Check for updates

A novel ML-driven test case selection approach for enhancing the performance of grammatical evolution

Krishn Kumar Gupt[1]*, Meghana Kshirsagar[2]*,
Douglas Mota Dias[2], Joseph P. Sullivan[1] and Conor Ryan[2]

[1]Department of Electrical and Electronic Engineering, Technological University of the Shannon: Midlands Midwest, Limerick, Ireland, [2]Department of Computer Science and Information Systems, University of Limerick, Limerick, Ireland

Computational cost in metaheuristics such as Evolutionary Algorithm (EAs) is often a major concern, particularly with their ability to scale. In data-based training, traditional EAs typically use a significant portion, if not all, of the dataset for model training and fitness evaluation in each generation. This makes EA suffer from high computational costs incurred during the fitness evaluation of the population, particularly when working with large datasets. To mitigate this issue, we propose a Machine Learning (ML)-driven Distance-based Selection (DBS) algorithm that reduces the fitness evaluation time by optimizing test cases. We test our algorithm by applying it to 24 benchmark problems from Symbolic Regression (SR) and digital circuit domains and then using Grammatical Evolution (GE) to train models using the reduced dataset. We use GE to test DBS on SR and produce a system flexible enough to test it on digital circuit problems further. The quality of the solutions is tested and compared against state-of-the-art and conventional training methods to measure the *coverage* of training data selected using DBS, i.e., how well the subset matches the statistical properties of the entire dataset. Moreover, the effect of optimized training data on run time and the effective size of the evolved solutions is analyzed. Experimental and statistical evaluations of the results show our method empowered GE to yield superior or comparable solutions to the baseline (using the full datasets) with smaller sizes and demonstrates computational efficiency in terms of speed.

KEYWORDS

test case selection, symbolic regression, digital circuit, grammatical evolution, clustering, fitness evaluation, solution size, time analysis

1 Introduction

EAs are well-known for their applications in different problem domains. One of the main features of EAs such as GE is using fitness evaluations to achieve a solution iteratively. This is the only step that guides how the chromosomes will change (using evolutionary operators) over time. As Kinnear et al. (1994) explains, "your fitness function is the only chance that you have to communicate your intentions to the powerful process that Genetic Programming represents." A fitness function, therefore, needs to do more than measure the performance of an individual; it needs to score individuals in such a way that they can be compared, and a more successful individual can be distinguished from a less one. Fitness evaluation time depends on several factors, such as the number of generations, population size, and

test case volume (number of variables/features and instances). It is one of the most time-consuming steps (Yang et al., 2003). Usually, the evolutionary parameters are user-based and can be tuned as per convenience and experience, but the test case is something we have less control over. A test case is a specific set of inputs, conditions, and expected outcomes used to assess the functionality and behavior of a design.

In the modern world, due to the ever-growing volume of test cases in problems like SR, digital circuits, etc., the use of GE becomes increasingly computationally costly and sometimes even impractical. Since these problem models learn and adapt by using data samples, their evolution is dependent on the training data[1] up to a great extent. At the same time, the greater the training data volume, the higher the evaluation time—the computational time required to evaluate the model on data—required. We preferred using GE over other data-based learning algorithms as it combines the power of Genetic Programming (GP) with the flexibility of context-free grammar (details in Section 3), enabling it to handle a wide range of problems domains and evaluate the proposed algorithm.

One approach to speed up the evolutionary process is to reduce the amount of data that needs to be processed at each generation. This is feasible because, in general, not all instances of the training data are equally important. Usually, it is not known in advance to a test engineer which data samples will be most useful, particularly in the case of a "black-box" model. For example, in domains such as digital circuit design, where stringent testing is crucial but exhaustive testing is not viable, test engineers play a vital role in ensuring that products perform as required.

In the past few years, real-world ML datasets have experienced a very significant increase in size largely due to superior data collection techniques and cheaper, faster storage such as Solid-State Drives (SSDs). As the world of Big Data has experienced, however, simply increasing the amount of training data doesn't necessarily improve the model-building process for ML. Too much data can lead to slower and more expensive training, and noise and redundancy within the dataset can negatively impact overall model accuracy (Duffy-Deno, 2021). For example, consider a dataset with millions of data instances, at least some of which are redundant. If one could successfully train an accurate model on a subset, then costs would be lower, perhaps significantly so, depending on the size of the subset. However, too small of a subset can lead to the missing of important patterns and subsequently produce less accurate models. This means the selected subset should statistically represent the appropriate distribution of the total test cases.

When using a large amount of training data in problems like SR or digital circuits, GE can require extensive resource consumption in fitness evaluation while evaluating test cases against each individual being trained. This makes the process more expensive for problems running for a large number of generations and/or population size or performing multiple runs. An effective test case selection, in such scenarios, can potentially contribute not only to improving the efficiency but also the accuracy of the model. However, while evolving SR models using GE, test case

selection before learning is rarely considered, although several studies employed feature selection (Ali et al., 2022). There are some existing works in the case of digital circuits (Ryan et al., 2021; Gupt et al., 2022a), but test case selection at the black-box level is still challenging and has much potential. Clustering is a standard technique to identify data instances based on their latent relationships, specifically to identify which instances are the most alike (Bindra et al., 2021; Kshirsagar et al., 2022). We use it as a pre-processing step to obtain a useful subset of the test cases to test the model.

A key challenge with data selection is to identify how small the subset can be without compromising eventual model quality. We performed test case selection on six different fractions of the total data, as described in Section 5.2 and Supplementary Table S5. This article introduces a novel approach to extract minimal test cases while maintaining a high level of data diversity, thereby eliminating any algorithmic bias. Moreover, our approach is problem-agnostic, as evidenced by its suitability in selecting test cases from diverse domains. We demonstrate its efficacy on 24 sets of benchmarks from the domains of SR (10 synthetic datasets and 10 real-world datasets) and digital circuits (four problems).

The algorithm offers an evolutionary advantage of competitive solutions to the examined baselines (where a larger dataset is used) regarding the test score and time efficiency. The contributions from our research are summarized below. Briefly, we

- Propose DBS, a problem-agnostic test case selection algorithm;
- Use the DBS algorithm to select training data to decrease the number of fitness evaluations during model training and, hence, reduces GE run time;
- Demonstrate how DBS performs at least as well as—and sometimes significantly better—in terms of achieving better test scores than the baseline considered;
- Show that the solutions produced using the training data selected by DBS sometimes have interesting results on effective solution size, a phenomenon that merits further investigation;
- Demonstrate our results by validating DBS algorithm on 24 problems, laying the foundation for future studies that investigate this approach in other problem domains and more complex problems.

This article is organized as follows. Section 2 describes the current research on test case selection in the domain of SR and digital circuits. In Section 3, we present the evolutionary approach of GE used in this paper along with the basic foundation. In Section 4, a detailed description of the proposed DBS algorithm is presented. The benchmark problems tested and the methodology used in the research are discussed in Section 5. We discuss our findings and compare them to the baseline results along with statistical validations in Section 6 and conclude the contributions in Section 7.

2 Background

This section reviews related studies on test case selection along with a brief background of SR and digital circuits. We roughly

---

1   For ease of understanding, test cases used to train the model are referred to as training data. Similarly, test cases used to test the model are referred to as testing data.

divide the related studies into two categories: test case selection for SR and digital circuits.

2.1 Symbolic regression

SR is a method that creates models in the form of analytic equations that can be constructed by using even a small amount of training data, as long as it is representative of the eventual training data (Kubalík et al., 2020). Sometimes, information or certain characteristics of the modeled system are known in advance, so the training set can be reduced using this knowledge (Arnaiz-González et al., 2016b). Instance selection as known in SR, sometimes referred to as *test case selection* in digital circuits, in such cases can significantly impact the efficiency and accuracy of the model's learning process by strategically choosing representative data points for evaluation.

In recent years, different studies have used test case selection in the field of SR and digital circuits. Kordos and Blachnik (2012) introduced instance selection for regression problems, leveraging solutions from Condensed Nearest Neighbor (CNN) and Edited Nearest Neighbor (ENN) typically used in classification tasks. In this study, we've incorporated it as a baseline for SR problems. Another work by the same researchers used the k-Nearest Neighbor (k-NN) algorithm for data selection in SR problems and later evaluated them using multi-objective EAs (Kordos and Łapa, 2018).

Son and Kim (2006) proposed an algorithm where the dataset was divided into several partitions. An entropy value is calculated for each attribute in each partition, and the attribute with the lowest entropy is used to segment the dataset. Next, Euclidean distance is used to locate the representative instances that represent the characteristics of each partition. Kajdanowicz et al. (2011) used clustering to group the dataset and distance based on the entropy variance for comparing and selecting training datasets. Their instance selection method computes distances between training and testing data, creates a ranking based on these distances, and selects a few closest datasets. This raises concerns about biased training, as it predominantly utilizes data that closely resemble the testing data for training purposes.

Numerous instance selection algorithms have been developed so far. A cluster-based instance selection algorithm utilizing a population-based learning approach is introduced in Czarnowski and Jędrzejowicz (2010) and Czarnowski (2012). This method aims to enhance the efficiency of instance selection by grouping similar data instances into clusters and then employing a population-based learning algorithm to identify representative instances. While effective for classification problems, it's worth noting that the population-based learning algorithm can be computationally demanding, especially with large datasets, due to its inherent complexity and extensive computations involved in identifying suitable representatives. The instance reduction in Czarnowski and Jędrzejowicz (2003) relies on a clustering algorithm, which, however, does not take advantage of additional information, such as data diversity within the cluster. As a result, it might inadvertently pick similar cases from within a cluster. The instance removal algorithm in Wilson and Martinez (2000) is based on

the relationships between neighboring instances. Nonetheless, the approach may benefit from enhanced user control over the resulting subset's size. Further, the method struggles with noisy instances, acting as border points, that can disrupt the removal order.

While various techniques have been proposed for data selection or reduction in ML tasks, most studies have concentrated on classification tasks (Wilson and Martinez, 2000; Czarnowski and Jędrzejowicz, 2003, 2010; Son and Kim, 2006; Czarnowski, 2012). In contrast, little attention has been paid to the challenges related to instance selection in SR (Arnaiz-González et al., 2016a; Chen et al., 2017).

2.2 Digital circuits

Digital circuits are of utmost importance in our daily lives, powering everything from alarm clocks to smartphones. A clear and concise way to represent the features of a digital design is by using a high-level language or Hardware Description Language (HDL). HDLs are specialized and high-level programming languages used to describe the behavior and structure of digital circuits. Designing a digital circuit begins with designing its specifications in HDLs and ends with manufacturing, with several testing processes involved throughout the different design phases. There are many HDL available, but Verilog and VHSIC Hardware Description Language (VHDL) are the most well-known HDL and are frequently used in digital design (Ferdjallah, 2011). In this study, we use VHDL as the design language for the circuits we examine, although Verilog would have been a perfectly acceptable alternative.

Testing is hugely important when designing digital circuits, as errors during fabrication can be prohibitively expensive to fix. This has resulted in the creation of extremely powerful and accurate circuit simulators (Tan and Rosdi, 2014). Numerous HDL simulators are available, such as ModelSim, Vivado, Icarus, and GHDL, to name a few. During simulation, test scenarios are carefully chosen to reflect the circuits' desired behaviors in actual applications. A typical simulation session has the circuit model encapsulated in a testbench that consists of a response analyzer and other important components (Gupt et al., 2021a). Testbenches are essentially HDL programs containing test cases, which are intended to assure good coverage through testing (Gupt et al., 2021b). Typically, hardware simulation is much slower than software simulation (Hsiung et al., 2018), which is one more reason to adopt test case selection and reduce the design and testing time. With this in mind, it is reasonable to expect that the benefit experienced in the circuit design experiments will be more noticeable than in SR experiments.

Many studies have used automatic and semi-automated test case selection methods. These methods' main objective is to provide high-quality tests, ideally through automated techniques, that provide good coverage while reducing the overall number of tests (Mutschler, 2014). The fundamental testing method, random testing, often involves randomly choosing test cases from the test case space (the set of all potential inputs) (Orso and Rothermel, 2014). Although it is cost-effective and easy to implement, it

may result in several issues, such as selecting redundant test cases, missing failure causing inputs, or lower coverage (Mrozek and Yarmolik, 2012). Kuhn and Okun (2006) proposed pseudo-exhaustive testing that involves exhaustively testing pieces of circuits instead of the complete circuits. Based on the number of defects the test cases can identify, Thamarai et al. (2010) suggested reducing the number of test cases in their work. They suggested heuristic techniques for determining the most crucial test cases for circuits with smaller test sets. Muselli and Liberati (2000) suggested clustering as a method for classifying problems involving binary inputs. In their research, they compared input patterns and their proximity to one another using Hamming distance to create clusters.

Bushnell and Agrawal (2002) provide insights on algorithms such as Automatic Test-Pattern Generation (ATPG) for generating test cases for structural testing of digital circuits, which can find redundant or unnecessary circuit logic and are useful to compare whether one circuit's implementation matches another's implementation. They highlight the impracticability of performing exhaustive testing of a 64-bit ripple-carry adder with 129 inputs and 65 outputs, suggesting that such testing can be suitable for only small circuits. In contrast, modern-day circuits tend to have an exponential volume of test cases.

Mrozek and Yarmolik (2012) proposed *anti-random testing* for Built In Self Test (BIST) and used Hamming distance as a measure to differentiate the input test cases from previously applied ones. In an extended work, Mrozek and Yarmolik (2017) proposed an optimized controlled random test where they used information from the previous tests to maintain as much distance as possible among input test cases. However, the quality of the generated test cases depends on the initial random test case; if it does not cover a diverse set of conditions, the subsequent test cases may not effectively explore the full range of possible inputs.

Although some studies have proposed test case selection in SR or digital circuits, they have not been explored and evaluated with metaheuristics like GE. Moreover, the existing methods are problem-specific, i.e., their test case selection algorithms apply to either SR or digital circuits. Our proposed DBS algorithm is problem agnostic and, with a minor tweak (based on the type of data), can be easily applied to both problem domains. We decided to evaluate it using GE (however, it can be used with other EAs, see Section 6 for details), where DBS is first used to select a subset of test cases and use them as training data for the model. This choice has the advantage of obtaining solutions with low prediction error [typically expressed as Root Mean Square Error (RMSE), for SR problems] or a high number of successful test cases (expressed at *hit-count* in the case of digital circuit design) using substantially reduced data. The obtained solutions using GE are tested against the testing data to calculate their accuracy or, alternatively, the quality of training data and strength of DBS algorithm.

3 Grammatical evolution

GE, initially proposed by Ryan et al. (1998), is a grammar-based GP approach used to evolve programs in arbitrary languages. Each individual in the population is represented by a list of integers called
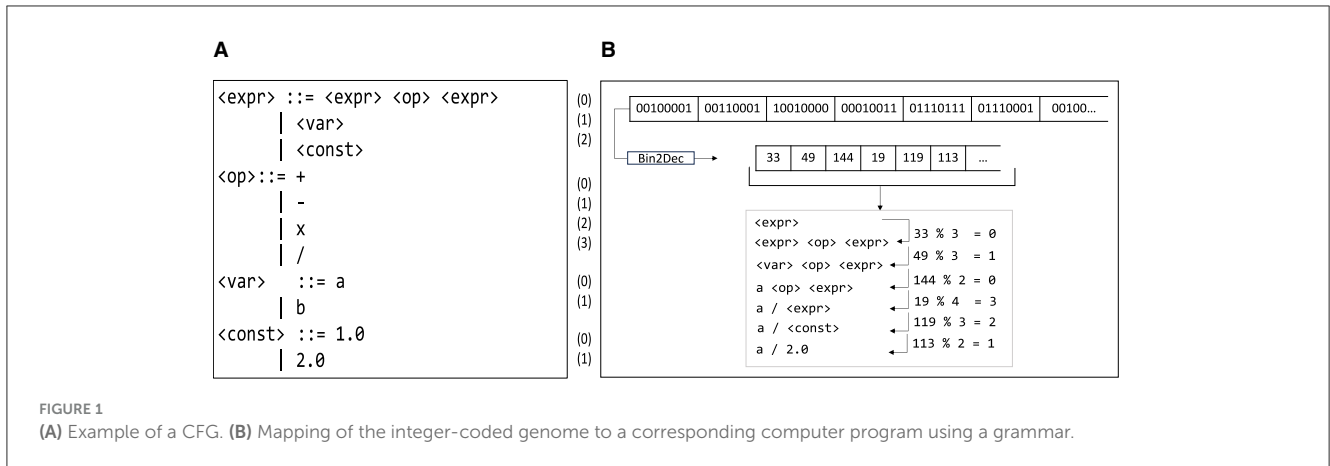
genotype, which is generated from initially random values from the interval [0, 255]. The list element utilizes a *mapping* process to map a genotype to a corresponding phenotype, which is a potential solution to a problem. All evolutionary operators are performed on the genotype.

The mapping is achieved by employing a grammar, typically described in Backus–Naur Form (BNF). This is a Context-Free Grammar (CFG) often used to specify the syntax rules of different languages used in computing (Ryan et al., 2018). The grammar is a tuple $G = (NT, T, S, P)$ where NT represents a non-empty set of *non-terminals (NT)* that expand into non-terminals and/or *terminals* (T), governed by a set of *production (P)* rules. Terminal and non-terminal symbols are the lexical elements used in specifying the production rules constituting a formal grammar. S is an element of NT in which the derivation sequences start, called the axiom. The rule in P is represented as $A ::= \alpha$, where $A \in NT$ and $\alpha \in (NT \sqcup T)$. Figure 1A shows an example of CFG.

GE typically employs an integer-coded genotype. A Genetic Algorithm (GA) performs the evolutionary process by initially creating a population of integer-coded genomes and then, at each iteration (generation), they are evaluated, and various genetic operations are conducted. Evaluation involves mapping them into their respective phenotypes using the GE mapper. The mapping is done starting from the axiom *(S)* of the grammar and expanding the leftmost NT. The codons (which represent integer values and are translated from 8-bit sub-strings, see Figure 1B) are used to choose which rule in P to expand by applying the modulo (mod) operator between the codon and the number of derivation rules in the respective NT.

Figure 1B gives an example of mapping using the example of grammar in Figure 1A. The mapping process starts with: (a) the axiom of the grammar <start>, which expands to three alternatives, and (b) the unused codon of genotype, i.e., 33. Next, a mod operator between codon (33) and the number of production rules (3) provides an index of rules to be expanded. The rule to be expanded at index 0 (33 *mod* 3 = 0) is <expr><op><expr>. This mapping procedure is repeated until all the *NTs* are expanded or there are no more codons left. If the genotype is exhausted, but some *NTs* still remain, a wrapping technique can be employed where codons are reused. The wrapping process stops if a valid individual is achieved or after a certain number of wraps have been performed. In the latter case, the individual is considered invalid if any *NTs* remain unmapped.

As with GP, GE evaluates each mapped program and assigns them a fitness score. Here, fitness is a measurement of the ability of an individual to attain the required objective. Candidate solutions are evaluated by comparing the output against the target function and taking the sum of the absolute errors over a certain number of trials. Programs that contain a single term or those that return an invalid or infinite result are penalized with an error value high enough that they have no opportunity to contribute to the next generation. Genetic selection, crossover, mutation, and replacement are probabilistically applied to the genotypic strings to produce the following generation. This continues for a specified number of generations, budgets, or until the sought solution is achieved. The search space in GE comprises all the possible computer programs that can be produced using a particular

FIGURE 1
(A) Example of a CFG. **(B)** Mapping of the integer-coded genome to a corresponding computer program using a grammar.

grammar. In recent years, GE has been widely used in many domains such as cybersecurity (Ryan et al., 2018), software testing (Anjum and Ryan, 2021), SR problems (Murphy et al., 2021; Youssef et al., 2021), digital circuit design (Gupt et al., 2022a) to name a few.

4 Proposed approach

DBS has two objectives when creating a training set: maintain high coverage while minimizing the amount of training data. The selected test cases are used as training data to evolve the respective model using GE. Since the test cases are spread over a vast space, it is important to categorize them into groups based on certain relationships. For this, we take advantage of clustering algorithms before test case selection is performed.

Based on the nature of data (integer and floating-point number) in SR, we use K-Means clustering with Euclidean distance (d_E) as shown in Equation 1.

$$d_E\left(p,q\right) = \sqrt{\sum_{i=1}^{n}\left(q_i - p_i\right)^2} \tag{1}$$

Here p and q are two test case points in Euclidean n-space. K-Means implicitly relies on pairwise Euclidean distances between data points to determine the proximity between observations. A major drawback in the K-Means algorithm is that if a centroid is initially introduced as being far away, it may very likely end up having no data points associated with it. We take care of this problem by using K-Means++ as the initialization method. This ensures a more intelligent introduction of the centroids and enhances the clustering process. The *elbow method* (a heuristic to determine the number of clusters in a dataset) is used along with the `KneeLocator`[2] function to get an optimal number of clusters. This completely automates the process and produces the optimal number of clusters.

———————

2   The `KneeLocator` function from the `kneed` Python library identifies the point of maximum curvature, known as the "knee" or "elbow," in a dataset, often used to determine the optimal number of clusters.

When preparing data for ML experiment, DBS is applied before building models. We describe the process for the digital circuit domain, although it is broadly the same for SR.

Since the test cases in digital circuit domains are binary (0,1) data and K-Means computes the mean, the standard "mean" operation is useless. Also, after the initial centers are chosen (which depends on the order of the cases), the centers are still binary data. There are chances of getting more frequent ties in the distance calculation, and the test cases will be assigned to a cluster arbitrarily (IBM, 2020). Hence, to group the data, we use Agglomerative Hierarchical Clustering (AHC), a bottom-up approach that is the most common type of hierarchical clustering used to group objects in clusters based on their similarity (Contreras and Murtagh, 2015). As Hamming distance d_H is more appropriate for binary data strings, we use it as a distance measure with AHC. To calculate the Hamming distance between two test cases, we perform their XOR operation, ($p \oplus q$), where p and q are two test case (binary) points of n-bits each, and then count the total number of 1s in the resultant string as shown in Equation 2.

$$d_H(p,q) = \sum_{i=0}^{n} |p_i - q_i| \tag{2}$$

We calculate the distance between clusters using complete-linkage, where the similarity of two clusters is the similarity of their most dissimilar members. This has been assessed (Tamasauskas et al., 2012) as a highly effective hierarchical clustering technique for binary data. In practice, this is comparable to picking the cluster pair whose merge has the smallest diameter. This merge criterion is non-local, and decisions regarding merges can be influenced by the overall clustering structure. This causes sensitivity to outliers, which is important for test case selection and a preference for compact clusters (Manning et al., 2008).

Once we have created the clusters, the test case selection is performed using DBS. The complete process is shown in Figure 2. Note that clustering is a pre-processing step for DBS. The DBS selection approach, given in Algorithm 1, employs a greedy strategy to pick the test case pairs most dissimilar to all others in the given cluster. As a result, test cases are chosen to maintain diversity (to represent a large test case space) and maximize coverage. A diversity analysis of the data selected using the DBS algorithm is presented in Ryan et al. (2021).
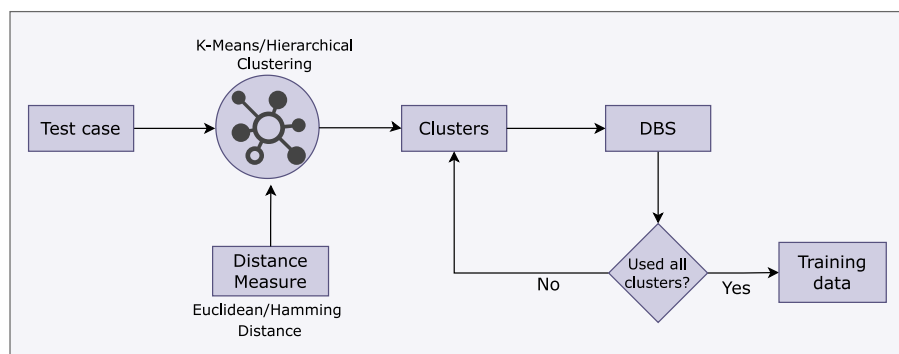
FIGURE 2
The selection process involving Machine Learning and DBS algorithm.

During test case selection, a distance matrix is first created by executing a proximity calculation on each test case of a given cluster. The choice of distance measurements is crucial, as was mentioned during the clustering procedure. We use Euclidean distance for SR data and Hamming distance for digital circuit data because they are more appropriate for their respective datasets. The test case selection is performed based on the maximum distance between input test case pairs. It is worth mentioning that the entire clustering and test case selection process is applied to the input test cases. It is possible that two or more distinct input test cases might pose the same distance from a given input test case, and hence, duplicate data selections are avoided. The size of the test case subset to be chosen or the stopping criterion relies on the user-defined budget or B.

Once the required test suite has been obtained, we use it as the training set. The approach is effective and the same for SR and digital circuits but necessitates choosing the distance measure based on the data type. Sorting the distances is essential while choosing the test cases. This facilitates a simple test case selection based on the ascending/descending order of test case diversity. In our experimental setup, we select test cases in descending order of diversity. The value of the count may need to be rounded off depending on how many test cases are in a cluster. We round it up using the `ceil` function.

5 Experimental setup

In this section, the benchmarks selected to evaluate the DBS method are discussed. Moreover, the methodology used for SR and digital circuit problems is discussed in detail, along with the GE setup and parameters.

5.1 Benchmarks

We selected benchmark suites with a high level of variance in terms of problem complexity and test case volume. These benchmarks are popular in their respective problem domains, i.e., SR and digital circuit.

Require: C clusters of test cases
Require: Selection budget B%
Ensure: Training data
1: *trainingData* ← []   ▷ Initialize an empty list
2: **for** each cluster C_i in C **do**
3:    *inputTestCases* ← Extract input values from test cases in C_i
4:    N ← Count of test cases in C_i
5:    Initialize a distance matrix D with dimensions $N \times N$
6:    **for** each pair of *inputTestCases* (T_j, T_k) in C_i **do**
7:       D ← distance between T_j and T_k ▷ use d_E or d_H
8:       Store the distance in $D[j][k]$
9:    **end for**
10:   Remove the upper half of the distance matrix ▷ $D[j][k]$ where $k \geq j$
11:   *sortedDistances* ← Sort distances in D in decreasing order
12:   $cnt \leftarrow \frac{B \cdot N}{100}$ ▷ Calculate the number of cases to select as B% of N
13:   *desireIndices* ← Indices of the top cnt test cases corresponding to *sortedDistances*
14:   **for** *index* in *desireIndices* **do**
15:       *trainingData* ← Append the corresponding test case from C_i
16:   **end for**
17: **end for**
18: **Return** *trainingData* (all selected test cases from all clusters)

Algorithm 1. Distance-based Selection (DBS).

5.1.1 SR benchmarks

The SR benchmarks are selected from both synthetic and real-world problems. Recently, a subset of these benchmarks was also used to test different tools and algorithms, based on GE, such as GELAB (Gupt et al., 2022b), AutoGE (Ali et al., 2021a), and GRAPE (de Lima et al., 2022).

We selected 10 synthetic SR datasets from McDermott et al. (2012). The problems are diverse, having one to five input features and sample sizes varying from 402 to 20,000. These benchmarks are listed in Supplementary Table S1. More details, such as mathematical expressions and training and test range, can be found in McDermott et al. (2012).

We also selected 10 real-world SR problems that have been widely used in many research (Oliveira et al., 2018; Ali et al., 2021b). These are listed in Supplementary Table S2. The dataset for Dow Chemical was obtained from the GP Benchmarks.[3] The other datasets were obtained from the UCI Machine Learning Repository[4] and StatLib-Datasets Archive.[5] The benchmarks are diverse in terms of problem difficulty, having 5 to 127 input features with varying numbers of instances from 506 to nearly 4,898. Each dataset is referred to with a short name for ease and will be used in the rest of the paper.

5.1.2 Digital circuit benchmarks

A brief description of the selected circuit problems is described below and is listed in Supplementary Table S3.

A binary magnitude comparator is a combinational logic circuit that takes two numbers and determines if one is greater than, equal to, or less than the other. These circuits are often used in Central Processing Units (CPU), Microcontroller Units (MCU), servo-motor control, etc. In this paper, a 5-bit comparator is used. A parity generator is a combinational logic circuit that generates the parity bit in a transmitter. A data bit's and a parity bit's sum can be even or odd. In the case of even parity, the additional parity bit will make the total number of 1s even. In contrast, in the case of odd parity, the additional parity bit will make the total number of 1s odd. A multiplexer, or MUX, is a logic circuit that acts as a switcher for a maximum of 2^s inputs and s selection lines and synthesizes to a single common output line in a recoverable manner for each input signal. They are used in communication systems to increase transmission efficiency. In this paper, an 11-bit multiplexer is used. An Arithmetic Logic Unit (ALU) is a combinational circuit that performs arithmetic and logic operations according to the control bits. It is a fundamental building block of the CPU. In this paper, we have used a 5-bit operand size and 2-bit control signal, and thus, it can target four operations (arithmetic and logic): addition, subtraction, AND, and OR.

5.2 Methodology

The experimental setup for both problem domains is relatively similar, except for their implementation of the fitness evaluator, and hence, both problems incorporate two separate fitness functions.

We performed 5,040 experiments in total, corresponding to 30 independent runs for each of the 24 problems using each training dataset (described next). Based on a small set of exploratory runs,

the evolutionary parameters used for all the experiments are given in Supplementary Table S4.

We evaluate the DBS algorithm for both domains at a range of test case selection budgets B. Using the baseline training data (see Supplementary Table S5 for details on this, created for each domain), we apply the DBS algorithm for test case selection using different budgets from 70 to 45% at intervals of 5%. The budget starts at 70% of the baseline training dataset as we tried to have a significant reduction. We considered six different budgets to select training data using DBS, i.e., 70%, 65%, 60%, 55%, 50%, 45%.

5.2.1 SR methodology

In the case of experiments involving SR datasets, we used the conventional train-test split approach; in our case, 70%–30% train-test split on the datasets. We use RMSE as a fitness function in this experiment. This is one of the most commonly used measures for evaluating the quality of predictions and is represented in Equation (3).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{\eta_o} (P_i - O_i)^2}{\eta_o}} \qquad (3)$$

Here, P_i and O_i are the predicted and observed (actual) values for the ith observation, respectively, while η_o is the total number of observations. Across all experiments, we allowed the model to train for 50 generations.

The best-performing individuals using 70% training data from each generation are tested against the 30% testing data, and their RMSE scores are recorded. We consider this as a baseline in this experiment.

A list of all the problems and their respective training and testing data sizes are given in Supplementary Table S5. These new training datasets selected using the DBS method are used to train the SR models. The best-performing individuals from each generation are tested against the testing data, and their RMSE scores are recorded. All experiments on a particular benchmark use the same testing data.

For ease of readability, 70% of the total data used to train is referred to as *baseline* in Supplementary Table S5 and the remaining 30% as *testing data*. The data selected from baseline training data using DBS at budget B is represented as DBS (B%) training data.

5.2.2 Digital circuit methodology

In the case of digital circuits, we use a slightly different approach. Unlike calculating predictions or RMSE as fitness in the case of SR, here we calculate the number of successful events or test cases (where expected output and observed output are the same) of the evolved circuit. The objective is to achieve a design that should pass all (or maximum) instances in training data. We call this fitness function "hit-count," as it counts the number of successful hits as

$$hit\text{-}count = \sum_{i=1}^{\eta_o} (P_i = O_i \rightarrow 1), \qquad (4)$$

where P_i and O_i are the predicted and observed outputs for ith observation, while η_o is the total number of observations

or instances in training data. Considering the above-mentioned objective and problem specification, the total data available for a benchmark is used as training data and exhaustive testing is performed using the same dataset. The best test scores are considered as baseline.

The DBS approach uses the same strategy of selecting a subset of baseline training data as is used in SR. The testing data is kept the same for all experiments on a benchmark. A brief detail of training and testing data size is given in Supplementary Table S5. The circuit's evolutionary process and architecture are shown in Figure 3. A major difference in the implementation of SR and digital circuit setup is the fitness evaluation module (shown in a separate block). In essence, this is the part where GE implementation differs for both domains. Thus, the grammar, test cases, and fitness evaluation modules are problem-specific. For example, the solutions in SR are mathematical models. These are evaluated using testing data, and RMSE score is calculated without any external tool, unlike in a digital circuit experiment where a HDL simulator is required.

Since the solutions in the latter case are circuits made up of VHDL code, they need a simulator to be compiled, executed, and evaluated. In this experiment, we use GHDL,[6] an open-source simulator for VHDL designs. In addition to the GHDL simulator, the evolved solutions (circuits) require a testbench to simulate and subsequently assess the design. This extra step extends the time required for the fitness evaluation process. In this case, a substantial influence on the fitness evaluation process is caused by the speed of the circuit simulator tool and the clock rate offered to evaluate training data instances in a synchronous order. These additional steps of HDL simulation make circuit evaluation much slower than software evaluation (Hsiung et al., 2018).

The GE circuit training procedure, as shown in Figure 3, involves the usage of training data. We employ yet another testbench to test the evolved circuit against testing data. In all cases, we save the best individuals across each generation and test them after a run is completed. As a result, there is a less frequent requirement to initialize the testing module.

The design and testing process of benchmarks from both problem domains is similar at an abstract level. As outlined in Section 3, GE individuals are mapped from binary strings to executable structures using grammars.

In this experiment, we used a machine with an Intel i5 CPU @ 1.6 GHz with 4 cores, 6 MB cache, and 8 GB of RAM. The test case selection algorithm is coded in Python. The GE experiments are run using libGE. LibGE is a C++ library for GE that is implemented and maintained by the Bio-computing and Developmental Systems (BDS) research group at the University of Limerick.

6 Results and discussion

The results from the benchmarks, as shown in Supplementary Tables S1, S2 (SR) and Supplementary Table S3 (digital circuits) is analyzed in this section. We also perform analysis on GE run time and solution size.

6.1 Test scores for SR

Figure 4 shows the testing score for the synthetic SR benchmarks. The test score of the best individuals trained using different subsets of DBS training data against the baseline is shown. In each of the `Keijzer-9`, `Keijzer-10`, `Keijzer-14`, and `Nguyen-9` problems, most DBS-trained solutions perform better than the baseline. In contrast, in `Nguyen-10`, `Keijzer-5`, `Korns-11`, and `Korns-12` problems, the results are competitive, and there is no significant difference in performance between any of the DBS-trained solutions and the baseline. In only one case, Figure 4A (`Keijzer-4`), did the baseline generally outperform the DBS. In the `Vladislavleva-5` problem, shown in Figure 4H, the baseline outperformed for all sizes except the budget DBS (65%). In all cases, the DBS-trained sets were significantly faster to train. See Section 6.3 for more details.

The test scores of the real-world SR benchmarks are shown in Figure 5. We employed the same strategy used in the previous experiment. The testing scores of the best-trained individuals are reported as the baseline. The training data used in the baseline is used for subset selection using the DBS algorithm. We performed test case selection using the same range of *B*. The test scores of the best individuals trained using training data obtained by the DBS algorithm were tested, and the results were compared. The box plots in Figure 5 show the min, max, mean, and median values of the obtained scores.

In each of the `airfoil`, `concrete`, `whitewine`, `housing`, and `pollution` problems, the test score of the best individuals trained using DBS appear to be statistically similar or better than baseline.
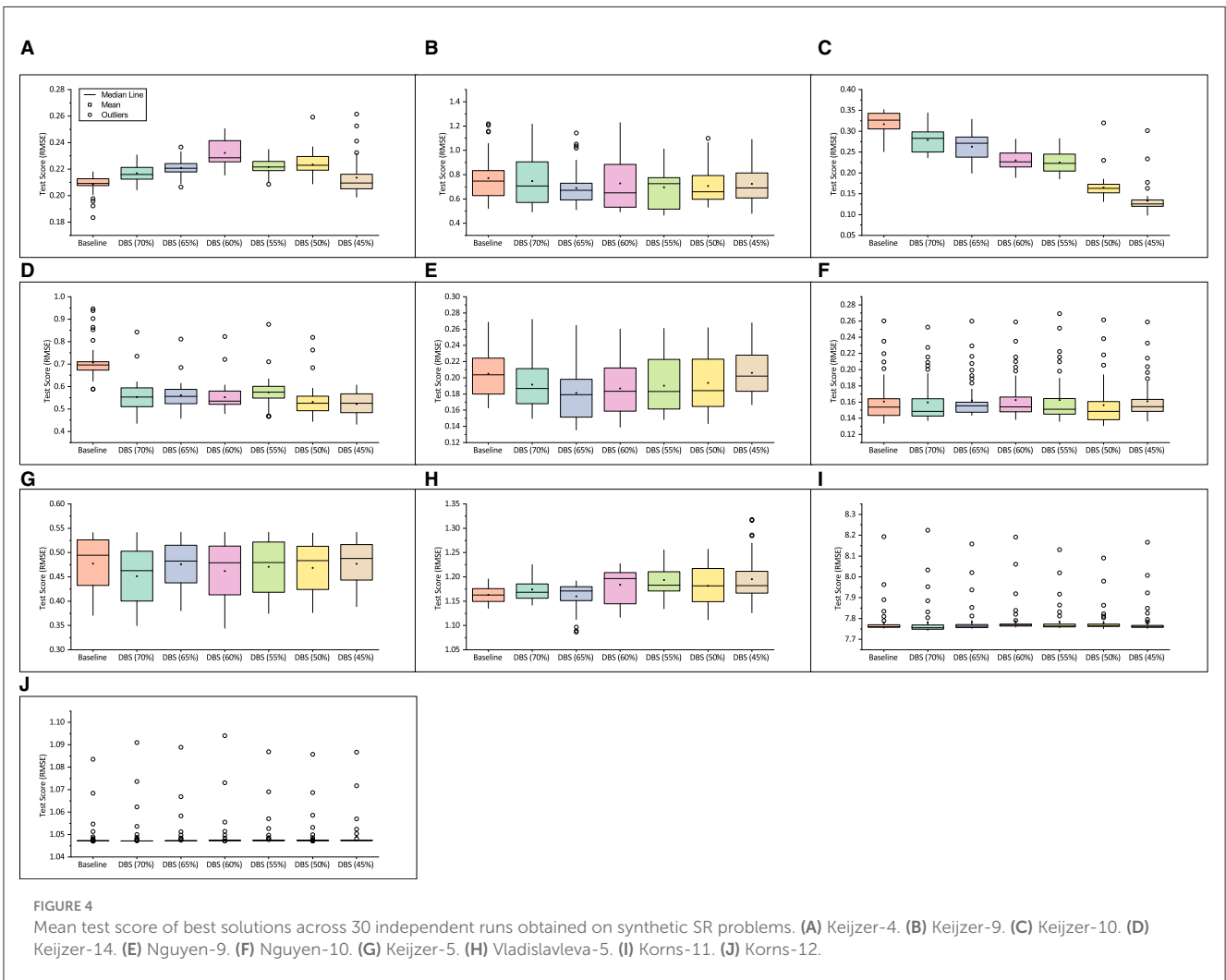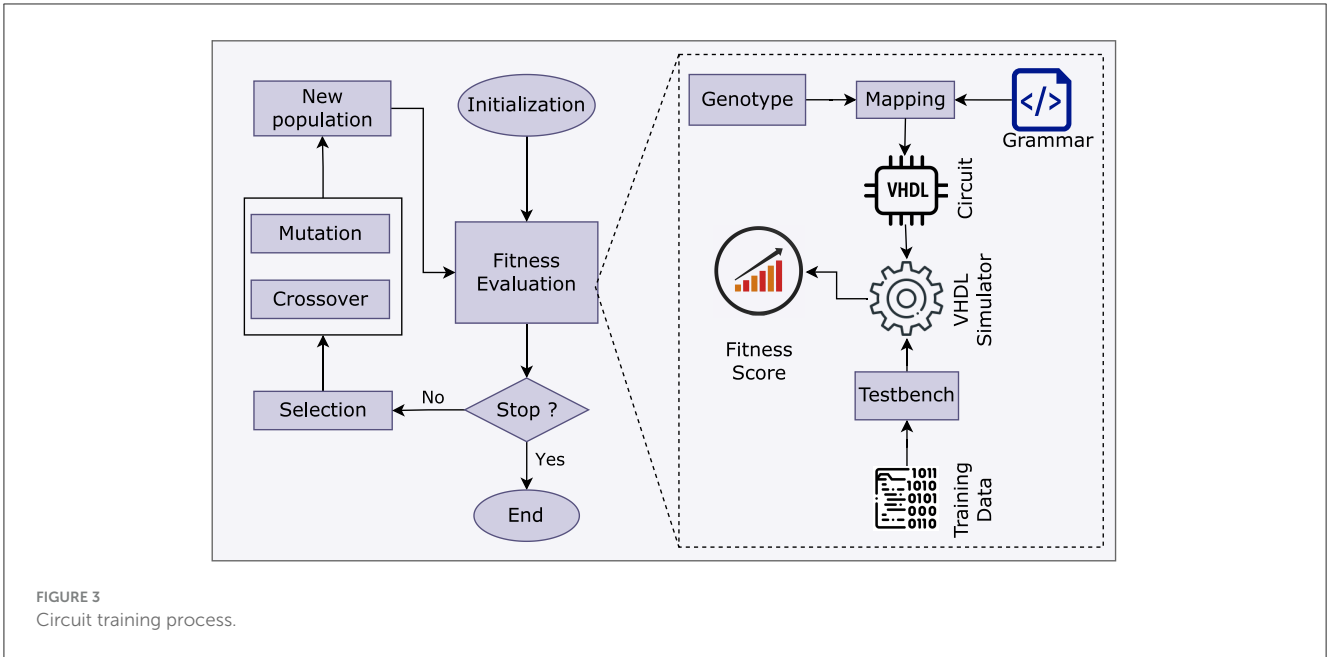
In the `cooling` and `crime` problems, the DBS-trained solutions outperformed the baseline. In the case of `heating`, the DBS-trained solutions generally outperform the baseline for all sizes except at DBS (50%). The solutions trained on DBS data demonstrate superior performance over the baseline for `redwine`, except when *B* was set at 50 and 45%.

For `dowchem` problem, the solutions trained on DBS data exhibit higher test scores than the baseline, except when *B* was set at 70%.

The application of DBS extends beyond a single EA as its primary function is to ensure data diversity to guide the evolutionary process. We also evaluated DBS using the same budgets on two SR datasets using Koza-style tree-based GP (see Supplementary Figure S1 and Supplementary Table S6). We used RMSE as the fitness function. The results from Supplementary Table S6 demonstrate that models trained using GP on different DBS budgets perform similar or better than the baseline. This shows that DBS is easily extensible to work with different EAs.

6.2 Test scores for circuit design

Figure 6 displays the outcome for digital circuit benchmarks. It should be noted that, unlike SR, where the RMSE score is calculated, the test score is determined by the number of instances passed out of the total available in the testing data. As we compare

---

FIGURE 3
Circuit training process.

FIGURE 4
Mean test score of best solutions across 30 independent runs obtained on synthetic SR problems. **(A)** Keijzer-4. **(B)** Keijzer-9. **(C)** Keijzer-10. **(D)** Keijzer-14. **(E)** Nguyen-9. **(F)** Nguyen-10. **(G)** Keijzer-5. **(H)** Vladislavleva-5. **(I)** Korns-11. **(J)** Korns-12.

FIGURE 5

Mean test score of best solutions across 30 independent runs obtained on real-world SR problems. **(A)** Airfoil. **(B)** Heating. **(C)** Cooling. **(D)** Concrete. **(E)** Redwine. **(F)** Whitewine. **(G)** Housing. **(H)** Pollution. **(I)** Dowchem. **(J)** Crime.

the test scores, we plot a generation versus test score graph of the best individuals obtained. This facilitates the identification of the subset of training data that can guide a better search. However, we do present the mean of best test score, as we did in the SR experiment during the statistical test.

Figure 6A represents the test score of the best solutions obtained across 50 generations for the `comparator`. The mean best test scores of the best individuals across 30 independent runs were observed to be the same. The problem seems trivial, and all experiments on the `comparator` performed similarly well.
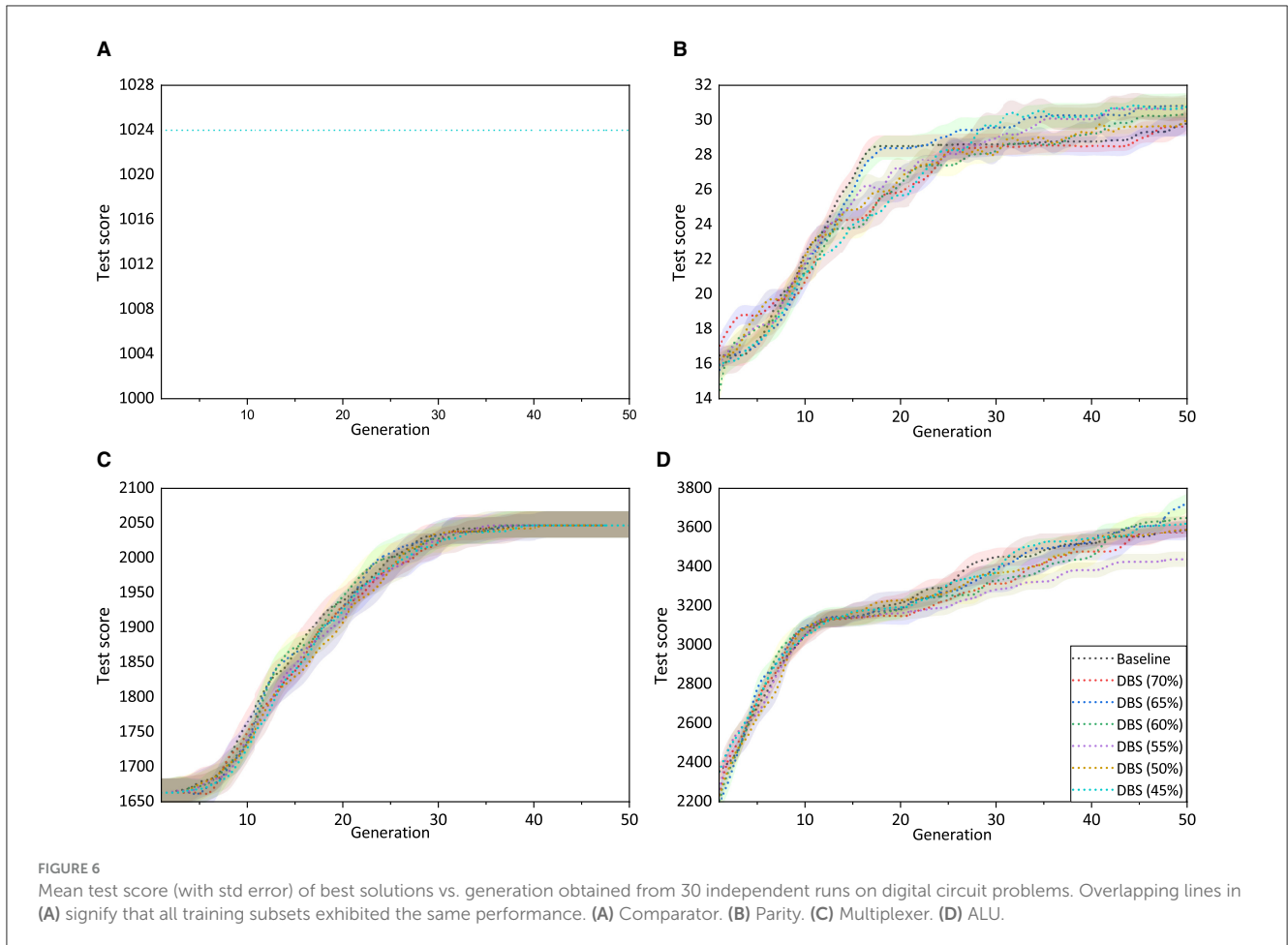
The test scores of the solutions obtained using the `parity` dataset are presented in Figure 6B. The best individuals trained using DBS (70%) could pass as much of the testing data as were passed by the best solution obtained on baseline training data. In the remaining cases, the DBS-trained data produced solutions with higher test coverage than those obtained in the baseline experiment.

In the case of the `multiplexer`, the best solutions obtained in the 50th generation were able to maintain a similar test score, as shown in Figure 6C. However, close observation of Figure 6C indicates that experiments using DBS training data (in some cases) could achieve better test scores in earlier generations than the baseline.

Figure 6D shows the mean best test score of solutions obtained on `ALU` dataset. The experiments using DBS training data in most of the cases, except DBS (70%) and DBS (55%), were able to get solutions that could achieve better test scores than the solutions obtained in the baseline.

The test results for 24 benchmarks are shown in Figures 4–6. We examined the results regarding the best-trained individuals' test scores and showed how they compared to the outcomes of their respective baseline experiments. The test scores presented using box plots and line graphs are good for insight into the performance of the different sets of training data used and are reliable to a certain extent. However, we need some strong mathematical evidence before coming to a conclusion. For this, we perform statistical tests to be able to fairly compare the different methods and validate the results.

We assessed the normality of the data using the Shapiro-Wilk Test at a significance level of $\alpha = 0.05$. The results indicate that the data samples do not meet the assumptions necessary for parametric tests, as they do not exhibit a normal distribution. In such cases, non-parametric tests are more appropriate. Non-parametric tests make fewer assumptions about the data distribution and are, therefore, more robust to violations of normality assumptions.

FIGURE 6
Mean test score (with std error) of best solutions vs. generation obtained from 30 independent runs on digital circuit problems. Overlapping lines in **(A)** signify that all training subsets exhibited the same performance. **(A)** Comparator. **(B)** Parity. **(C)** Multiplexer. **(D)** ALU.

As the results obtained from distinct training datasets are mutually independent, the Wilcoxon rank-sum[7] (a non-parametric test) is thus employed to determine whether there is any statistically significant difference between the test scores of solutions obtained. We use a significance level of 5% (i.e., 5.0×10^{-2}) for the two hypotheses.

To test for a statistical difference between the two samples, we compare the DBS approaches with the baseline using the null hypothesis (H0) that "there is no difference between the test scores of the baseline and DBS approaches." If the statistical analysis does not reveal a significant difference, we label the outcome as equal or similar, denoted by the symbol "=" to indicate no substantial disparity between the performance of the baseline and DBS approaches. Otherwise, we perform another test, and the following hypotheses are tested:

- Null Hypothesis (H0): The models trained using the baseline training data have a better test score than the models trained using the DBS-selected training data at a budget *B*;
- Alternative Hypothesis (H1): The test score of models trained using DBS training data is better than the test score of models trained using baseline training data.

––––––––––

7   The Wilcoxon rank-sum test is a non-parametric test that does not rely on the assumption of normality. Shapiro-Wilk Test was performed to check if a more appropriate test could be applied.

After the statistical analysis, the results are marked with a "+" symbol to indicate significantly better performance or a "−" symbol to indicate significantly worse performance. This hypothesis testing framework allows us to determine whether the DBS-selected training data leads to improved model performance compared to the baseline training data under the specified budget constraint.

Table 1 shows the mean best test scores of the best individuals across 50 generations obtained from 30 independent runs. Note that the mean test scores are rounded to four decimal points. This can make, sometimes, the test scores at two or more DBS budgets appear to be the same, but in reality, it is extremely uncommon to obtain the same test score after carrying out a large number of runs, in our case, 30 runs. All DBS test results were statistically tested against the corresponding baseline. The p-values are calculated, and accordingly, the output is reported in Table 1.

Table 2 summarizes the outcomes of the statistical analysis conducted on real-world SR benchmarks. Instead of solely relying on the box plots, this strengthens the results and aids in deciding whether there is sufficient evidence to "reject" a hypothesis or to "support" the proposed approach.

We discover that the DBS algorithm performs better in contrast to how well it performed with synthetic SR problems because real-world SR problems present a higher number of features and level of complexity than synthetic SR benchmarks. While examining the findings for each benchmark, DBS surpassed the baseline at one or more test case selection budgets while

TABLE 1 Results of the Wilcoxon test for SR *synthetic benchmarks* considering a significance level of $\alpha = 0.05$.

Benchmarks	Baseline	DBS (70%)	DBS (65%)	DBS (60%)	DBS (55%)	DBS (50%)	DBS (45%)
Keijzer-4	0.2088	0.217−	0.2208−	0.2323−	0.2217−	0.2234−	0.2135=
Keijzer-9	0.7724	0.7462+	0.6893+	0.7278+	0.6955+	0.7059+	0.7238+
Keijzer-10	0.317	0.2788+	0.2628+	0.2301+	0.225+	0.1651+	0.133+
Keijzer-14	0.7064	0.5533+	0.5603+	0.552+	0.5742+	0.5308+	0.522+
Nguyen-9	0.205	0.1915+	0.1812+	0.1868+	0.1902+	0.1937+	0.206=
Nguyen-10	0.1608	0.1595+	0.162=	0.1625−	0.1625−	0.1562+	0.1609=
Keijzer-5	0.4774	0.4515+	0.4761=	0.4617+	0.4705+	0.4686+	0.4767=
Vladislavleva-5	1.1629	1.1739−	1.1599=	1.1837−	1.1933−	1.1816−	1.1952−
Korns-11	7.7804	7.7806+	7.783-	7.7877−	7.7846−	7.7802−	7.7821=
Korns-12	1.0486	1.0491=	1.0488-	1.049−	1.049−	1.049-	1.049−
Total +/=/-	−	7/1/2	4/3/3	5/0/5	5/0/5	6/0/4	3/5/2

Values shown are the mean best test scores across 30 independent runs. The symbols +, =, − indicate whether the corresponding results for DBS are significantly better, not significantly different, or worse than baseline, respectively. The last row summarizes this information.

TABLE 2 Results of the Wilcoxon test for SR *real-world benchmarks* considering a significance level of $\alpha = 0.05$.

Benchmarks	Baseline	DBS (70%)	DBS (65%)	DBS (60%)	DBS (55%)	DBS (50%)	DBS (45%)
Airfoil	16.6198	16.5061=	15.0541+	15.051+	14.8581+	15.7739+	15.7069+
Heating	7.929	7.3309+	7.303+	7.2609+	7.5126+	8.5253-	7.7708+
Cooling	7.2973	6.8732+	6.787+	6.7121+	6.6915+	6.8563+	6.9468+
Concrete	14.239	14.233=	13.769+	13.954+	14.107+	13.586+	13.497+
Redwine	0.769	0.7677=	0.7647+	0.7669+	0.7651+	0.7698−	0.7708−
Whitlewine	0.8246	0.8137+	0.8133+	0.8162+	0.8181=	0.8126+	0.8164=
Housing	11.1763	10.8883+	10.9344+	11.1181+	11.0832=	11.0703=	10.9704+
Pollution	102.291	102.0317=	95.3476+	95.2581=	99.7087=	98.7799=	98.1966=
Dowchem	0.3392	0.3414−	0.3398=	0.3396=	0.3401−	0.3386+	0.338+
Crime	0.1543	0.1522+	0.1519+	0.153+	0.1536+	0.1537+	0.1522+
Total +/=/-	−	5/4/1	9/1/0	8/2/0	6/3/1	6/2/2	7/2/1

Values shown are the mean best test scores across 30 independent runs. The symbols +, =, − indicate whether the corresponding results for DBS are significantly better, not significantly different, or worse than the baseline. The last row summarizes this information.

TABLE 3 Results of the Wilcoxon test for *digital circuit* benchmarks considering a significance level of $\alpha = 0.05$.
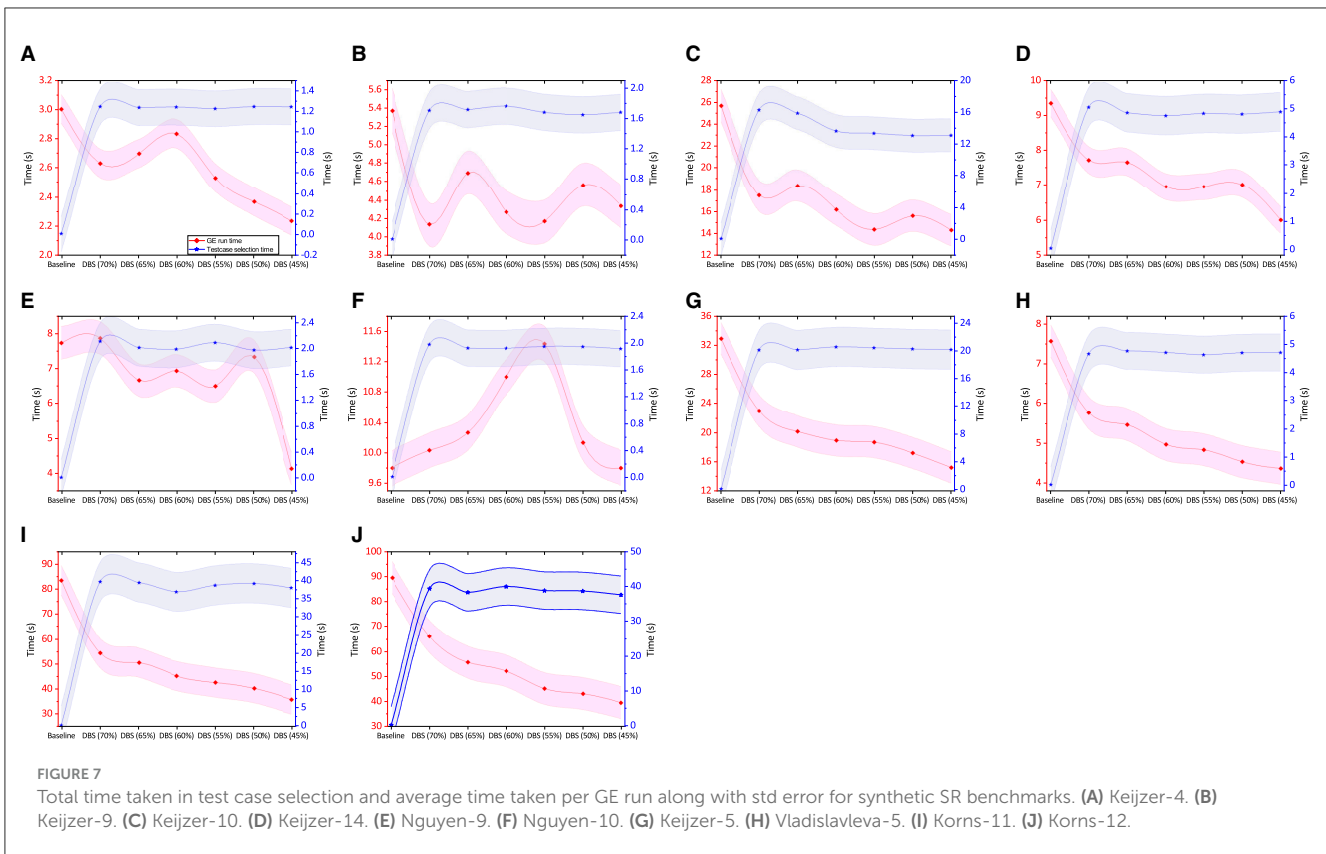
Benchmarks	Baseline	DBS (70%)	DBS (65%)	DBS (60%)	DBS (55%)	DBS (50%)	DBS (45%)
Comparator	1,024	1,024=	1,024=	1,024=	1,024=	1,024=	1,024=
Parity	26.334	25.78067−	26.756+	25.816−	26.46667=	26.07267=	26.18533=
MUX	1,930.923	1,922.645−	1,927.595=	1,928.789=	1,922.048−	1,919.659-	1,919.829−
ALU	3,250.112	3,212.179−	3,255.58=	3,226.407-	3,166.453−	3,229.893−	3,260.591+
Total +/=/-	−	0/1/3	1/3/0	0/2/2	0/2/2	0/2/2	1/2/1

Values shown are the mean best test scores across 30 independent runs. The symbols +, =, − indicate whether the corresponding results for DBS are significantly better, not significantly different, or worse than the baseline. The last row summarizes this information.

achieving competitive performance at other budgets. The best test results achieved by DBS on SR datasets are compared against RegCNN and RegENN, two instance selection approaches for regression problems (Kordos and Blachnik, 2012). The comparative analysis is discussed in Supplementary Figures S2, S3 and Supplementary Tables S7, S8.

Table 3 displays the findings of the Wilcoxon rank-sum test conducted on the test score for digital circuits on different DBS training budgets against the baseline. We followed the same strategy for the statistical test as used in the case of SR.

ALU has the maximum number of operations, and we observe that on smaller budgets of DBS (45%), it outperforms the baseline

FIGURE 7
Total time taken in test case selection and average time taken per GE run along with std error for synthetic SR benchmarks. **(A)** Keijzer-4. **(B)** Keijzer-9. **(C)** Keijzer-10. **(D)** Keijzer-14. **(E)** Nguyen-9. **(F)** Nguyen-10. **(G)** Keijzer-5. **(H)** Vladislavleva-5. **(I)** Korns-11. **(J)** Korns-12.

while being almost competitive on other budgets. Smaller budgets tend to have strictly more diverse test cases. This not only demonstrates the efficacy of DBS on digital circuits but also shows a dramatic reduction of more than 50% of test cases to train the models.

Furthermore, we see a similar pattern repeated on the remaining three problems, which indicates better testing scores due to high coverage resulting from diversity in training cases.

Moreover, the baseline training was conducted exhaustively by using all test cases, which is not feasible in real-world complex circuits that need to satisfy multiple constraints simultaneously.

It is evident from Tables 1–3 that the proposed DBS algorithm helps select a subset of test cases to train models that perform at least as well as and sometimes significantly better, in terms of achieving better test score, than the considered baseline.

Although B depends on several factors, including the difficulty of the problem, resource constraints, and the type of data being utilized, the results in Table 2 show that DBS (65%) and DBS (60%) consistently outperformed or matched the baseline across all problems. Likewise, in Table 3, DBS (65%) never underperformed the baseline on any of the benchmarks. This can be a promising starting point when applying DBS. Nevertheless, users can define the budget based on their requirements and available resources.
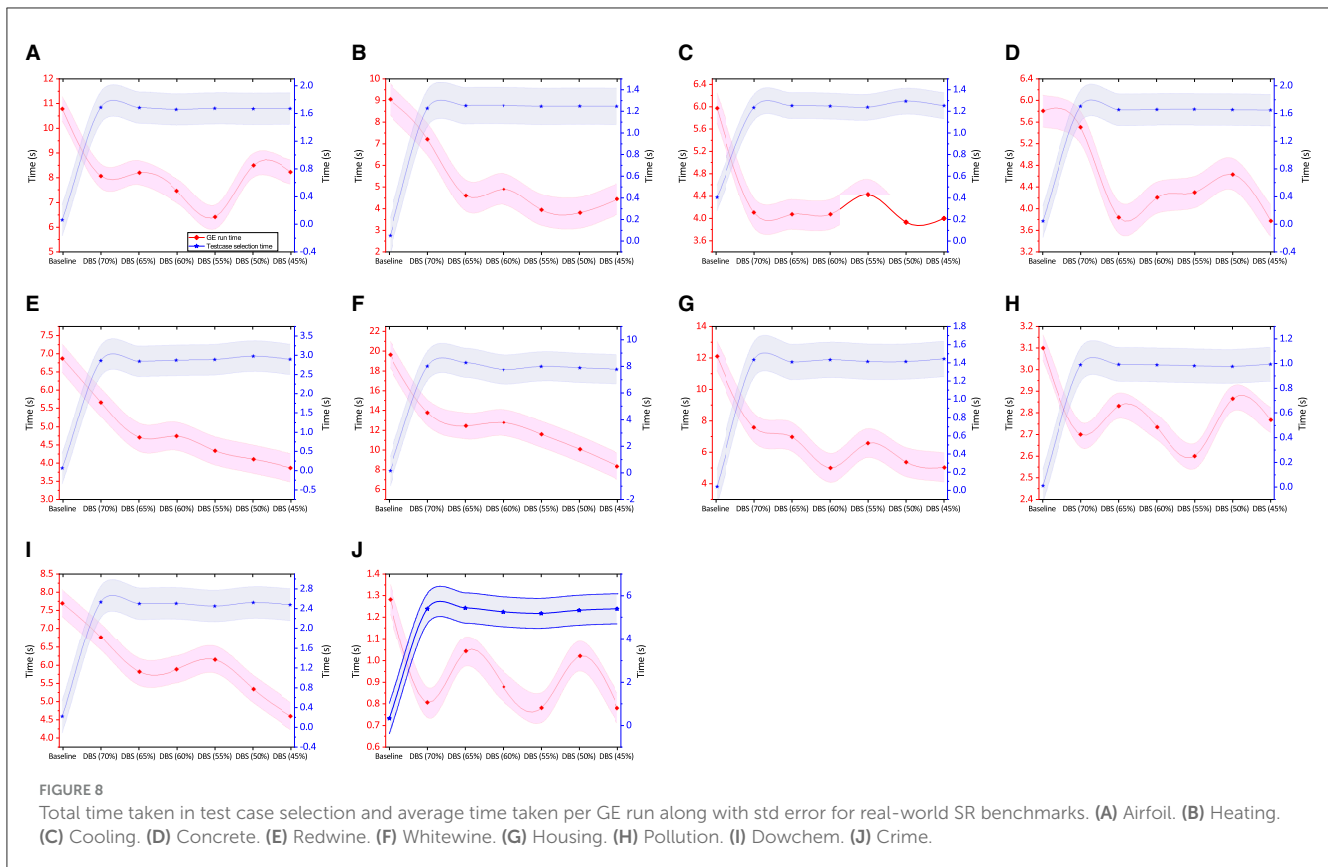
6.3 Time analysis

It is crucial to measure the time efficiency and, hence, the computational cost of the method. To compare the GE run time,

we performed a time analysis for all the problems investigated in this paper; where appropriate, we also present the time taken by the DBS algorithm in selecting the subset of test cases.

The graphs in Figure 7 give the results of the time analysis for synthetic SR benchmarks. In all 10 of those benchmarks, the maximum test selection time by DBS is at the budget DBS (70%) and lower. A decrease in B decreases the test case selection time. Of course, the time taken in preparing baseline data is lower (see the blue line in Figure 7) as its simple task was to split the dataset into the ratio of 70%-30%.

To measure the time cost for each GE run, we calculate the total training time taken across 30 independent runs and take an average of that. A time analysis graph for `Keijzer-4` is shown in Figure 7A. The shaded region in the line graph shows standard error. Using DBS training data, the model training took less time than the baseline experiment did. However, the expected time consumption using DBS (60%) should have been lower than that at DBS (70%). If we look closely at the numerical data, the time difference between the two DBS instances is in milliseconds, which is too small to track. The alternate strategy might involve performing tens of thousands of runs to obtain a better value; however, this is not practical owing to time restrictions. The depiction of the `Keijzer-9` instance in Figure 7B illustrates a similar finding to `Keijzer-4`. The training time for DBS with a lesser budget is likewise anticipated to be shorter than one with a higher budget. The time difference is too small, and that is because the overall run time is not long enough to distinguish the difference. In the case of `Keijzer-10`, we see a distinct pattern, and a possible reason is that the benchmark poses a higher number of
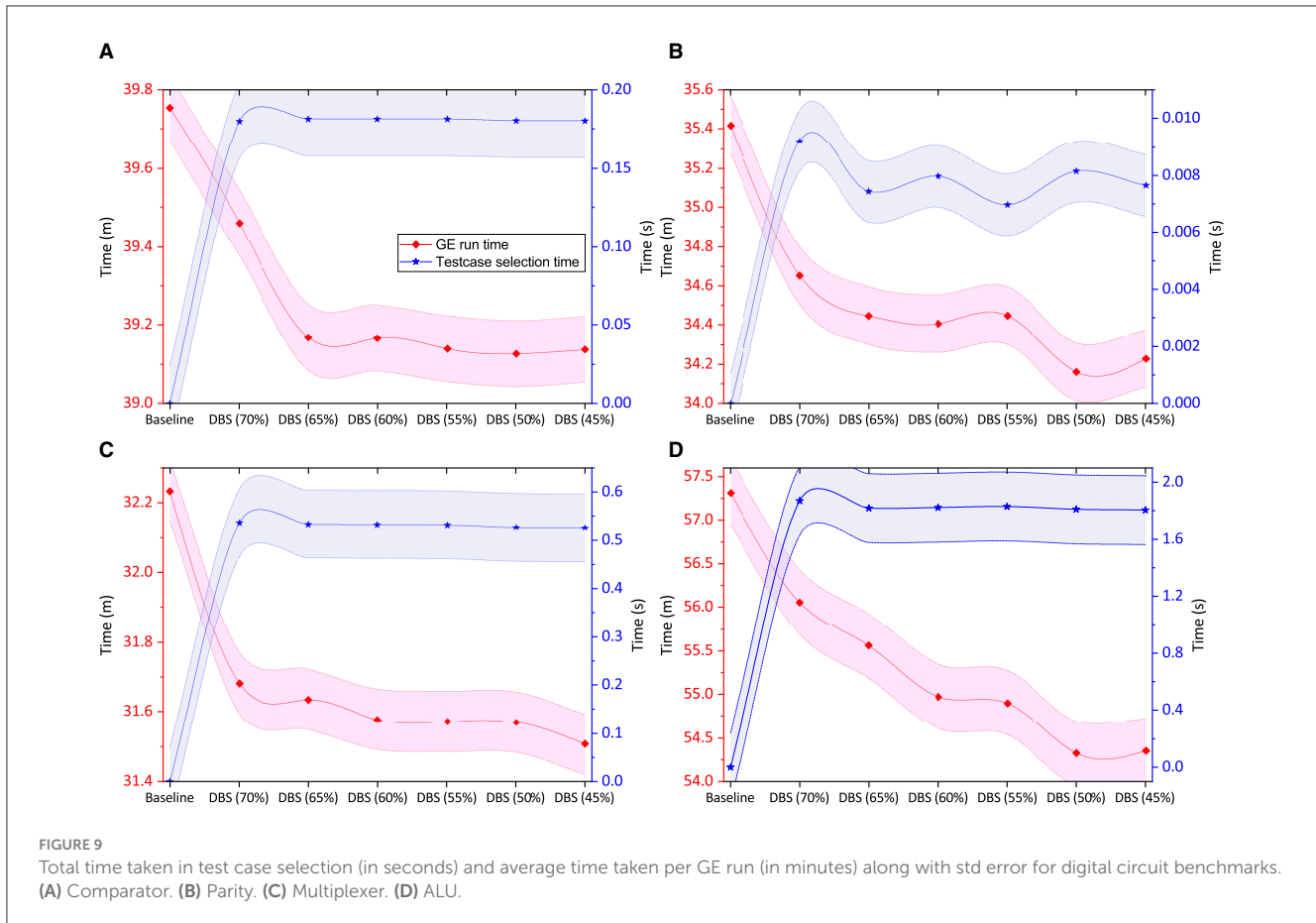
FIGURE 8
Total time taken in test case selection and average time taken per GE run along with std error for real-world SR benchmarks. **(A)** Airfoil. **(B)** Heating. **(C)** Cooling. **(D)** Concrete. **(E)** Redwine. **(F)** Whitewine. **(G)** Housing. **(H)** Pollution. **(I)** Dowchem. **(J)** Crime.

instances that increase the total run time, as shown in Figure 7C. A similar trend is observed for `Keijzer-14` in Figure 7D. In the case of `Nguyen-9` in Figure 7E, the training duration in baseline and DBS (70%) is comparable; however, in other cases, we see a pattern similar to `Keijzer-9` (Figure 7B). One potential explanation is the total number of instances in both problems is ≈ 1k. However, in most instances, the DBS training period is shorter than the baseline training time. For `Nguyen-10`, Figure 6 shows a completely different pattern. Even though DBS (55%) has fewer instances than the baseline, the training period is longer. In order to determine the potential cause of such instances where smaller training data takes longer to process than bigger training data, we made the decision to examine the effective solution size, which is further covered in Section 6.4. The training times for `Keijzer-5`, `Vladislavleva-5`, `Korns-11`, and `Korns-12` problems are evident and as expected. This is primarily due to the longer training period and/or increased problem complexity. Briefly, the baseline run time for all the algorithms is significantly higher, as observed in Figure 7. For example, in the case of `Keijzer-10`, the worst-case scenario of DBS (70%) is 9 s lower than the baseline.

The GE run time or training time analysis of real-world SR experiments is shown in Figure 8. In comparison to synthetic SR datasets, these benchmarks have more instances. The complexity of the problem has increased as well. The training time in most cases is typically predictable (in terms of graph trend) and, in some ways, evident. The blue line displays the test case selection time for the baseline and DBS approaches. The baseline strategy typically takes less time to choose test cases than the DBS approach as it does not

involve any complex steps as in DBS. The DBS took nearly the same or less time on most test case selection budgets.

Figure 8A depicts the amount of training time required for the `airfoil` experiment. The DBS training set significantly reduces the GE run time compared to the baseline training period. However, the time at DBS (50%) is longer than at DBS (55%). We hypothesize that this may be because the experiment employing DBS (50%) produced greater individual (solution) sizes. We want to conduct an analysis of solution size to study it further, even if our main objective is to compare the time efficiency of DBS with the baseline, which is solved in the majority of cases. The time analysis in Figure 8B for the `heating` dataset is as expected. Although using DBS training data at some budgets appears to take a bit longer than their neighboring higher budget, the time difference is relatively small, which is another instance where having thousands of runs is necessary to identify a clear trend. Figures 8C, D show a pattern that is similar. Given that these problems have an equal amount of features and instances, it is conceivable that this is why identical patterns in `heating`, `cooling`, and `concrete` run-time are observed. As can be seen in Figures 8E–G, the training times for the `redwine`, `whitewine`, and `housing` experiments are as anticipated. A shorter run time in these experiments is accomplished since the fitness evaluation is lower with smaller training data using DBS. The strong trend is evident, and a large number of instances and features in the dataset may be a contributing factor. Figure 8H of the `pollution` experiment shows a pattern for training time that is comparable to that shown in Figure 8D. This may be related to solution size or the fact that

FIGURE 9
Total time taken in test case selection (in seconds) and average time taken per GE run (in minutes) along with std error for digital circuit benchmarks.
(A) Comparator. (B) Parity. (C) Multiplexer. (D) ALU.

the number of instances is low. Figure 8I displays the GE run time for `dowchem` experiment. In the case of `crime` experiments, as in Figure 8J, the GE run time of experiments using different DBS budgets varies arbitrarily by small fractions. This results from the run duration being so short that managing such minute variations is challenging.

The DBS training data, in fact, required less time to train the model in most instances, as can be seen by looking at most of the graphs presented above. This study indicates that training these models on training data selected using DBS is quicker and yields results comparable to or better than those produced in the baseline experiments. This can be particularly advantageous when designing complex systems that inherently require longer computational times.

The training time or GE run analysis for digital circuit benchmarks is presented in Figure 9. The DBS algorithm has a significant contribution in speeding up the evolutionary process and hence has a smaller run time than the baseline. It is evident from the time analysis in Figures 7–9 that test case selection using DBS can greatly reduce the fitness evaluation time, and hence the total GE run time.
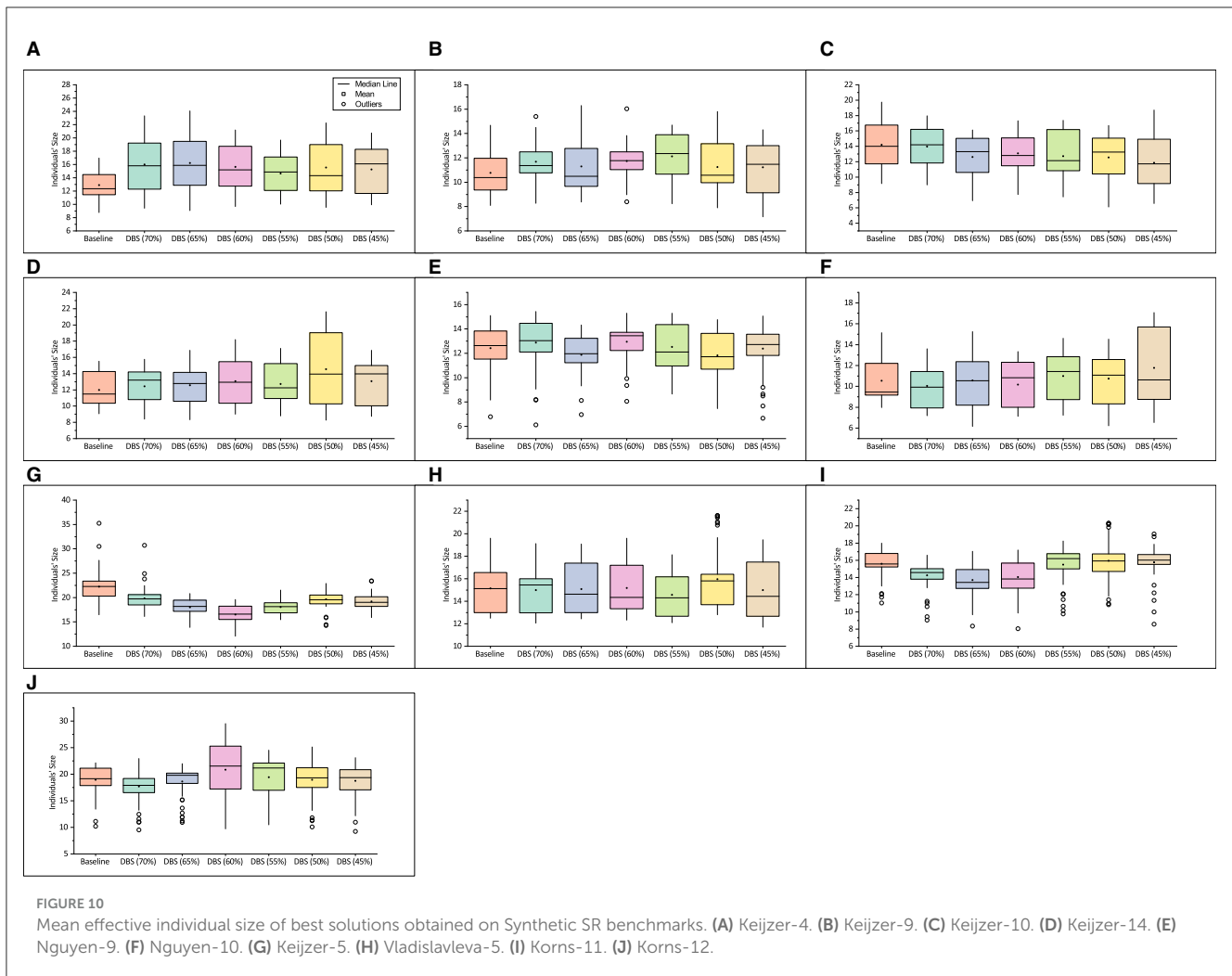
6.4 Solution size

To determine if the training data selected using DBS impacts the size of solutions achieved, we analyze the effective individual

size of the best solutions over 30 independent runs carried out on each instance of training data used on 24 benchmarks. This has several advantages, as it can give insight into their impact on GE run time and whether or not bloat occurs. We use the same approach for all of the benchmarks used.

Figure 10 presents the effective individual size of 10 synthetic SR benchmarks. With the exception of `Keijzer-4` and `Keijzer-14`, the solutions obtained using the DBS training data have effective individual sizes that are either similar to or smaller than the baseline.

Recalling Figures 7A–E, we can observe that the time analysis either shows the expected pattern of lowering run time while decreasing the training data or if it doesn't, the time difference is too small. In the experiment with `Nguyen-10` in Figure 10F, DBS (55%) takes noticeably longer to train than the experiments with bigger training data. The fact that DBS's mean effective individual size at DBS (55%) is larger than that of individuals trained on a higher training budget is one possible explanation. The results for the following experiments are presented in Figures 10G–J. The GE run time graphs for the same in Figures 7G–J show the outcome as anticipated. Usually, a modest variation in effective individual size has less of an influence if the training size, number of features, and GE run-time are large.

Furthermore, we concur that it is challenging to relate how individual size affects the overall training time when the GE run time is very small, but it is conceivable if the difference in individual size is sufficiently significant. Second, we are less inclined to look at

FIGURE 10
Mean effective individual size of best solutions obtained on Synthetic SR benchmarks. **(A)** Keijzer-4. **(B)** Keijzer-9. **(C)** Keijzer-10. **(D)** Keijzer-14. **(E)** Nguyen-9. **(F)** Nguyen-10. **(G)** Keijzer-5. **(H)** Vladislavleva-5. **(I)** Korns-11. **(J)** Korns-12.

instances where GE run time behaves as expected, that is, when it becomes shorter as training data size decreases, but we are more interested in situations where GE run time increases significantly as training data size reduces.

We perform a similar analysis for real-world SR problems, which is shown in Figure 11. In the case of airfoil, the mean effective individual size is similar, and consequently, we observe the GE run time in Figure 8A is somewhat as expected or the run time difference is small in opposite cases. For instance, DBS (50%) and DBS (45%) took slightly more time than DBS (55%) to train the model; the time difference is low. In the case of heating, the mean effective size at DBS (55%) is slightly higher, but a direct impact is not observed in the GE time graph present in Figure 8B. A possible explanation is that as the complexity of the problem increases, a small difference in the size of individuals does not have much impact on training time, provided the training time is sufficiently large. The mean effective individual size in the case of cooling experiment is almost comparable to each other. In the case of concrete dataset, the effective individual sizes obtained in the experiment using DBS (60%), DBS (55%), and DBS (50%) are in increasing order, as shown in Figure 11D. A direct impact of this is observed in the GE run time. Remember, the training

time in Figure 8D is small, and the difference among individual sizes in Figure 11D is large. Conversely, we are less likely to see a direct correlation between individual size and training duration in complex problems.

The mean effective individual size of redwine (in Figure 11E) is comparable to that of their neighbors on the left. However, the effective individual size at DBS (55%) is slightly higher, but as mentioned earlier, the impact of a small difference in individual size is hard to identify on complex problems having small run time differences unless thousands of independent runs are performed. A similar observation is made in whitewine and housing experiments, as shown in Figures 11F, G. The effective individual size between two consecutive training data in pollution experiments is high, as reported in Figure 11H. The impact of large individual size on GE run time is noticeable in Figure 8H. This is possible due to the short GE run time and smaller training data size (see Supplementary Table S5) that causes the impact of individual size to be clearly noticeable. However, this is less possible if experiments using two distinct and sufficiently large training sets produce small size differences among individuals. The effective individual size for dowchem experiments is shown in Figure 11I. DBS training data consistently produced individuals
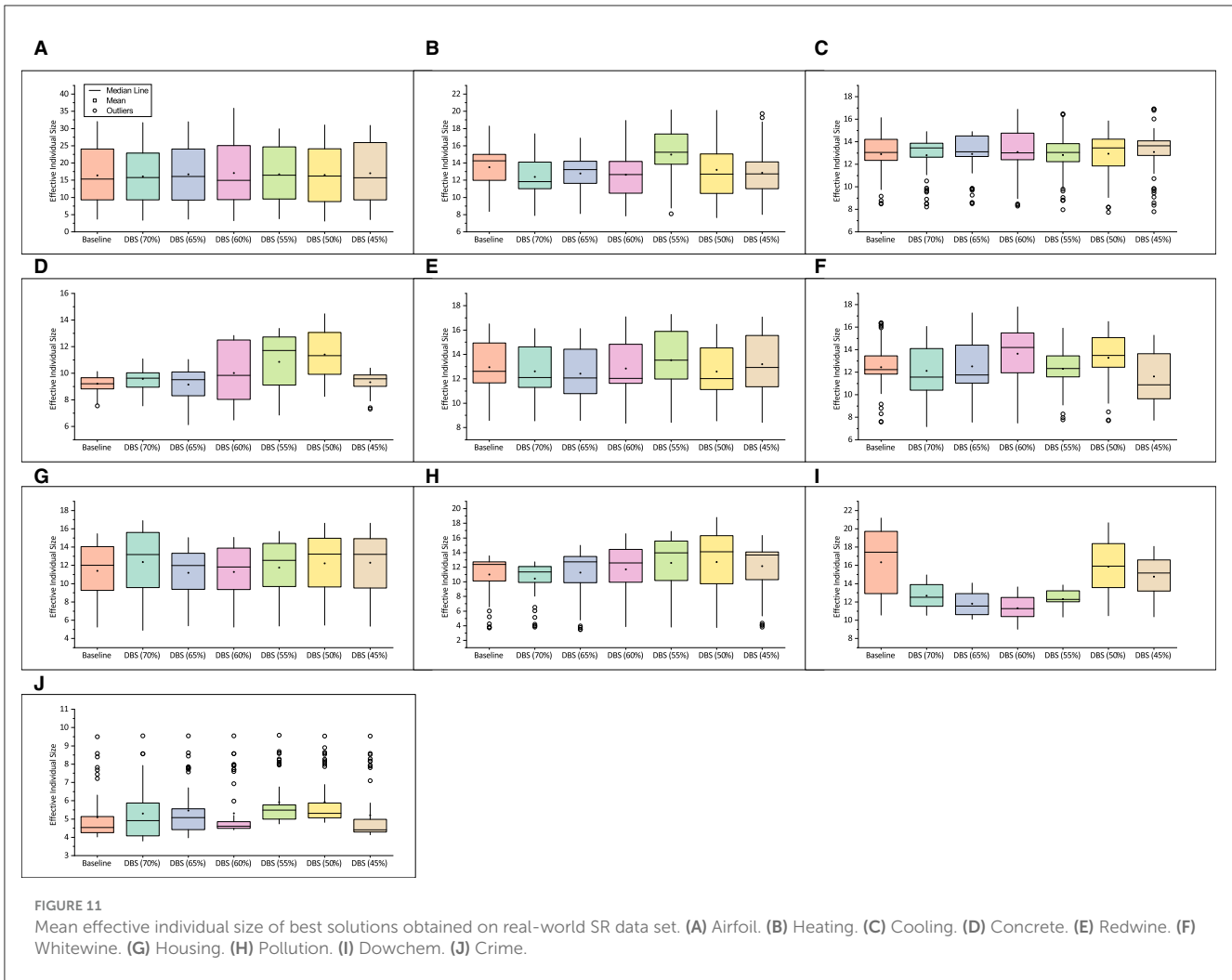
FIGURE 11
Mean effective individual size of best solutions obtained on real-world SR data set. **(A)** Airfoil. **(B)** Heating. **(C)** Cooling. **(D)** Concrete. **(E)** Redwine. **(F)** Whitewine. **(G)** Housing. **(H)** Pollution. **(I)** Dowchem. **(J)** Crime.

that were smaller than the baseline in terms of effective size. Figure 11J shows the individuals' sizes recorded in experiments using different training datasets from `crime` dataset.

In a similar approach, we analyze the mean effective individual size of the best solutions obtained on circuit problems as shown in Figure 12. Since the GE run time in Figure 9 is as expected, the variation in effective individual size has less of an influence if the training size and GE run-time are large, as mentioned previously.

Analyzing Figures 10–12, the training data selected using the DBS approach is good at producing individuals having an effective size comparable to or smaller than the baseline approach in one or more instances of experiments, and thus have fewer chances of producing bloats.

7 Conclusion

We have introduced DBS algorithm to obtain a reduced set of training data for GE that generally produces similar or better results in terms of test case performance but with a substantially lower training cost. The algorithm offers flexibility to define a test case selection budget, which benefits

the users in using the training data size of their choice. We tested the DBS approach on six different budgets. The test results of the solutions achieved using these training data were examined and compared to the corresponding baseline methods.

We evaluated the DBS on 24 benchmarks from the domain of SR and digital circuits. The best solutions of a benchmark trained using different sets of training data in independent instances were tested against a common set of testing data. A graphical visualization of the obtained test scores is presented along with their discussion and statistical tests. In most of the cases, the solutions obtained using DBS training data were similar to or better than the models trained using baseline training data. The statistical test of the results indicates the training data selected using the proposed algorithm covers a wide region of the test case space and, hence, has less scope to compromise the quality of the solution. Time analysis is also provided, and it is observed that, in most instances, the training times for the models employing the DBS training data were comparable to or shorter than the baseline. The effective individual size of the best solution achieved is analyzed. In most of the cases, the mean effective individual size of the solutions was found to be comparable to their corresponding baseline.
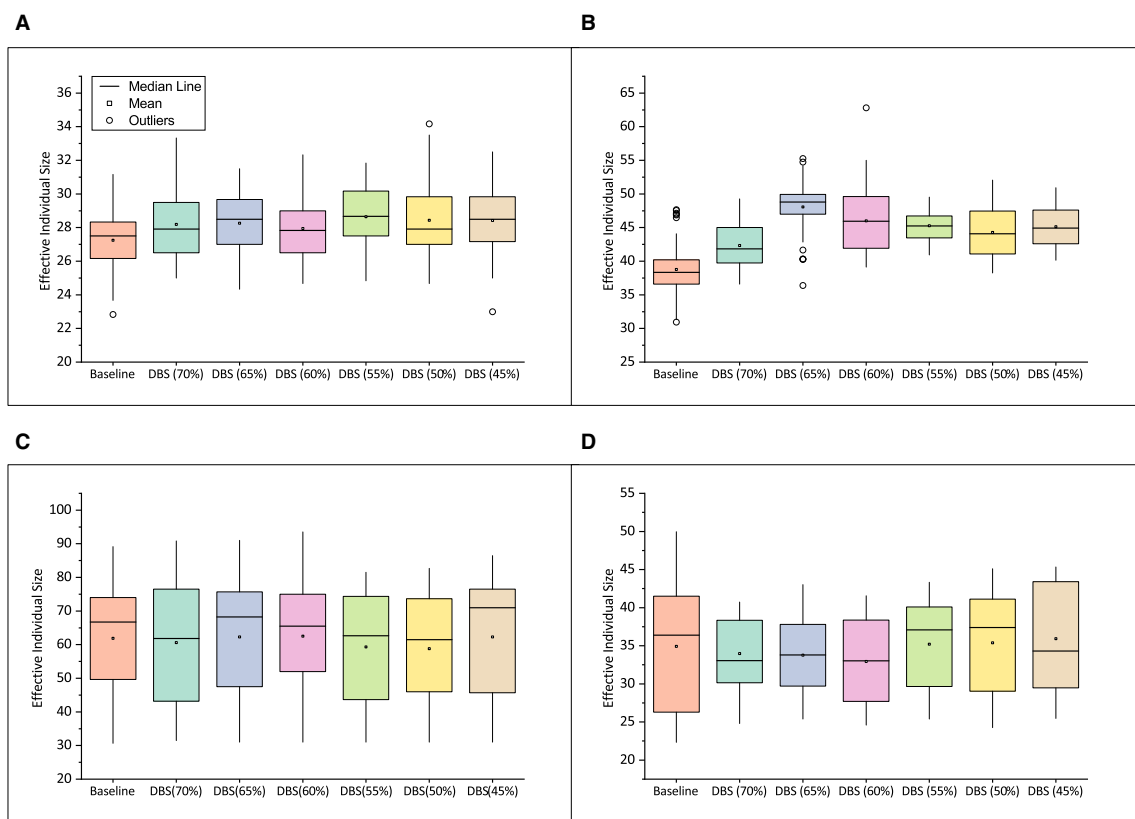
We have not used multi-objective optimization, as it defeats the purpose behind our approach, which is to train the models using minimal yet diverse training cases while being efficient in the utilization of computational resources. Although DBS would benefit multi-objective optimization, the increased number of fitness evaluations would incur additional computational overhead, making it costly for complex problems like evolving digital circuits.

Our results also suggest many novel directions for methodological research in test case selection to train models using EA like GE. Moreover, the experiments using DBS in this paper serve as both an early case study and a platform to facilitate test case selection in other problem domains.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary material, further inquiries can be directed to the corresponding authors.

Author contributions

KG: Data curation, Formal analysis, Methodology, Software, Validation, Visualization, Writing – original draft,

Conceptualization. MK: Supervision, Writing – review & editing, Conceptualization, Methodology, Validation. DD: Writing – review & editing, Supervision, Validation. JS: Funding acquisition, Resources, Supervision, Writing – review & editing, Project administration. CR: Conceptualization, Funding acquisition, Resources, Supervision, Validation, Writing – review & editing, Project administration.

Funding

Acknowledgments

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships

that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be

evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fcomp.2024.1346149/full#supplementary-material

References

Ali, M., Kshirsagar, M., Naredo, E., and Ryan, C. (2021a). "AutoGE: a tool for estimation of grammatical evolution models," in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Vol. 2*, 1274–1281. doi: 10.5220/0010393012741281

Ali, M., Kshirsagar, M., Naredo, E., and Ryan, C. (2021b). "Towards automatic grammatical evolution for real-world symbolic regression," in *Proceedings of the 13th International Joint Conference on Computational Intelligence (IJCCI 2021)*, 68–78. doi: 10.5220/0010691500003063

Ali, M. S., Kshirsagar, M., Naredo, E., and Ryan, C. (2022). "Automated grammar-based feature selection in symbolic regression," in *Proceedings of the Genetic and Evolutionary Computation Conference* (New York, NY: ACM), 902–910. doi: 10.1145/3512290.3528852

Anjum, M. S., and Ryan, C. (2021). Seeding grammars in grammatical evolution to improve search-based software testing. *SN Comput. Sci.* 2, 1–19. doi: 10.1007/s42979-021-00631-7

Arnaiz-González, Á., Díez-Pastor, J.-F., Rodríguez, J. J., and García-Osorio, C. (2016b). Instance selection of linear complexity for big data. *Knowl.-Based Syst.* 107, 83–95. doi: 10.1016/j.knosys.2016.05.056

Arnaiz-González, Á., Díez-Pastor, J. F., Rodríguez, J. J., and García-Osorio, C. (2016a). Instance selection for regression: adapting DROP. *Neurocomputing* 201, 66–81. doi: 10.1016/j.neucom.2016.04.003

Bindra, P., Kshirsagar, M., Ryan, C., Vaidya, G., Gupt, K. K., Kshirsagar, V., et al. (2021). "Insights into the advancements of artificial intelligence and machine learning, the present state of art, and future prospects: seven decades of digital revolution," in *Smart Computing Techniques and Applications*, eds. S. C. Satapathy, V. Bhateja, M. N. Favorskaya, and T. Adilakshmi (Singapore: Springer Singapore), 609–621. doi: 10.1007/978-981-16-0878-0_59

Bushnell, M. L., and Agrawal, V. D. (2002). *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, Volume 17 of Frontiers in Electronic Testing*. Boston, MA: Springer. doi: 10.1007/b117406

Chen, Q., Zhang, M., and Xue, B. (2017). Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. *IEEE Trans. Evol. Comput.* 21, 792–806. doi: 10.1109/TEVC.2017.2683489

Contreras, P., and Murtagh, F. (2015). "Hierarchical clustering," in *Handbook of Cluster Analysis*, eds. C. Hennig, M. Meila, F. Murtagh, and R. Rocci (New York, NY: Chapman and Hall/CRC), 124–145. doi: 10.1201/b19706-11

Czarnowski, I. (2012). Cluster-based instance selection for machine classification. *Knowl. Inf. Syst.* 30, 113–133. doi: 10.1007/s10115-010-0375-z

Czarnowski, I., and Jędrzejowicz, P. (2003). "An approach to instance reduction in supervised learning," in *International Conference on Innovative Techniques and Applications of Artificial Intelligence* (New York, NY: Springer), 267–280. doi: 10.1007/978-0-85729-412-8_20

Czarnowski, I., and Jędrzejowicz, P. (2010). "Cluster integration for the cluster-based instance selection," in *International Conference on Computational Collective Intelligence* (Cham: Springer), 353–362. doi: 10.1007/978-3-642-16693-8_37

de Lima, A., Carvalho, S., Dias, D. M., Naredo, E., Sullivan, J. P., and Ryan, C. (2022). Grape: grammatical algorithms in python for evolution. *Signals* 3, 642–663. doi: 10.3390/signals3030039

Duffy-Deno, K. (2021). *The Curse of Big Data – bintel.io*. Available online at: https://www.bintel.io/blog/the-curse-of-big-data (accessed October 30, 2022).

Ferdjallah, M. (2011). *Introduction to Digital Systems: Modeling, Synthesis, and Simulation using VHDL*. Hoboken, NJ: John Wiley & Sons. doi: 10.1002/9781118007716

Gupt, K. K., Kshirsagar, M., Rosenbauer, L., Sullivan, J. P., Dias, D. M., Ryan, C., et al. (2022a). "Predive: preserving diversity in test cases for evolving

digital circuits using grammatical evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (New York, NY: ACM), 719–722. doi: 10.1145/3520304.3529006

Gupt, K. K., Kshirsagar, M., Sullivan, J. P., and Ryan, C. (2021a). "Automatic test case generation for prime field elliptic curve cryptographic circuits," in *2021 IEEE 17th International Colloquium on Signal Processing & Its Applications (CSPA)* (Langkawi: IEEE), 121–126. doi: 10.1109/CSPA52141.2021.9377300

Gupt, K. K., Kshirsagar, M., Sullivan, J. P., and Ryan, C. (2021b). "Automatic test case generation for vulnerability analysis of galois field arithmetic circuits," in *2021 IEEE 5th International Conference on Cryptography, Security and Privacy, CSP 2021* (Zhuhai: IEEE), 32–37. doi: 10.1109/CSP51677.2021.9357567

Gupt, K. K., Raja, M. A., Murphy, A., Youssef, A., and Ryan, C. (2022b). GELAB – the cutting edge of grammatical evolution. *IEEE Access* 10, 38694–38708. doi: 10.1109/ACCESS.2022.3166115

Hsiung, P.-A., Santambrogio, M. D., and Huang, C.-H. (2018). *Reconfigurable System Design and Verification*. Boca Raton, FL: CRC Press. doi: 10.1201/9781315219035

IBM (2020). *Clustering binary data with K-Means (should be avoided) – ibm.com*. Available online at: https://www.ibm.com/support/pages/clustering-binary-data-k-means-should-be-avoided (accessed October 11, 2022).

Kajdanowicz, T., Plamowski, S., and Kazienko, P. (2011). "Training set selection using entropy based distance," in *2011 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)* (Amman: IEEE), 1–5. doi: 10.1109/AEECT.2011.6132530

Kinnear, K. E., Langdon, W. B., Spector, L., Angeline, P. J., and O'Reilly, U.-M. (1994). *Advances in Genetic Programming, Volume 3*. Cambridge, MA: MIT press.

Kordos, M., and Blachnik, M. (2012). "Instance selection with neural networks for regression problems," in *International Conference on Artificial Neural Networks* (Cham: Springer), 263–270. doi: 10.1007/978-3-642-33266-1_33

Kordos, M., and Łapa, K. (2018). Multi-objective evolutionary instance selection for regression tasks. *Entropy* 20:746. doi: 10.3390/e20100746

Kshirsagar, M., Gupt, K. K., Vaidya, G., Ryan, C., Sullivan, J. P., Kshirsagar, V., et al. (2022). Insights into incorporating trustworthiness and ethics in ai systems with explainable ai. *Int. J. Nat. Comput. Res.* 11, 1–23. doi: 10.4018/IJNCR.310006

Kubalík, J., Derner, E., and Babuška, R. (2020). "Symbolic regression driven by training data and prior knowledge," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (New York, NY: ACM), 958–966. doi: 10.1145/3377930.3390152

Kuhn, D. R., and Okun, V. (2006). "Pseudo-exhaustive testing for software," in *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop, SEW-30* (Columbia, MD: IEEE), 153–158. doi: 10.1109/SEW.2006.26

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge, MA: Cambridge university press. doi: 10.1017/CBO9780511809071

McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., et al. (2012). "Genetic programming needs better benchmarks," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation* (New York, NY: ACM), 791–798. doi: 10.1145/2330163.2330273

Mrozek, I., and Yarmolik, V. (2012). Antirandom test vectors for BIST in hardware/software systems. *Fundam. Inform.* 119, 163–185. doi: 10.3233/FI-2012-732

Mrozek, I., and Yarmolik, V. (2017). "Optimal controlled random tests," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. *10244 LNCS* (Cham: Springer), 27–38. doi: 10.1007/978-3-319-59105-6_3

Murphy, A., Youssef, A., Gupt, K. K., Raja, M. A., and Ryan, C. (2021). "Time is on the side of grammatical evolution," in *2021 International Conference on Computer Communication and Informatics (ICCCI)* (Coimbatore: IEEE), 1–7. doi: 10.1109/ICCCI50826.2021.9402392

Muselli, M., and Liberati, D. (2000). Training digital circuits with hamming clustering. *IEEE Trans. Circuits Syst. I. Fundam. Theory Appl.* 47, 513–527. doi: 10.1109/81.841853

Mutschler, A. (2014). "Yield ramp challenges increase," in *Semiconductor Engineering*. Available online at: https://semiengineering.com/yield-ramp-challenges-increase

Oliveira, L. O. V. B., Martins, J. F. B. S., Miranda, L. F., and Pappa, G. L. (2018). "Analysing symbolic regression benchmarks under a meta-learning approach," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '18* (New York, NY: Association for Computing Machinery), 1342–1349. doi: 10.1145/3205651.3208293

Orso, A., and Rothermel, G. (2014). "Software testing: a research travelogue (2000–2014)," in *Future of Software Engineering Proceedings, FOSE 2014* (New York, NY: Association for Computing Machinery), 117–132. doi: 10.1145/2593882.2593885

Ryan, C., Collins, J. J., and Neill, M. O. (1998). "Grammatical evolution: evolving programs for an arbitrary language," in *European Conference on Genetic Programming* (Berlin: Springer), 83–96. doi: 10.1007/BFb0055930

Ryan, C., Kshirsagar, M., Gupt, K. K., Rosenbauer, L., Sullivan, J. (2021). "Hierarchical clustering driven test case selection in digital circuits," in *Proceedings of the 16th International Conference on Software Technologies – ICSOFT* (Setbal: SciTePress), 589–596. doi: 10.5220/0010605800002992

Ryan, C., O'Neill, M., and Collins, J. J. (2018). *Handbook of Grammatical Evolution*. New York, NY: Springer, 1–497. doi: 10.1007/978-3-319-78717-6

Son, S.-H., and Kim, J.-Y. (2006). "Data reduction for instance-based learning using entropy-based partitioning," in *International Conference on Computational Science and Its Applications* (Cham: Springer), 590–599. doi: 10.1007/11751595_63

Tamasauskas, D., Sakalauskas, V., and Kriksciuniene, D. (2012). "Evaluation framework of hierarchical clustering methods for binary data," in *2012 12th International Conference on Hybrid Intelligent Systems (HIS)* (Pune: IEEE), 421–426. doi: 10.1109/HIS.2012.6421371

Tan, T. S., and Rosdi, B. A. (2014). Verilog HDL simulator technology: a survey. *J. Electron. Test.* 30, 255–269. doi: 10.1007/s10836-014-5449-5

Thamarai, S., Kuppusamy, K., and Meyyappan, T. (2010). Heuristic approach to optimize the number of test cases for simple circuits. *arXiv* [Preprint]. arXiv:1009.6186. doi: 10.48550/arXiv.1009.6186

Wilson, D. R., and Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Mach. Learn.* 38, 257–286. doi: 10.1023/A:1007626913721

Yang, T., McDonough, J., and Jacob, J. (2003). "Parallelization of a genetic algorithm for curve fitting chaotic dynamical systems," in *Parallel Computational Fluid Dynamics 2002*, eds. K. Matsuno, P. Fox, and A. Ecer (Amsterdam: Elsevier), 563–570. doi: 10.1016/B978-044450680-1/50071-1

Youssef, A., Gupt, K. K., Raja, M. A., Murphy, A., and Ryan, C. (2021). "Evolutionary computing based analysis of diversity in grammatical evolution," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)* (Coimbatore: IEEE), 1688–1693. doi: 10.1109/ICAIS50930.2021.9395792