.39E-01 | 3.78E-01 | 3.07E-06 | 4.11E-01 | 1.67E-06 |         |         |      |         |         |
| f_4  | 1.02E+00 | 1.59E+01 | 9.51E-01 | 9.47E+00 | 1.29E+00 | 8.85E+00 |      |         |         |
| 10   | 5.02E+00 | 5.33E+00 | 4.14E+00 | 4.24E+00 | 3.79E+00 | 4.09E+00 |      |         |         |
| ± 2.18E+00 | 1.68E+00 | 8.78E-05 | 1.53E+00 | 3.00E-06 |         |         |      |         |         |
| $f$ | $D$ | Minimum | Median | Maximum |
|-----|-----|---------|--------|---------|
|     |     | Mean | SD   | Mean | SD   | Mean | SD   | Mean | SD   |
| $f_3$ | 5.00E + 00 | 1.51E + 00 | 1.60E + 00 | 1.58E + 00 | 5.74E + 00 |
|     | ± 8.6E + 00 | ± 1.3E + 00 | ± 1.26E + 00 | ± 1.26E + 00 | 1.17E - 06 |
|     | ± 4.5E + 00 | ± 5.6E + 01 | ± 2.1E + 00 | ± 4.0E + 01 | ± 4.5E + 01 |
|     | ± 2.4E + 01 | ± 2.6E + 01 | ± 1.2E + 01 | ± 1.2E + 01 | ± 2.0E + 01 |
|     | ± 9.5E + 00 | ± 9.5E + 00 | ± 1.3E - 04 | ± 8.7E + 00 | ± 1.9E - 04 |
|     | ± 1.9E + 03 | ± 2.0E + 03 | ± 7.0E - 05 | ± 1.4E + 03 | ± 9.2E - 07 |
| $f_4$ | 6.0E + 03 | 2.1E + 04 | 2.1E + 03 | 2.1E + 03 | 4.2E + 04 |
|      | ± 3.5E + 03 | ± 4.4E + 03 | ± 1.7E + 03 | ± 7.8E + 04 | ± 1.5E + 05 |
|     | ± 5.1E + 04 | ± 3.9E + 04 | ± 3.0E + 04 | ± 2.1E - 07 |
|     | ± 2.5E + 01 | ± 1.8E + 01 | ± 6.6E + 00 | ± 7.6E + 00 | ± 4.5E + 02 |
|     | ± 1.1E + 02 | ± 1.1E + 02 | ± 9.2E - 03 | ± 8.4E + 01 | ± 1.1E - 03 |
|     | ± 3.9E + 01 | ± 3.8E + 02 | ± 2.2E + 03 | ± 2.2E + 03 | ± 2.3E + 03 |
|     | ± 1.3E + 03 | ± 2.7E + 04 | ± 5.0E - 05 | ± 8.2E + 04 | ± 3.1E - 04 |
|     | ± 1.2E + 01 | ± 1.2E - 01 | ± 1.0E + 03 | ± 3.2E - 01 | ± 1.0E + 03 |
|     | ± 2.4E + 00 | ± 2.4E - 00 | ± 1.9E + 00 | ± 1.9E + 00 | ± 2.0E - 01 |
|     | ± 1.6E + 00 | ± 1.6E + 00 | ± 6.4E - 02 | ± 6.3E - 02 | ± 1.8E - 02 |
|     | ± 1.0E + 00 | ± 1.0E + 00 | ± 4.0E - 04 | ± 4.0E - 04 | ± 1.0E + 00 |
|     | ± 9.5E - 01 | ± 9.5E - 01 | ± 4.0E - 03 | ± 4.0E - 03 | ± 9.5E - 01 |
|     | ± 1.1E - 01 | ± 1.0E - 01 | ± 2.0E + 03 | ± 2.0E + 03 | ± 1.1E + 03 |
|     | ± 1.0E + 00 | ± 1.0E + 00 | ± 6.0E - 02 | ± 6.0E - 02 | ± 1.0E + 00 |
|     | ± 6.6E + 00 | ± 6.6E + 00 | ± 1.9E - 02 | ± 6.4E - 02 | ± 6.6E + 00 |
|     | ± 5.9E + 00 | ± 5.9E + 00 | ± 1.9E + 00 | ± 7.0E + 00 | ± 6.4E - 02 |
|     | ± 6.2E + 00 | ± 6.2E + 00 | ± 1.9E + 00 | ± 6.5E - 01 | ± 2.3E - 08 |
| $f_9$ | 5.0E + 02 | 2.15E + 03 | 5.93E + 02 | 2.23E + 03 | 7.63E + 02 |
|      | ± 2.6E + 02 | ± 2.45E + 02 | ± 2.35E + 02 | ± 2.0E - 03 | ± 2.2E + 03 |
|     | ± 4.1E + 01 | ± 4.1E + 01 | ± 4.1E + 01 | ± 4.1E + 01 | ± 4.1E + 01 |
|     | ± 4.1E - 01 | ± 3.4E - 01 | ± 3.4E - 01 | ± 3.4E - 01 | ± 3.4E - 01 |
|     | ± 1.8E + 02 | ± 1.8E + 02 | ± 1.8E + 02 | ± 1.8E + 02 | ± 1.8E + 02 |
|     | ± 1.8E + 02 | ± 1.8E + 02 | ± 1.8E + 02 | ± 1.8E + 02 | ± 1.8E + 02 |
|     | ± 6.4E - 01 | ± 6.2E - 01 | ± 2.7E - 06 | ± 6.5E - 01 | ± 2.3E - 08 |
### Table 3: Continued.

| $f$     | Min Value | Mean | Min Value | Mean | Min Value | Mean | Min Value | Mean |
|---------|-----------|------|-----------|------|-----------|------|-----------|------|
| $D$     | $f_{12}$  |      | $f_{13}$  |      | $f_{14}$  |      | $f_{15}$  |      |
| 3.70E + 00 | 9.50E + 00 | 2.80E + 00 | 8.91E + 00 | 2.88E + 00 | 9.80E + 00 | 3.09E + 00 | 2.52E + 00 | 4.88E - 01 | 2.03E + 00 | 3.56E - 01 | 2.63E + 00 | 1.11E + 00 | 1.11E + 00 | 1.08E + 00 | 1.08E + 00 | 1.07E + 00 | 1.06E + 00 | 1.14E - 01 | 3.88E - 04 | 3.59E - 01 | 2.14E - 04 | 1.73E + 00 | 4.20E + 00 | 1.80E + 00 | 4.22E + 00 | 1.58E + 00 | 3.68E + 00 | 2.96E + 00 | 2.97E + 00 | 2.72E + 00 | 2.71E + 00 | 2.66E + 00 | 3.04E - 02 | 1.55E - 01 | 4.53E - 02 | 1.21E - 01 | 4.44E - 02 | 1.08E - 01 | 7.84E - 02 | 8.18E - 02 | 7.22E - 02 | 7.21E - 02 | 6.74E - 02 | 6.88E - 02 | 1.99E - 02 | 1.59E - 02 | 2.67E - 04 | 1.44E - 02 | 5.03E - 07 | 1.86E - 03 | 1.27E - 03 | 1.17E - 03 | 1.07E - 02 | 1.26E - 03 | 1.12E - 02 | 3.58E - 03 | 3.58E - 03 | 5.43E - 03 | 4.93E - 03 | 5.12E - 03 | 5.75E - 03 | 5.66E - 03 | 5.28E - 03 | 5.54E - 03 | 4.93E - 03 | 5.12E - 03 | 4.80E - 03 | 4.80E - 03 | 2.58E - 01 | 1.95E - 03 | 2.84E + 02 | 2.73E - 01 | 2.23E + 00 | 1.71E - 01 | 1.48E + 00 | 1.83E - 01 | 2.99E + 00 | 9.30E - 01 | 9.76E - 01 | 6.46E - 01 | 7.10E - 01 | 5.77E - 01 | 6.90E - 01 | 4.08E - 01 | 4.08E - 01 | 2.74E - 01 | 3.92E - 07 | 4.21E - 01 | 5.93E - 09 | 2.11E - 01 | 8.83E - 01 | 1.76E - 01 | 9.73E - 01 | 1.64E - 01 | 8.63E - 01 | 4.73E - 01 | 5.03E - 01 | 3.55E - 01 | 3.78E - 01 | 2.81E - 01 | 3.07E - 01 | 1.29E - 01 | 1.26E - 01 | 4.24E - 02 | 4.24E - 13 | 1.22E - 01 | 3.26E - 21 | 3.08E - 03 | 6.10E - 02 | 1.09E - 03 | 6.74E - 02 | 3.67E - 04 | 3.57E - 02 | 1.54E - 02 | 1.97E - 02 | 7.82E - 03 | 1.11E - 02 | 4.51E - 03 | 7.39E - 03 | 1.33E - 02 | 9.97E - 03 | 2.63E - 09 | 6.87E - 03 | 5.70E - 17 | 9.45E - 04 | 1.11E - 02 | 5.33E - 04 | 7.35E - 03 | 1.68E - 04 | 5.32E - 03 | 3.70E - 03 | 4.22E - 03 | 1.64E - 03 | 1.89E - 03 | 1.08E - 03 | 1.30E - 03 | 2.06E - 03 | 1.07E - 03 | 1.29E - 20 | 8.56E - 04 | 2.16E - 27 | 4.02E - 03 | 4.15E - 01 | 2.48E - 03 | 3.84E - 01 | 2.41E - 03 | 4.92E - 01 | 4.10E - 02 | 5.50E - 02 | 2.21E - 02 | 3.68E - 02 | 1.86E - 02 | 4.22E - 02 | 5.20E - 03 | 8.20E - 02 | 2.42E - 03 | 1.88E - 01 | 1.66E - 03 | 1.36E - 01 | 2.07E - 02 | 2.31E - 02 | 1.31E - 02 | 1.93E - 02 | 6.25E - 03 | 1.38E - 02 | 1.23E - 02 | 2.19E - 02 | 1.04E - 05 | 2.30E - 02 | 4.27E - 17 | 5.72E + 00 | 1.10E + 03 | 2.49E + 00 | 6.26E + 02 | 2.50E + 00 | 2.62E + 03 | 2.12E + 01 | 5.79E + 01 | 1.80E + 01 | 6.39E + 01 | 1.27E + 01 | 7.01E + 01 | 1.29E + 02 | 1.18E + 02 | 1.74E - 01 | 2.77E + 02 | 1.89E - 05 | 1.04E + 01 | 8.49E + 01 | 8.69E + 00 | 2.67E + 02 | 7.58E + 00 | 3.46E + 02 | 1.64E + 01 | 1.88E + 01 | 1.19E + 01 | 2.13E + 01 | 1.06E + 01 | 1.95E + 01 | 3.63E + 01 | 3.58E + 01 | 1.59E - 01 | 1.27E + 02 | 7.85E - 02 | 3.06E + 01 | 2.12E + 00 | 3.44E + 00 | 1.92E + 00 | 5.01E + 00 | 1.58E + 00 | 4.20E + 00 | 2.80E - 01 | 6.40E + 00 | 4.15E - 02 | 9.05E - 02 | 6.01E + 00 | 2.58E - 02 | 9.01E - 01 | 1.20E + 00 | 5.28E - 01 | 8.65E - 01 | 5.03E - 01 | 1.19E + 00 | 1.04E + 00 | 9.14E - 01 | 6.06E - 07 | 1.79E + 00 | 9.06E - 06 |
| $f$ | $D$ | Minimum | Median | Maximum | $p$ value |
|-----|-----|---------|--------|---------|-----------|
|     |     | Value   | Mean   | SD      | Value     | Mean   | SD      | Value     | Mean   | SD      | Value     |
|     |     | Minimum |        | Maximum |           |        | Minimum |        | Maximum |           |        | Minimum |        | Maximum |
| $f_1$ | 1.46E+00 | 2.49E+01 | 5.51E-01 | 1.84E+01 | 1.93E-01 | 1.90E+01 |
| $f_2$ | 5.66E+00 | 6.28E+00 | 3.13E+00 | 3.72E+00 | 2.71E+00 | 4.45E+00 |
|     | ±1.35E+00 | ±2.47E+00 | ±1.05E-15 | ±4.19E+00 | ±2.78E-10 | 
| $f_3$ | 2.16E+00 | 1.11E+01 | 1.42E+00 | 3.24E+01 | 9.44E-01 | 8.43E+00 |
|     | ±1.35E+00 | ±3.25E+00 | ±1.25E-17 | ±1.32E+00 | ±1.58E-21 | 
| $f_4$ | 5.87E-03 | 1.12E+00 | 4.99E-03 | 1.34E+00 | 3.09E-03 | 3.30E+00 |
|     | 8.86E-02 | 1.65E-01 | 5.66E-02 | 1.38E-01 | 8.61E-02 | 2.95E-01 |
|     | ±1.96E-01 | ±2.15E-01 | ±3.69E-03 | ±5.54E-01 | ±1.90E-01 | 
| $f_5$ | 1.36E-02 | 1.19E+00 | 4.93E-03 | 1.22E+00 | 4.72E-03 | 1.86E+00 |
|     | 6.32E-02 | 1.24E-01 | 2.42E-02 | 2.75E-02 | 3.80E-02 | 1.35E-01 |
|     | ±1.66E-01 | ±1.55E-01 | ±7.46E-09 | ±2.49E-01 | ±2.72E-03 | 
| $f_6$ | 3.58E-03 | 1.92E-01 | 1.20E-03 | 9.98E-01 | 9.28E-04 | 9.98E-01 |
|     | 1.73E-02 | 2.54E-02 | 6.69E-03 | 5.27E-02 | 8.29E-03 | 6.27E-02 |
|     | ±2.82E-02 | ±1.88E-01 | ±1.08E-10 | ±1.96E-01 | ±9.22E-05 | 
| $f_7$ | 2.42E+00 | 2.32E+00 | 9.43E-01 | 2.03E+01 | 6.68E-01 | 2.03E+01 |
|     | 1.12E+01 | 1.18E+01 | 1.02E+01 | 1.02E+01 | 8.62E+00 | 8.86E+00 |
|     | ±3.95E-00 | ±3.89E+00 | ±4.58E-03 | ±3.45E-00 | ±8.84E-09 | 
| $f_8$ | 1.44E+01 | 4.54E+01 | 3.53E+00 | 3.65E+01 | 8.75E-01 | 2.79E+01 |
|     | 3.47E-01 | 3.37E+01 | 1.52E+01 | 1.71E+01 | 8.14E+00 | 9.53E+00 |
|     | ±6.66E+00 | ±7.99E+00 | ±6.39E-27 | ±5.97E+00 | ±1.87E-33 | 
| $f_9$ | 4.04E+01 | 4.13E+01 | 4.03E+01 | 4.15E+01 | 4.02E+01 | 4.11E+01 |
|     | 4.07E-01 | 4.07E-01 | 4.06E+01 | 4.06E+01 | 4.05E+01 | 4.05E+01 |
|     | ±1.90E-01 | ±1.60E-01 | ±4.35E-10 | ±1.65E-01 | ±8.70E-15 | 
| $f_{10}$ | 1.81E+02 | 1.81E+02 | 1.80E+02 | 1.81E+02 | 1.80E+02 | 1.81E+02 |
|     | 1.81E+02 | 1.81E+02 | 1.81E+02 | 1.81E+02 | 1.81E+02 | 1.81E+02 |
|     | ±1.29E-01 | ±1.08E-01 | ±4.36E-18 | ±1.28E-01 | ±1.14E-25 | 
| $f_{11}$ | 4.05E-01 | 3.28E+00 | 2.13E-01 | 3.14E+00 | 1.97E-01 | 2.65E+00 |
|     | 1.87E+00 | 1.84E+00 | 1.45E+00 | 1.49E+00 | 1.12E+00 | 1.23E+00 |
|     | ±5.88E-01 | ±6.29E+00 | ±4.46E-05 | ±5.57E-01 | ±1.65E-11 | 
| $f_{12}$ | 2.67E-01 | 1.39E+00 | 1.57E-01 | 2.07E-00 | 8.07E-02 | 5.88E-01 |
|     | 5.83E-01 | 6.16E-01 | 3.50E-01 | 3.96E-01 | 2.36E-01 | 2.45E-01 |
|     | ±2.25E-01 | ±2.25E-01 | ±2.84E-16 | ±9.53E-02 | ±4.47E-30 | 
| $f_{13}$ | 1.04E-01 | 6.17E-01 | 4.94E-02 | 5.92E-01 | 9.47E-02 | 5.98E-01 |
|     | 3.49E-01 | 3.52E-01 | 3.12E-01 | 3.20E-01 | 3.31E-01 | 3.27E-01 |
|     | ±1.11E+01 | ±1.12E+01 | ±2.86E-02 | ±1.09E+01 | ±7.84E-02 | 
| $f_{14}$ | 6.96E-02 | 4.84E-01 | 1.43E-02 | 3.90E-01 | 1.22E+00 | 2.88E+01 |
|     | 1.93E-01 | 2.08E-01 | 7.37E-02 | 9.07E-02 | 5.03E-02 | 6.06E-02 |
|     | ±1.00E-01 | ±6.20E-02 | ±3.90E-01 | ±6.30E-20 | ±4.44E-02 | 6.33E-28 |
| $f_{15}$ | 2.00E-01 | 7.73E-01 | 1.21E-01 | 7.01E-01 | 2.00E-01 | 7.10E-01 |
|     | 4.13E-01 | 4.24E-01 | 4.00E-01 | 3.98E-01 | 4.00E-01 | 3.86E-01 |
|     | ±1.39E+01 | ±1.30E-01 | ±1.16E-01 | ±1.22E+01 | ±3.43E-02 | 
| $f_{16}$ | 2.00E-01 | 7.03E-01 | 2.00E-01 | 5.14E-01 | 2.00E-01 | 6.16E-01 |
|     | 3.65E-01 | 3.66E-01 | 3.10E-01 | 3.25E-01 | 3.08E-01 | 3.35E-01 |
|     | ±8.08E-02 | ±6.94E-02 | ±2.20E-04 | ±9.77E-02 | ±1.57E-03 | 

Table 3: Continued.
| $f$     | $D$ | Minimum Mean | Maximum Mean | Minimum Mean | Maximum Mean | Minimum Mean | Maximum Mean |
|--------|-----|--------------|--------------|--------------|--------------|--------------|--------------|
|        |     | Median SD    |              | Median SD    |              | Median SD    |              |
|        |     |              |              |              |              |              |              |
| $f_{15}$ | 5   | $4.37E-02$   | $1.27E-01$   | $4.24E-02$   | $1.32E-01$   | $4.28E-02$   | $8.59E-02$   |
|         |     | $7.25E-02$   | $7.30E-02$   | $5.54E-02$   | $5.82E-02$   | $5.12E-02$   | $5.45E-02$   |
|         |     | $1.08E-03$   | $9.64E-03$   | $6.29E-04$   | $5.55E-03$   | $6.01E-04$   | $3.29E-03$   |
|         |     | $1.89E-03$   | $5.85E-04$   | $1.22E-03$   | $1.52E-03$   | $8.99E-04$   | $1.01E-03$   |
| $f_2$  | 5   | $2.77E+00$   | $2.83E+00$   | $2.18E+00$   | $2.30E+00$   | $1.96E+00$   | $2.13E+00$   |
|         |     | $4.93E+00$   | $1.97E+00$   | $3.09E-01$   | $1.81E+00$   | $3.58E-01$   | $1.95E-01$   |
| $f_3$  | 5   | $4.35E+00$   | $4.71E-01$   | $2.17E-01$   | $2.96E-01$   | $2.93E-01$   | $2.71E-01$   |
|         |     | $5.48E+00$   | $5.77E+00$   | $1.31E-01$   | $1.80E+00$   | $3.85E+00$   | $2.10E+00$   |
| $f_4$  | 5   | $4.20E+00$   | $4.74E-01$   | $2.98E-01$   | $3.03E-01$   | $2.49E-01$   | $2.68E-01$   |
|         |     | $9.70E+00$   | $5.08E+00$   | $2.47E+00$   | $6.39E+00$   | $4.47E+00$   | $4.88E+00$   |
| $f_5$  | 5   | $1.51E+00$   | $2.41E+02$   | $8.81E+01$   | $1.93E+02$   | $1.00E+02$   | $1.66E+02$   |
|         |     | $1.04E+02$   | $3.64E+03$   | $8.64E+01$   | $1.54E+03$   | $9.04E+01$   | $3.07E+03$   |
| $f_6$  | 5   | $6.38E+02$   | $7.67E+02$   | $4.29E+02$   | $4.78E+02$   | $3.65E+02$   | $4.71E+02$   |
|         |     | $6.70E+01$   | $8.64E+01$   | $2.11E+01$   | $2.42E+02$   | $2.50E+00$   | $6.09E+01$   |
| $f_7$  | 5   | $5.54E+00$   | $5.85E+01$   | $1.27E+01$   | $1.73E+01$   | $1.60E+01$   | $1.80E+01$   |
|         |     | $6.04E+00$   | $5.85E+01$   | $1.27E+01$   | $1.73E+01$   | $1.60E+01$   | $1.80E+01$   |
| $f_8$  | 5   | $1.54E+01$   | $3.08E+03$   | $1.46E+01$   | $5.00E+02$   | $1.39E+01$   | $1.19E+03$   |
|         |     | $5.27E+01$   | $1.50E+02$   | $3.49E+01$   | $5.30E+01$   | $3.08E+01$   | $5.44E+01$   |
| $f_9$  | 5   | $5.84E+01$   | $1.16E+00$   | $5.35E-01$   | $8.39E-01$   | $5.39E-01$   | $9.49E-01$   |
|         |     | $1.02E+00$   | $7.72E-01$   | $3.33E-03$   | $9.58E-01$   | $1.35E-02$   | $5.88E-01$   |
| $f_{10}$ | 10  | $1.84E+00$   | $1.60E+01$   | $2.16E+00$   | $1.40E+01$   | $8.81E-01$   | $1.48E+01$   |
|         |     | $6.18E+00$   | $6.60E+00$   | $5.08E+00$   | $5.83E+00$   | $5.21E+00$   | $5.65E+00$   |
| $f_{11}$ | 10  | $8.00E-01$   | $1.00E+00$   | $7.95E-01$   | $1.00E+00$   | $4.05E-01$   | $1.00E+00$   |
|         |     | $1.00E+00$   | $9.92E-01$   | $9.96E-01$   | $9.78E-01$   | $9.99E-01$   | $9.72E-01$   |
| $f_{12}$ | 10  | $2.60E-02$   | $4.13E-02$   | $6.41E-08$   | $8.29E-02$   | $1.42E-04$   | $1.42E-04$   |

Table 3: Continued.
6.2.2. Analysis on SPX vs. SH-SPX. SPX [21] is a better alternative of BLX, and we can observe from Table 3 that SPX noticeably outperforms BLX. From Table 3, it is also very clear that SHX further boosts the performance of SPX to a large extent. In particular, the results of minimum and median are improved by involving SHX for all settings. As pointed out in [21], SPX has the ability to maintain the mean and covariance of the parent individuals, which is consistent with the design guideline of good crossover operators mentioned in Section 3. Since SHX manages an archive that stores search history over few generations, it can preserve some useful statistics (e.g., centroids of clusters) much longer. That is why SHX is able to enhance SPX.

6.2.3. Analysis on UNDX vs. SH-UNDX. Similar to BLX and SPX, Table 3 shows that the results involving SHX are improved in most settings. UNDX is also designed to generate offspring inheriting the distribution of the parent individuals [35]. Therefore, statistics of the search history provided by SHX are useful for UNDX to enhance search ability.
(a) Figure 2: Continued.
Figure 2: Continued.
Figure 2: Continued.
Figure 2: Continued.
Figure 2: Continued.
6.3. Comparison in Convergence Curve. With the aid of search history, SHX not only achieves better results but also improves the convergence speed. In this section, we compare the generation alternation for over all the test functions in the case of $D = 10$. Evaluation values of elite individuals from the 1st generation to the 100th generation are plotted in Figure 2. The mean value of 100 trials is represented by the line, and the range between the minimum and the maximum is represented by the shaded area. Smaller area means more stable search. It should be noted that as the ranges of parameters are not constrained during the search procedure, methods can achieve infinitely small values of fitness, and a lower value does not mean a better result in the case of $f_{10}$, as explained in Section 6.2.

For BLX, SPX, and UNDX, exploiting SHX shows faster convergence speed comparing against them without SHX in most cases. The superiority becomes more obvious when the problem setting is more difficult (e.g., multimodal functions $f_{8-15}$ vs. unimodal functions $f_{1-7}$).

6.4. Comparison in Processing Time. In this section, we show the runtime overhead of the processing brought by SHX. Figure 3 shows the comparisons in processing time of an optimization task ($D = 10$ and a single fitness evaluation takes 0.01 second) for BLX and SPX. The parameter setting follows Table 2, and all the results are averaged over 10 trials. It took 93.9 seconds and 94.1 seconds for BLX and SPX to complete the entire process, respectively. SH-BLX_random took additional 1.7 seconds to BLX. SH-BLX_sequential took 1.6 seconds more than BLX. Similarly, the additional runtime for SH-SPX_random and SH-SPX_sequential to SPX were 3.9 seconds and 3.9 seconds, respectively. These numbers demonstrate the additional runtime only occupies a small part of the total processing time. These additional computational costs mainly occur in the clustering with archive data and the label assignment with candidate offspring. The cost can be further reduced by fusing efficient distance measure or parallel computing. For a fixed size of an archive, the runtime grows linearly with the increase in the
number of generations. Considering the complexity of the fitness function and the budget, SHX is a practical alternative to other crossover models.

7. Conclusions

In this paper, we have proposed a novel crossover model (SHX) which is simple yet effective and efficient. It can be easily integrated with any existing crossover operators. The key idea is to exploit search history over generations to gain useful information for generating offspring. Experimental results demonstrate that our SHX can significantly boost the performance of existing crossovers, in terms of the final results demonstrate that our SHX can significantly boost the performance of existing crossovers, in terms of the final number of generations. Considering specific contexts, and parallelization can be easily integrated with any existing crossover operators. The key idea is to exploit search history over generations to gain useful information for generating offspring. Experimental results demonstrate that our SHX can significantly boost the performance of existing crossovers, in terms of the final solution and the convergence speed. Also, according to experiments, the induced extra runtime is negligible compared to the total processing time.

SHX still has a few limitations. (1) Additional hyperparameters need to be determined. (2) The induced additional runtime may be unable to sufficiently support applications which require high processing speed. As the future work, we would like to address the above limitations. For instance, hyperparameters can be adaptively set by considering specific contexts, and parallelization can be introduced to speed up SHX.

Data Availability

The test data used to support the findings of this study are included within the article.

Disclosure

A preliminary version of this work appears in GECCO2020 and has also been mentioned in the manuscript which can be viewed at the following link: https://arxiv.org/abs/2003.13508.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partly supported by JSPS KAKENHI, Grant number (JP18K17823).

References

[1] A. H. Wright, “Genetic algorithms for real parameter optimization,” in Foundations of Genetic Algorithms, vol. 1, pp. 205–218, Elsevier, Amsterdam, Netherlands, 1991.
[2] F. Herrera, M. Lozano, and J. L. Verdegay, “Tackling real-coded genetic algorithms: operators and tools for behavioural analysis,” Artificial Intelligence Review, vol. 12, no. 4, pp. 265–319, 1998.
[3] D. Whitley, “An overview of evolutionary algorithms: practical issues and common pitfalls,” Information and Software Technology, vol. 43, no. 14, pp. 817–831, 2001.
[4] D. Sholomon, O. David, and N. S. Netanyahu, “A genetic algorithm-based solver for very large jigsaw puzzles,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1767–1774, Portland, OR, USA, June 2013.
[5] L. Xie and A. Yuille, “Genetic CNN,” in Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), pp. 1379–1388, Venice, Italy, October 2017.
[6] E. Real, S. Moore, A. Selle et al., “Lare-scale evolution of image classifiers,” in Proceedings of the International Conference on Machine Learning (ICML), pp. 2902–2911, Sydney, Australia, August 2017.
[7] K. Deep and M. Thakur, “A new crossover operator for real coded genetic algorithms,” Applied Mathematics and Computation, vol. 188, no. 1, pp. 895–911, 2007.
[8] C. Garcia-Martinez, M. Lozano, F. Herrera, D. Molina, and A. M. Sánchez, “Global and local real-coded genetic algorithms based on parent-centric crossover operators,” European Journal of Operational Research, vol. 185, no. 3, pp. 1088–1113, 2008.
[9] P.-H. Tang and M.-H. Tseng, “Adaptive directed mutation for real-coded genetic algorithms,” Applied Soft Computing, vol. 13, no. 1, pp. 600–614, 2013.
[10] S. Picek, D. Jakobovic, and M. Golub, “On the recombinator operator in the real-coded genetic algorithms,” in Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 3103–3110, IEEE, Cancun, Mexico, June 2013.
[11] Y.-C. Chuang, C.-T. Chen, and C. Hwang, “A simple and efficient real-coded genetic algorithm for constrained optimization,” Applied Soft Computing, vol. 38, pp. 87–105, 2016.
[12] F. Herrera, M. Lozano, and A. M. Sánchez, “A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study,” International Journal of Intelligent Systems, vol. 18, no. 3, pp. 309–338, 2003.
[13] M. Crapišek, S.-H. Liu, and M. Mernik, “Exploration and exploitation in evolutionary algorithms: a survey,” ACM Computing Surveys, vol. 45, pp. 1–33, 2013.
[14] H.-G. Beyer and K. Deb, “On self-adaptive features in real-parameter evolutionary algorithms,” IEEE Transactions on Evolutionary Computation, vol. 5, no. 3, pp. 250–270, 2001.
[15] S. Lloyd, “Least squares quantization in pcm,” IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129–137, 1982.
[16] T. Nakane, X. Lu, and C. Zhang, “SHX: search history driven crossover for real-coded genetic algorithm,” 2020, https://arxiv.org/abs/2003.13508.

[17] L. J. Eshelman and J. D. Schaffer, “Real-coded genetic algorithms and interval-schemata,” in *Foundations of Genetic Algorithms*, vol. 2, pp. 187–202, Elsevier, Amsterdam, Netherlands, 1993.

[18] L. J. Eshelman, K. E. Mathias, and J. D. Schaffer, “Crossover operator biases: exploiting the population distribution,” in *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA)*, East Lansing, MI, USA, July 1997.

[19] K. Deb and R. B. Agrawal, “Simulated binary crossover for continuous search space,” *Complex Systems*, vol. 9, pp. 115–148, 1995.

[20] I. Ono and S. Kobayashi, “A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover,” in *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA)*, pp. 246–253, East Lansing, MI, USA, 1997.

[21] S. Tsutsui, M. Yamamura, and T. Higuchi, “Multi-parent recombination with simplex crossover in real coded genetic algorithms,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 657–664, Orlando, FL, USA, July 1999.

[22] Y. Sano and H. Kita, “Optimization of noisy fitness functions by means of genetic algorithms using history of search,” in *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN)*, Springer, Paris, France, pp. 571–580, September 2000.

[23] H. B. Amor and A. Rettinger, “Intelligent exploration for genetic algorithms: using self-organizing maps in evolutionary computation,” in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation—GECCO’05*, pp. 1531–1538, Washington, DC, USA, June 2005.

[24] S. Y. Yuen and C. K. Chow, “Continuous non-revisiting genetic algorithm,” in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pp. 1896–1903, IEEE, Trondheim, Norway, May 2009.

[25] H. Kita, “A comparison study of self-adaptation in evolution strategies and real-coded genetic algorithms,” *Evolutionary Computation*, vol. 9, no. 2, pp. 223–241, 2001.

[26] Y. Akimoto, R. Hasada, J. Sakuma, I. Ono, and S. Kobayashi, “Generation alternation model for real-coded ga using multi-parent: proposal and evaluation of just generation gap (JGG),” in *Proceedings of the 19th SICE Symposium on Decentralized Autonomous Systems*, pp. 341–346, Tokyo, Japan, January 2007.

[27] S. Kobayashi, “The frontiers of real-coded genetic algorithms,” *Transactions of the Japanese Society for Artificial Intelligence*, vol. 24, no. 1, pp. 147–162, 2009.

[28] D. E. Goldberg, “Genetic algorithms in search, optimization, and machine learning,” 1989.

[29] J. Wang, M. Zhang, O. K. Ersoy, K. Sun, and Y. Bi, “An improved real-coded genetic algorithm using the heuristical normal distribution and direction-based crossover,” *Computational Intelligence and Neuroscience*, vol. 2019, Article ID 4243853, 17 pages, 2019.

[30] E.-u. Haq, I. Ahmad, A. Hussain, and I. M. Almanjahie, “A novel selection approach for genetic algorithms for global optimization of multimodal continuous functions,” *Computational Intelligence and Neuroscience*, vol. 2019, Article ID 8640218, 14 pages, 2019.

[31] M. Jamil and X. S. Yang, “A literature survey of benchmark functions for global optimisation problems,” *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

[32] I. Fister, S. Fong, and J. Brest, “A novel hybrid self-adaptive bat algorithm,” *The Scientific World Journal*, vol. 2014, Article ID 709738, 2 pages, 2014.

[33] M. N. Ab Wahab, S. Nefzi-Meziani, and A. Atyabi, “A comprehensive review of swarm optimization algorithms,” *PLoS One*, vol. 10, no. 5, Article ID e0122827, 2015.

[34] I. Ono, H. Kita, and S. Kobayashi, “A real-coded genetic algorithm using the unimodal normal distribution crossover,” in *Advances in Evolutionary Computing*, pp. 213–237, Springer, Berlin, Germany, 2003.

[35] H. Kita, I. Ono, and S. Kobayashi, “Theoretical analysis of the unimodal normal distribution crossover for real-coded genetic algorithms,” in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation Proceedings*, pp. 529–534, Anchorage, AK, USA, May 1998.